Using FreeRTOS on F&S Boards

Manual on how to use/configuring the software

Version 1.68
(2017-07-05)

Untere Waldplätze 23
D-70569 Stuttgart
Germany

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

About This Document

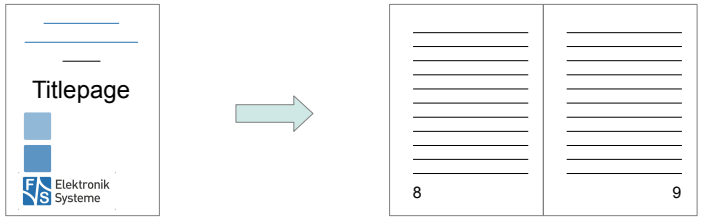
This document describes how to configure the Linux kernel, the device tree and the board to use it with FreeRTOS and its demo applications provided. The software is configured for the efusA9X and PicoCOMA9X from F&S under Linux/Buildroot.

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`

History

Date	V	Platform	A,M,R	Chapter	Description	Au
2017-03-16	0.1	All	A	-	Title and “About this document” part created	TC
2017-03-22	0.2	All	A, M	1,2,3,4	Modified title page (removed logos), Added Introduction, Added information on Installation, Added Configuration on UBoot	TC
2017-03-23	0.3	All	A, M	4,5,7,8	Added Modifying Device Tree chapter, Added FreeRTOS examples chapter, Added General Modifications chapter, Modified Appendix	TC
2017-03-23	0.4	All	A, M	6	Changed flexcan_network_epit subchapter, Added Description for gpio_imx	TC
2017-03-24	0.5	All	A, M	3,5,7	Explicit mentioning of armgcc download link, Changed download link for FreeRTOS, Added legal information, Claryfied path for examples	TC
2017-03-27	0.6	All	M	4,7	Added information on enable_m4() to u-boot modifying section, Changed information on ecspi examples	TC
2017-03-28	0.7	All	A, M	4,7	Added information on changes in uboot regarding dram allocation, Changed information on rpmsg examples	TC
2017-03-30	0.8	All	A	5	Added information header regarding RPMsg	TC
2017-04-03	0.9	All	M	5, 7	Changed “modifications made” section for rpmsg examples, modified attention header to represent current state of development for RPMsg on boards with less than 1 GB RAM.	TC
2017-04-10	0.91	All	A, M	5, 7	Added subchapters to chapter 5.6 to clearify situation on RPMsg, changed “modifications made” part in RPMsg examples in chapter 6	TC
2017-04-12	0.95	All	A	7	Added more information in “changes needed” for the RPMsg pingpong examples	TC
2017-04-19	1.0	All	A, M	2, 3, 4, 7	Changed name for define in subchapter “Protecting modules”, Added new description for i2c examples from NXP, Added own examples, changed some naming conventions for paths, Added attention header for GPIOs	TC
2017-04-21	1.1	All	M	7	Changed command for booting m4, Changed attention header in chapter 3 regarding the current state of development	TC
2017-04-25	1.2	All	A,M	-	Modified title and “About this Document” to reflect changes on PicoCOMA9X	TC
2017-05-05	1.3	All	A, M	2,3,4,5,6, 7	Changed structure of pin assignments chapter; moved the tables from chapter 6 to 2 and modified their content, changed references to tables in chapter 6, mentioned picocoma9x.dts additionally, Added “Known Issues” section to appendix, added information about interfering audio chip and RPMsg	TC
2017-05-08	1.35	All	M	4,5	Changed “Changes in gpio_pins.[ch]” to reflect newes development regarding the configure_gpio_pin() function	TC
2017-05-10	1.4	All	A, M	6	Added subchapter about resetting Cortex-M4, modified information about i2c demo	TC
2017-05-30	1.41	All	M	4	Fixed typo in path to file	TC
2017-05-31	1.5	All	A,M	5,6,7	Changed information on RPMsg VRING allocation addresses to reflect recent changes in the build system, added chapter about new build process, Moved FreeRTOS examples from chapter 6 to 7	TC
2017-06-07	1.6	All	A, M	2, 6	Added title to pin tables to clarify the board name, modified information regarding build/make/install and clean	TC
2017-06-20	1.61	All	M	2	Fixed wrong interface for efusa9x GPIOs	T

						C
2017-06-23	1.65	All	A, M	1,7	Added figures and an informational text about time measurement examples provided by F&S	T C
2017-06-29	1.66	All	A	1	Added figure for performance scaling govenor, added result text at the end of chapter 1	T C
2017-07-03	1.67	All	M	1	Modified screenshots to a more unified measuring scale	T C
2017-07-05	1.68	All	M	1	Swapped images; Trigger set on Input; Input signal is upper one while output is on the lower part of the images	T C

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

Table of Contents

1 Introduction

FreeRTOS is the market leading, de-facto standard Real Time Operating System for embedded systems from Real Time Engineers Ltd. It supports 35 architectures, received over 113000 downloads during 2014 and is professionally developed, strictly quality controlled, robust and free to embed in commercial products without any requirement to expose your proprietary source code.

NXP provides a BSP package (FreeRTOS BSP 1.0.1) with a modified FreeRTOS and some demo and driver examples for their I.MX 6SoloX SABRE-SD and SABRE-AI boards.

The efusA9X and PicoCOMA9X from F&S also have a Cortex-M4 tied to a Cortex-A9, so we ported the BSP package from NXP to our own boards, which makes it possible to easily use the Cortex-M4 with FreeRTOS while running the Cortex-A9 with Linux (Buildroot).

To show the benefits of the heterogenous SoC, F&S created one FreeRTOS and one Linux application, gpio_toggle and tgpio, which just reacts to an interrupt GPIO via polling and toggles an output GPIO. We also measured the jitter with an oscilloscope and provide you with the results in form of screenshots of the measurements and numbers.

The lower signal is the input, the upper one the output signal. Frequency for the input signal was 500 Hz.

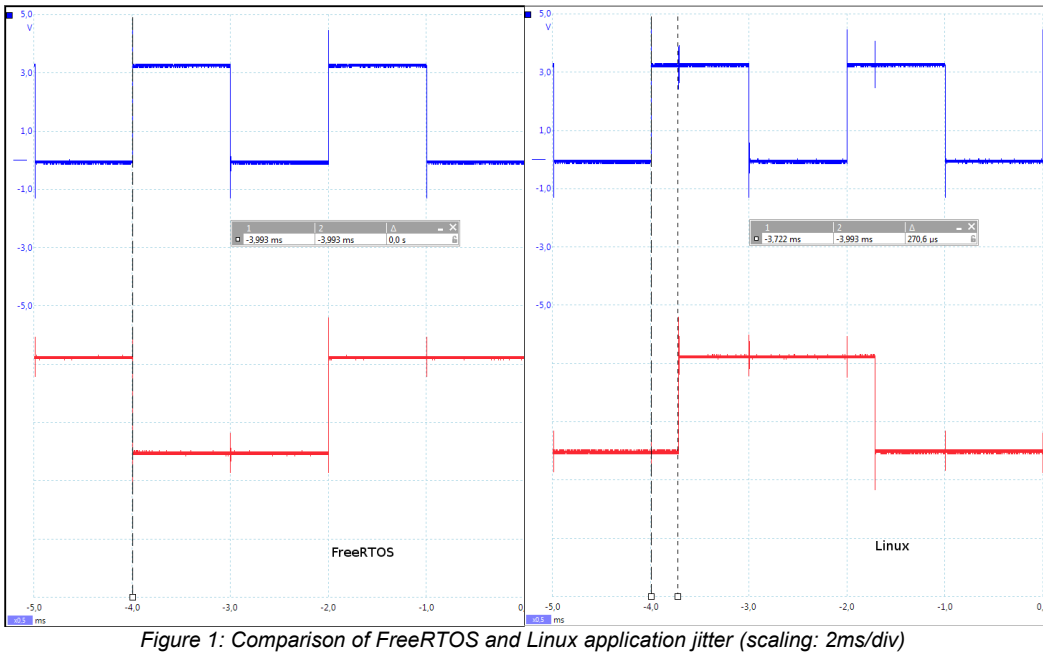


Figure 1: Comparison of FreeRTOS and Linux application jitter (scaling: 2ms/div)

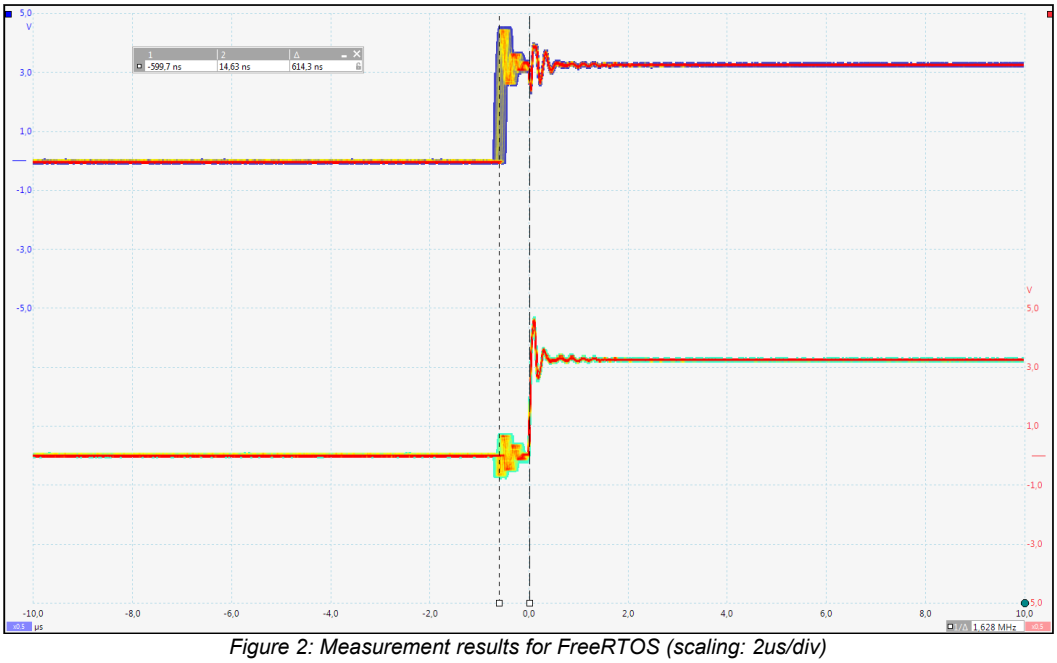


Figure 2: Measurement results for FreeRTOS (scaling: 2us/div)

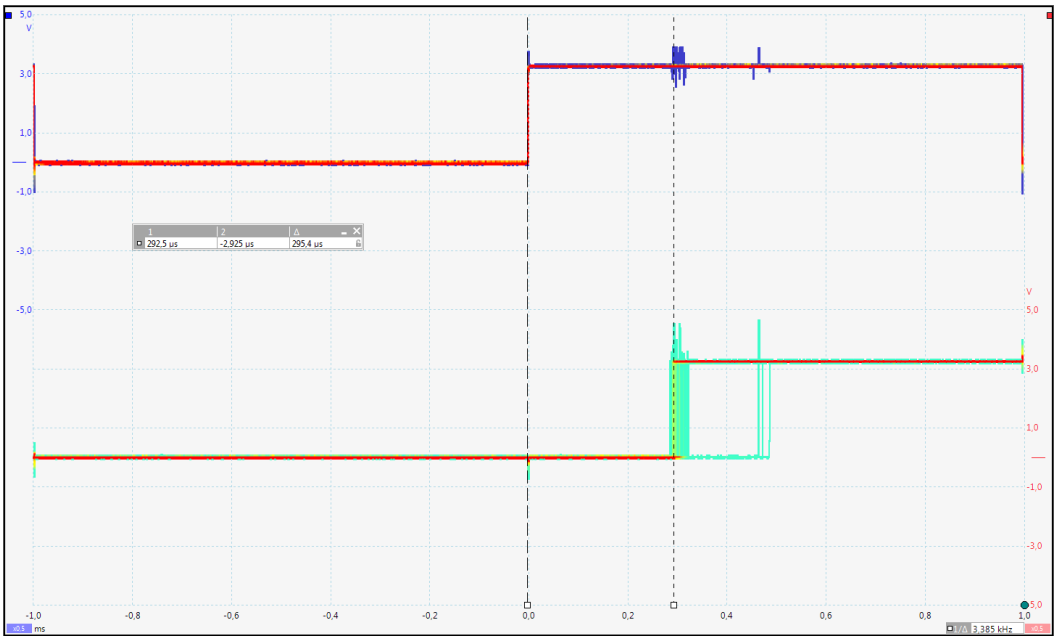


Figure 3: Measurement results for Linux application with interactive as governor (scaling: 200us/div)

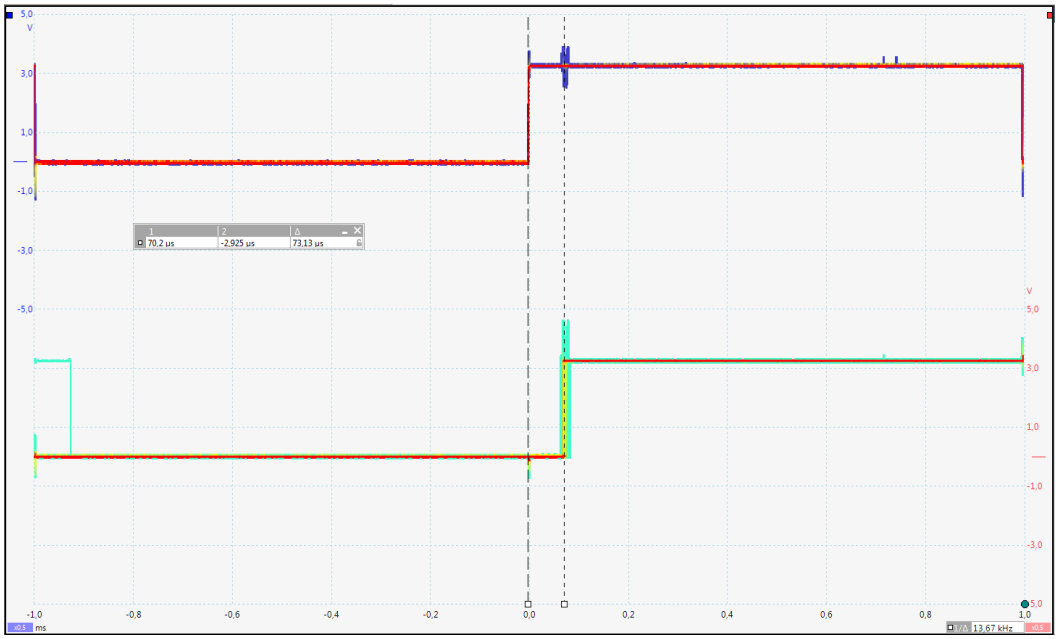


Figure 4: Measurement results for Linux application with performance as governor (scaling: 200us/div)

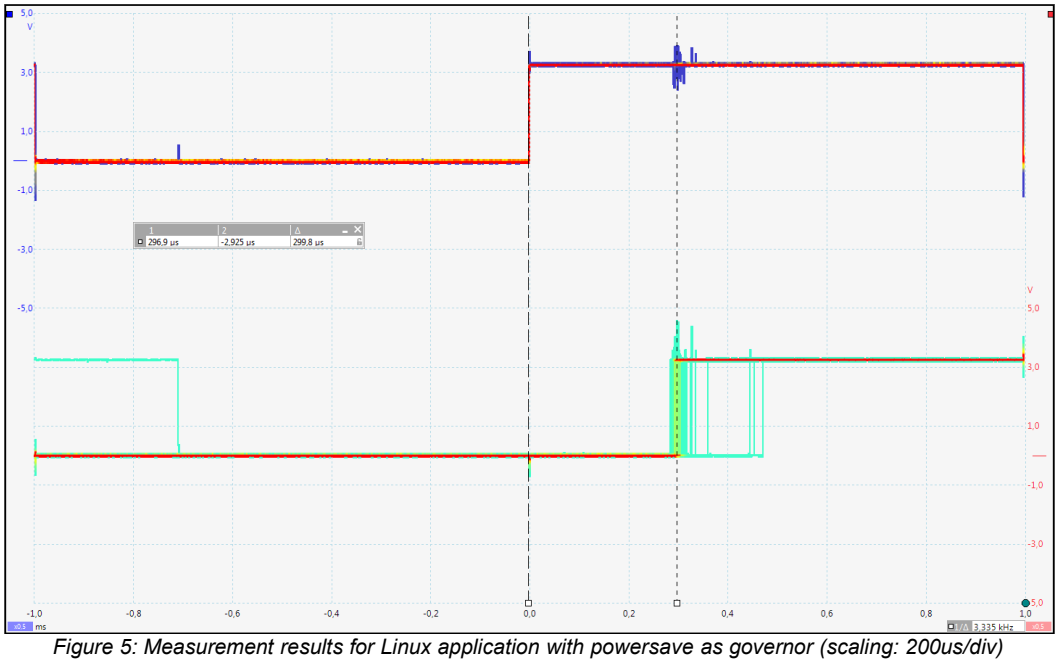


Figure 5: Measurement results for Linux application with powersave as governor (scaling: 200us/div)

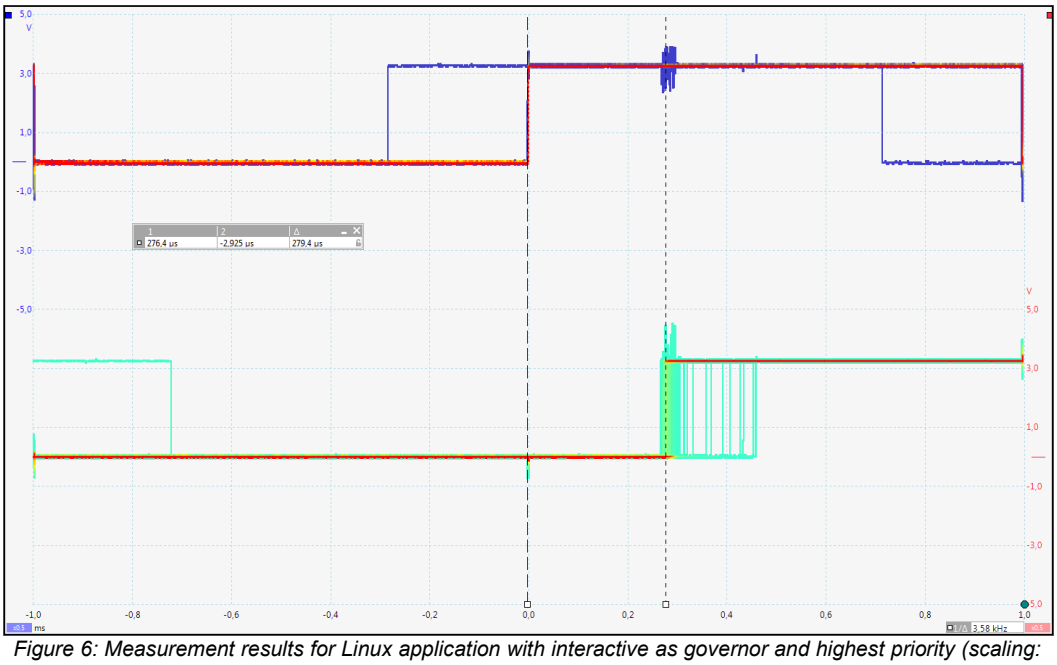


Figure 6: Measurement results for Linux application with interactive as governor and highest priority (scaling: 200us/div)

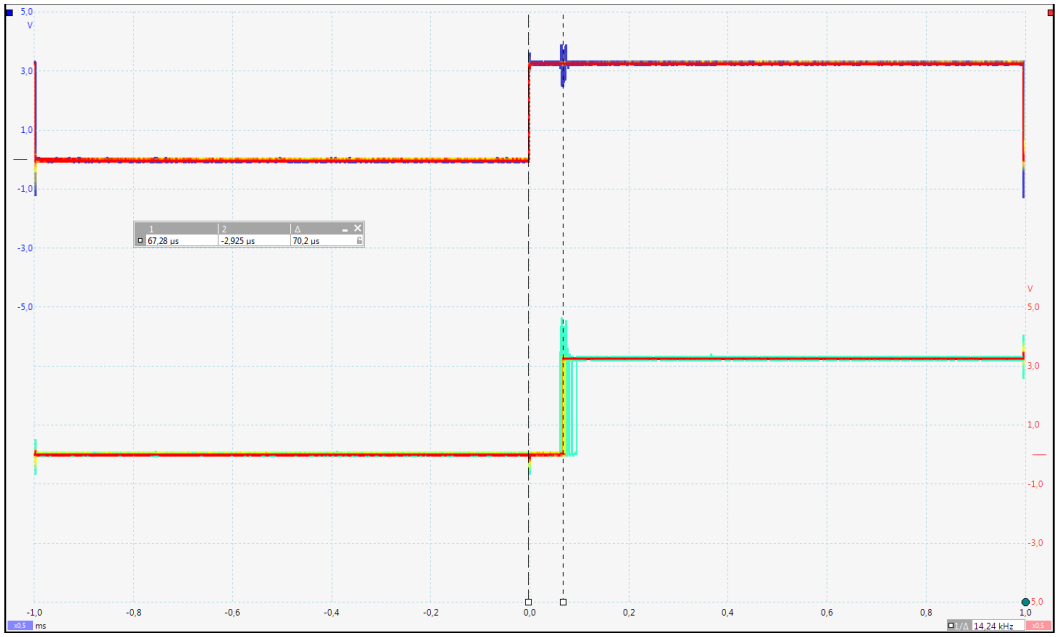


Figure 7: Measurement results for Linux application with performance as governor and highest priority (scaling: 200us/div)

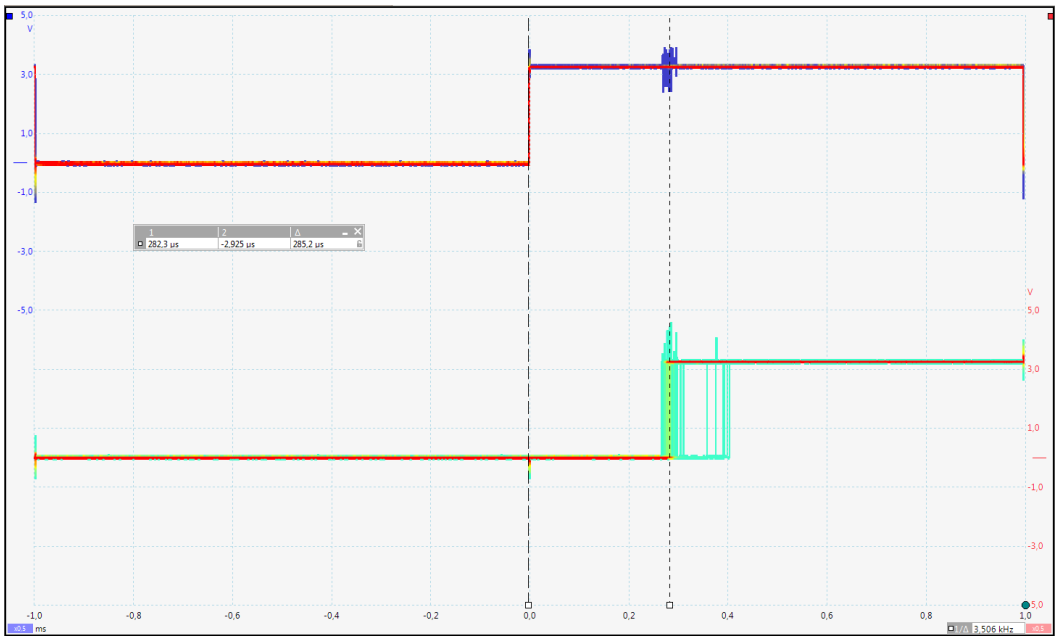


Figure 8: Measurement results for Linux application with powersave as governor and highest priority (scaling: 200us/div)

Results

	Minimal jitter	Maximal jitter	Average jitter
FreeRTOS	482 ns	731 ns	614 ns
Linux, interactive	288 us	484 us	295 us
Linux, powersave	285 us	468 us	299 us
Linux, performance	65 us	86 us	73 us
Linux, interactive, high priority	267 us	460 us	279 us
Linux, powersave, high priority	267 us	406 us	285 us
Linux, perfomance, high priority	61 us	96 us	70 us

Table 1: Measurement results

The FreeRTOS applications jitter took about 614 ns on average with a range from 482 to 731 ns, while the Linux applications average was 295 us with a range from 288 to 484 us. This shows that FreeRTOS operates ~480 times faster on average.

Using “*performance*” as scaling govenor results in 73 us jitter. That's still ~120 times slower than FreeRTOS.

Considering higher priority for the Linux applications results in ~435 and ~115 times slower reaction than FreeRTOS.

The Results show that FreeRTOS combined with an Cortex-M4 is a significant improvement regarding real-time constraints compared to a multi-purpose OS. Measurement was done with one task in FreeRTOS application and highest priority; On Linux we used different scaling governors and lower nice levels (-20).

Remark

The documentation is based on the FreeRTOS BSP 1.0.1 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.

2 Pin Assignments

2.1 GPIOs

For the F&S Boards, the pin assignments for the `gpio_pins.c` differs from the NXP boards. So if you want to use GPIOs for the demo applications, driver examples or your own projects, you can use the preconfigured GPIOs listed in Table 2 and Table 3.

efusA9X

Set as	Name in source file	Function	Device	GPIO	efus-SINTF	Pin
KEY	gpioPwm5	PWM_B	PWM5	GPIO3_IO24	J22	30
LED	gpioPwm6	BL_CTRL	PWM6	GPIO3_IO23	J3	10
	gpioSpiAMiso	SPI_A_MISO	SPI5	GPIO4_IO23	J22	33
	gpioSpiAMosi	SPI_A_MOSI	SPI5	GPIO4_IO20	J22	34
	gpioSpiAClk	SPI_A_CLK	SPI5	GPIO4_IO31	J22	35
	gpioSpiACs1	SPI_A_CS1	SPI5	GPIO4_IO28	J22	36
	gpioSpiACs2	SPI_A_CS2	SPI5	GPIO4_IO18	J22	37
	gpioI2cBSda	I2C_B_SDA	I2C3	GPIO2_IO19	J22	45
	gpioI2cBScl	I2C_B_SCL	I2C3	GPIO2_IO14	J22	46
	gpioCanATx	CAN_A_TX	FLEXCAN1	GPIO7_IO07	J13	3
	gpioCanARx	CAN_A_RX	FLEXCAN1	GPIO7_IO09	J13	4

Table 2: Pin Assignment of preconfigured GPIOs for efusA9X

PicoCOMA9X

Set as	Name in source file	Function	De-vice	GPIO	PCO Mnet	Pin	PC2-SINTF	Pin
KEY	gpioSpiMiso	SPI_MISO	SPI1	GPIO2_IO11	J3	3	J10	3
LED	gpioSpiMosi	SPI_MOSI	SPI1	GPIO2_IO15	J3	6	J10	6
	gpioSpiClk	SPI_CLK	SPI1	GPIO2_IO10	J3	5	J10	5
	gpioSpiCso	SPI_CS0	SPI1	GPIO2_IO16	J3	4	J10	4
	gpioI2cSda	I2C_A_SDA	I2C4	GPIO7_IO03	J3	9	J10	9
	gpioI2cScl	I2C_A_SCL	I2C4	GPIO7_IO2	J3	10	J10	10
	gpioCanTx	CAN_A_TX	CAN1	GPIO7_IO07	J3	13	J10	25
	gpioCanRx	CAN_A_RX	CAN1	GPIO7_IO09	J3	14	J10	26

Table 3: Pin Assignment of preconfigured GPIOs for PicoCOMA9X

Remark

If you want to use the preconfigured GPIOs you have to protect them like stated in 4.2.3 Protecting modules against reconfiguration, otherwise the Cortex-A9 will reconfigure the GPIO.

Attention

Some of the preconfigured GPIOs on the efusA9X are not used as the default devices in the BSP package, but the ones for the PicoCOMA9X are used as SPI, I2C or CAN respectively. If you need them otherwise, you have to use the other preconfigured GPIOs or define your own ones. See the `gpio_pins.c` and `gpio_pins.h` for an example.

Note 1: Information on default assignment of GPIOs

2.2 FlexCAN

The following tables list the Ports used for FlexCAN on the respective starter kits and their pins. Use them for the corresponding examples.

efusA9X

Function	Device	Name in pin_mux.c	efus-SINTF	Pin
CAN_B_TX	FLEXCAN2	CAN2	J22	55
CAN_B_RX	FLEXCAN2	CAN2	J22	56

Table 4: Pin Assignment for CAN2 Pins on efusA9X

PicoCOMA9X

Function	Device	Name in pin_mux.c	PC2-SINTF	Pin	PCOMnet	Pin
CAN_A_TX	FLEXCAN1	CAN1	J10	25	J3	13
CAN_A_RX	FLEXCAN1	CAN1	J10	26	J3	14

Table 5: Pin Assignment for CAN1 Pins on PicoCOMA9X

2.3 ECSPI

The following tables list the Ports used for eCSPI on the respective starter kits and their pins. Use them for the corresponding examples.

efusA9X

Function	Device	Name in pin_mux.c	efus-SINTF	Pin
SPI_B_MISO	SPI1	ECSPI1	J22	23
SPI_B_MOSI	SPI1	ECSPI1	J22	24
SPI_B_CLK	SPI1	ECSPI1	J22	25
SPI_B_CS1	SPI1	ECSPI1	J22	26

Table 6: Pin Assignment for eCSPI on efusA9X

PicoCOMA9X

Function	Device	Name in pin_mux.c	PC2-SINTF	Pin	PCOMnet	Pin
SPI_MISO	SPI1	ECSPI1	J10	3	J3	3
SPI_MOSI	SPI1	ECSPI1	J10	6	J3	6
SPI_CLK	SPI1	ECSPI1	J10	5	J3	5
SPI_CS0	SPI1	ECSPI1	J10	4	J3	4

Table 7: Pin Assignment for eCSPI on PicoCOMA9X

2.4 I2C

The following tables list the Ports used for I2C on the respective starter kits and their pins. Use them for the corresponding examples.

efusA9X

Name	Function	Device	efus-SINTF	Pin
I2C-Clk	I2C_A_CLK	I2C2	J22	42
I2C-Dat	I2C_A_DAT	I2C2	J22	43
I2C-Irq	I2C_A_IRQ	I2C2	J22	44

Table 8: Pin Assignment for I2C2 Pins on efusA9X

PicoCOMA9X

Name	Function	Device	PC2-SINTF	Pin	PCOMnet	Pin
I2C-Clk	I2C_A_SCL	I2C4	J10	9	J3	9
I2C-Dat	I2C_A_SDA	I2C4	J10	10	J3	10

Table 9: Pin Assignment for I2C4 Pins on PicoCOMA9X

I2C-Extension-Board

Name	Connector	Pin
SCL	J1	11
SDA	J1	10
INT	J1	9

Table 10: Pin Assignment on I2C Extension Board

3 Installing Toolchain and FreeRTOS BSP

3.1 Installation of the GCC embedded toolchain

The examples can be build with the GCC embedded toolchain (gcc-arm-none-eabi-5_4_2016q3), which can be found under https://launchpad.net/gcc-arm-embedded.

After downloading the file you can 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 to the rc file of your favourite shell (e.g. zshrc, bashrc, etc.)

3.2 Installation of the FreeRTOS BSP 1.0.1

The FreeRTOS BSP package modified by F&S can be found on the website under:

-- Missing Link, will be added in further release. You should be provided with a bundled tar archive instead --

After you have downloaded the tarball extract it to a directory of your choice via:

```
tar -xvjf FreeRTOS_BSP_1.0.1_iMX6SX_F_S_Vx.y.tar.bz2
```

3.3 Description of the FreeRTOS examples directory structure

3.3.1 demo_apps

Here you can find the applications which highlight certain key features of the ARM Cortex-M4 Core combined with FreeRTOS. The demos can be found under `FreeRTOS_BSP_iMX6SX_F_S_Vx.y/examples/BOARD_NAME/demo_apps`.

3.3.2 driver_examples

You can find simple applications here which are intended to show peripheral drivers working with FreeRTOS in the bare metal environement. The drivers can be found under `FreeRTOS_BSP_iMX6SX_F_S_Vx.y/examples/BOARD_NAME/driver_examples`.

Attention

Examples which make use of onboard sensors, e.g. magnetic field strength measuring were not ported, so use them at your own risk!

The status of the applications will be stated in chapter 7.

Note 2: Information about not ported examples

4 Configuration for Cortex-M4 usage

4.1 Configuring UBoot

F&S provides you with a modified UBoot which can make use of the Cortex-M4. In case you need the source files or if you have to change something, you can follow this instructions to make your UBoot use the M4:

1. add the following lines to `u-boot-f+s/board/F+S/fsimx6sx/fsimx6sx.c`:

```
#define ENABLE_M4 (3<<1) /* add this at line 100 */
```

and

```
void enable_m4(void {
unsigned int val = 0;
struct mxc_ccm_reg *mxc_ccm = (struct mxc_ccm_reg *)
CCM_BASE_ADDR;
val = readl(&mxc_ccm->CCGR3);
val |= ENABLE_M4;
writel(val, &mxc_ccm->CCGR3);
} /* add this function at line 372 */
```

You need to add this functionc all to the *board_init()* function in the same file:

```
enable_m4();     /* add this at line 386 */
```

2. add `cmd_bootaux.c` to `u-boot-f+s/common` from the official UBoot release from NXP.
3. add

```
obj-$(CONFIG_CMD_BOOTAUX) += cmd_bootaux.o
```

to `u-boot-f+s/common/Makefile` at line `230`

4. add

```
#define CONFIG_IMX_BOOTAUX
#define CONFIG_CMD_BOOTAUX
```

to `u-boot-f+s/include/configs/fsimx6sx.h` in section “M4 specific configuration”

5. add

```
#define M4_BOOTROM_BASE_ADDR     0x007F8000
```

to `u-boot-f+s/arch/arm/include/asm/arch-mx6/imx-regs.h`

6. add

```
writel(stack, M4_BOOTROM_BASE_ADDR);
```

```
writel(pc, M4_BOOTROM_BASE_ADDR + 4);
src_reg = (struct src *) SRC_BASE_ADDR;
setbits_le32(&src_reg->scr, 0x00400000);
clrbits_le32(&src_reg->scr, 0x00000010);
```

to `u-boot-f+s/arch/arm/cpu/armv7/mx6/soc.c` at line `377`

7. To get the RPMsg examples up and running with Linux, you have to add the following function to `u-boot-f+s/board/F+S/fsimx6sx/fsimx6sx.c`:

```
/* Initialize the RAM banks and leave space for the two rpmsg
vrings in between (exclude 0xBFFF0000 - 0xC0000000) */
void dram_init_banksize(void)
{
DECLARE_GLOBAL_DATA_PTR;

unsigned int size = gd->ram_size;

if (size > 0x40000000) {
gd->bd->bi_dram[1].start = gd->ram_base + 0x40000000;
gd->bd->bi_dram[1].size = size - 0x40000000;
size = 0x40000000;
} else {
gd->bd->bi_dram[1].start = 0;
gd->bd->bi_dram[1].size = 0;
}
gd->bd->bi_dram[0].start = gd->ram_base;
gd->bd->bi_dram[0].size = size - 0x10000;
}
```

This is needed since *dram_init_banksize()* initializes the whole DRAM before booting Linux which prevents the RPMsg driver from allocating it's shared memory.

4.2 Modifying the Linux Device Tree

4.2.1 Booting Cortex-A9 with Linux while Cortex-M4 is running

To prevent kernel panics while booting the Cortex-A9 the following lines must be added to the device tree (`efusa9x.dts/picocoma9x.dts`) in section *&clks*:

```
fsl,shared-mem-addr = <0x91F000>;
```

```
fsl,shared-mem-size = <0x1000>;
```

This will tell the Cortex-A9 which memory range will be shared between Cortex-A9 and Cortex-M4.

4.2.2 Shared clock for low power mode

Some examples make use of the low power management feature of the Cortex-M4. The Cortex-A9 often takes part in this example by shutting down peripheral resources. Adding the following lines to *&clks*

```
fsl,shared-clks-number = <0x23>;
fsl,shared-clks-index = <IMX6SX_CLK_PLL2_BUS IMX6SX_CLK_PLL2_PFD0
IMX6SX_CLK_PLL2_PFD2 IMX6SX_CLK_PLL3_USB_OTG
IMX6SX_CLK_PLL3_PFD1 IMX6SX_CLK_PLL3_PFD2
IMX6SX_CLK_PLL3_PFD3 IMX6SX_CLK_PLL4_AUDIO
IMX6SX_CLK_PLL5_VIDEO
IMX6SX_CLK_OCRAM IMX6SX_CLK_CAN1_SERIAL
IMX6SX_CLK_CAN1_IPG IMX6SX_CLK_CAN2_SERIAL
IMX6SX_CLK_CAN2_IPG IMX6SX_CLK_CANFD
IMX6SX_CLK_ECSPI1 IMX6SX_CLK_ECSPI2
MX6SX_CLK_ECSPI3 IMX6SX_CLK_ECSPI4
IMX6SX_CLK_ECSPI5 IMX6SX_CLK_QSPI1
IMX6SX_CLK_QSPI2 IMX6SX_CLK_SSI1
IMX6SX_CLK_SSI2 IMX6SX_CLK_SSI3
IMX6SX_CLK_UART_SERIAL IMX6SX_CLK_UART_IPG
IMX6SX_CLK_PERIPH_CLK2_SEL IMX6SX_CLK_DUMMY
IMX6SX_CLK_I2C1 IMX6SX_CLK_I2C2
IMX6SX_CLK_I2C3 IMX6SX_CLK_I2C4
IMX6SX_CLK_EPIT1 IMX6SX_CLK_EPIT2>;
```

will let you use the *SharedClk_EnableNode()* function to tell the Cortex-A9 to NOT shut them down when running in low power mode.

Remark

If you do not need all of the peripherals shared for the Cortex-M4, feel free to just add those needed for your application.

4.2.3 Protecting modules against reconfiguration

Some examples for the Cortex-M4 need peripherals which might be reconfigured by booting up the Cortex-A9 with Linux.

To prevent this you should protect the module by disabling it in the Device Tree (`efusa9x.dts/picocoma9x.dts`). This can be done by changing the modules status by modifying

```
status = "okay";
```

to

```
status = "disabled";
```

This can be enhanced by using a *#define* with a *#if defined* block.

Example

To protect PWM6 against reconfiguration from the linux kernel you have to change the status in the section *&pwm6:*

```
pinctrl-names = "default";
pinctrl-0 = <&pinctrl_pwm6_0>;
status = "okay";
```

to

```
pinctrl-names = "default";
pinctrl-0 = <&pinctrl_pwm6_0>;
#if !defined(GPIO_LED_M4)
      status = "okay";
#else
      status = "disabled";
#endif
```

after that add a define to the `efusa9x.dts/picocoma9x.dts`:

```
#define GPIO_LED_M4   1
```

This will ensure that the GPIO is not configured as PWM6 if you boot the Linux kernel.

Attention

Since the examples use different modules you will get a detailed description which modules to disable in chapter 6 in the corresponding subchapter of the example (section “Changes needed”).

Note 3: Module disabling

4.2.4 Enabling RPMsg node

To run the RPMsg examples you need to add the following section to enable the usage the RPMsg modules inside of Linux:

```
&rpmsg{
      status = "okay";
};
```

If you use the PicocCOMA9X, you additionally have to comment the following line out in the `picocoma9x.dts`:

```
#define CONFIG_PICOCOMA9X_AUDIO
```

5 General Modifications on FreeRTOS examples

The files needed for general modifications (`board.*`, `pin_mux.*`, etc.) can be found under `FreeRTOS_BSP_iMX6SX_F_S_VX.Y/examples/BOARD_NAME`.

5.1 Changes in board.c

F&S boards use a different input clock for the UART than NXP. So we changed line

```
DbgConsole_Init(BOARD_DEBUG_UART_BASEADDR, 24000000, 115200);
```

to

```
DbgConsole_Init(BOARD_DEBUG_UART_BASEADDR, 80000000, 115200);
```

We also removed unnecessary parts in *BOARD_ClockInit()* and *dbg_uart_init()* since some of these settings were already covered by our Nboot.

5.2 Changes in board.h

- We modified the *BOARD_GPIO_KEY_** and *BOARD_GPIO_LED_** defines to comply with our GPIOs on the board.
- The *BOARD_ADC_INPUT_CHANNEL* was changed to Nr. `0` because there is no channel `3` connected to the goldfinger connector on the efusA9X.
- The pin mux configuration was a bit tedious, so we added a macro which can be used in conjunction with

  `/platform/devices/MCIMX6X/include/imx6sx_iomuxc_pins.h`

  which resembles a file from the Linux kernel to set the pad settings. This simplifies the setting of the pads and mux's in `pin_mux.c` and leave the IOMUXC configuration coherent with the Linux ones used by the Cortex-A9

Macro for setting the pad

```
set_iomux(PAD_MUX_SETTINGS, pad_value)

PAD_MUX_SETTINGS .............   Array of 5 values which corresponds
                                 to the iomuxc values stated in
                                 imx6sx_iomuxc_pins.h

pad_value ...................    hex value representing the settings
                                 for the pad register
```

Example

```
/* Setting the TXD for UART1 */
set_iomux(MX6SX_PAD_GPIO1_IO04__UART1_TX, 0x1B0B1);
```

This will set the UART1_TX to 0x1B0B1 in the IOMUXC.

5.3 Changes in gpio_pins.c

We added predefined GPIOs for the blinking_imx_* and other demos. The unused ones were commented out since we modified the ported examples.

The *configure_gpio_pin()* was changed. This decision was made since this code seems a bit confusing; this part should be done in the pin_mux.c because it's similar to setting the pads and mux's in the IOMUXC. We use the *set_iomux()* macro in this function instead.

5.4 Changes in gpio_pins.h

We modified the *_gpio_config* struct to reflect the changes made for the *configure_gpio_pin()* function.

Some of the *extern gpio_config_t* declarations were commented out since we changed the predefined GPIOs.

5.5 Changes in pin_mux.c

Instead of setting the Pads, Mux's and everything else via the NXP defines we make use of the *set_iomux()* macro described in 5.2 Changes in board.h to simplify the whole setup process.

We mainly changed the pads and mux's to the ones used on the efusA9X board from F&S.

5.6 Changes for the RPMsg Protocol

If you want to run the RPMsg examples or use the API for your own applications, follow the steps provided in chapter 4.1, section 7 and chapter 4.2.4.

5.6.1 VRING allocation addresses

On boards with less than 1GB RAM the hardcoded addresses for the VRING allocation won't fit, so you have to change them. This is done by calling the `prepare.sh` script described in the next chapter.

In `/middleware/multicore/open-amp/porting/imx6sx_m4/platform_info.c`

change

```
#define VRING0_BASE   0xBFFF0000
#define VRING1_BASE   0xBFFF8000
```

according to you RAM size, e.g. for 512 MB RAM:

```
#define VRING0_BASE   0x9FFF0000
#define VRING1_BASE   0x9FFF8000
```

Attention

Manually changing the VRING allocation address for the Cortex-M4 side is not needed any more. Use the `prepare.sh` script described in chapter “6.1 Prepare.sh“.

However, the allocation addresses on Linux side have to be changed manually according to the following lines

Note 4: RPMsg

In `linux/arch/arm/mach-imx/imx_rpmsg.c` change line `293` to `294`

```
rpdev->vring[0] = 0xBFFF0000;
rpdev->vring[1] = 0xBFFF8000;
```

to the same value as in the `platform_info.c`, e.g. for 512 MB RAM to

```
rpdev->vring[0] = 0x9FFF0000;
rpdev->vring[1] = 0x9FFF8000;
```

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 Prepare.sh

This script will configure board relevant settings and create symlinks to the board specific header files in `examples/fsimx6sx`. You can execute the script in your terminal by typing

```
./prepare.sh
```

and follow the instructions given by the cli:

```
Choose on of the following boards for which you want to build the examples:
efusa9x[1]   picocoma9x[2]
Enter number in []-brackets for the corresponding board: 2
Symlinks to board specific files created!
Do you want a Release or Debug build?
(r/d) [default: r]: r
Which DRAM size does your board have?
(512/1024) [default: 1024]: 512
If you have a PicoCOMA9X with a fused PCIE_DISABLE, you need
the workaround to disable RDC_* calls, otherwise your CPU will hang
(See documentation, last chapter)
(y/n) [default: n]: y
All set up, starting cmake...
```

Attention

You might have to change the variable *PACKAGE_PATH* inside of `prepare.sh` according to the location of the FreeRTOS BSP on your system!

Note 5: PACKAGE_PATH in prepare.sh

6.2 Make

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

```
make -j4
```

To build all examples located in `examples/fsimx6sx` and install the binaries to `FreeRTOS_BSP_iMX6SX_F_S_Vx.y/bin/$BOARD`.

If you want to build a specific example just type

```
make -j4 example_name && make install/fast
```

to build and install the binary of the choosen example.

Type

```
make help
```

for a list of possible examples for make.

6.3 Clean.sh

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.

7 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.

7.1 General build information

To build any of the following examples, you have to switch in the corresponding `armgcc` folder by executing

```
cd
FreeRTOS_BSP_iMX6SX_F_S_VX.Y/examples/BOARD_NAME/demo_apps/demo_name/armgcc
```

where *X.Y* must be replaced by the packages version and *demo_name* by the name of the demo folder you want to build.

Next, you have to execute

```
./clean.sh && ./build.sh && cp release/demo_name.bin /tftp_dir
```

where *demo_name* must be replaced by the name of the demo application you have built and *tftp_dir* by the path to your tftp folder.

Remark

Some of the demos/drivers have more subdirs so you might have to build more than one application or change the cd command stated above to match the directories location of the armgcc folder.

7.2 Resetting the Cortex-M4 in UBoot

If you use TFTP and change something in your application, it might seem convenient to only reset the Cortex-M4 instead of the whole board. This can be achieved by running the following code segment inside of the UBoot

```
setenv reset_m4 "mw.l 0x20D80000 0xA0481500; tftp ${m4_file};
cp.b $loadaddr 0x7F8000 $filesize; mw.l 0x20D8000 0xA0480508"
```

and then execute

```
run reset_m4
```

to issue a Cortex-M4 Platform Reset, refetching the application from the TFTP server and reloading it on the Cortex-M4.

Remark

The *m4_file* command must be defined for this to work.

7.3 demo_apps

7.3.1 hello_world

Description

Just a simple application which prints a “hello world!” string on the Cortex-M4 side and echoing input back to the user if he is connected to the Cortex-M4 via UART.

Modifications made

Modified board.c; some clock settings are handled by the Nboot, so we removed the corresponding code.

Changes needed

If you want to run this example while booting the Cortex-A9, you have to disable the UART configuration in the Device Tree by changing the following line in *&uart3:*

```
status = "disabled";
```

Execute binary

Run

```
tftp hello_world.bin; cp.b $loadaddr 0x7f8000 $filesize; bootaux
0x7f800
```

to start the example.

Remark

Since the rest of the examples also use the Cortex-M4 you might keep the UART3 disabled.

Using direct addresses in UBoot seems a bit inconvenient. You should add

```
setenv m4 “tftp ${m4_file}; cp.b $loadaddr 0x7f8000 $filesize;
bootaux 0x7f8000

setenv m4_file “hello_world.bin”
```

as environment variables to the UBoot to easily run them via

```
run m4
```

and change `m4_file` according to the example used.

7.3.2 hello_world_ocram

Description

This example doesn't use the TCM but can be launched in OCRAM instead. Since F&S do not provide OCRAM chips on the efusa9x by default you have to use the L2-Cache as OCRAM to run the hello_world_ocram example.

Modifications made

Changed `ORIGIN` in `/platform/devices/MCIMX6X/linker/gcc/MXIMX6X_M4_ocram.ld`

from `0x00910XXX` to `0x00980XXX` so that the L2, if configured as OCRAM, can be used.

Changes needed

none

Execute binary

Run

```
mw.l 0x020e402c 0x2
```

to enable L2 as OCRAM and then start the M4 with

```
tftp hello_world_ocram.bin; cp.b $loadaddr 0x00980000 $filesize;
dcache flush; bootaux 0x00980000
```

7.3.3 hello_world_ddr

Description

Works similar to hello_world_ocram except that it uses DDR instead of OCRAM.

Modifications made

none

Changes needed

none

Execute binary

Run

```
tftp hello_world_ddr.bin; cp.b $loadaddr 0x9ff00000 $filesize;
dcache flush; bootaux 0x9ff00000
```

7.3.4 hello_world_qspi

This example wasn't ported because we do not offer a QSPI chip on the efusA9X and PicoCOMA9X.

If you want to run this example you have to port it by yourself.

7.3.5 blinking_imx_demo

Description

By running this demo you can let a LED blink or print out '+' and '-' with different frequencies by pressing a key connected to gpioPwm5.

Modifications made

Changed `BOARD_GPIO_KEY_* and BOARD_GPIO_LED_*` in `board.h` to the gpios configured in `gpio_pins.c` in `./examples/efusa9x`.

Changed

```
configure_gpio_pin(BOARD_GPIO_KEY_CONFIG);
```

to

```
set_iomux(MX6SX_PAD_LCD1_DATA23__GPIO3_IO_24, 0x30B0);
set_iomux(MX6SX_PAD_LCD1_DATA22__GPIO3_IO_23, 0x30B0);
```

in `hardware_init.c`.

Changes needed

If you do not want to connect a LED to gpioPwm6 you can change the line

```
#define BOARD_GPIO_LED_CONFIG    (&gpioPwm6)
```

to

```
#define BOARD_GPIO_LED_CONFIG    0
```

If you want to boot the Cortex-A9, you need to follow the changes described in chapter 4.2.3 Protecting modules against reconfiguration.

FreeRTOS examples

Execute binary

Run

```
setenv m4_file "blinking_imx_demo_epit.bin"
run m4
```

in UBoot (see 7.3.1 on how to set the environment variable `m4`).

7.3.6 can_wakeup

Description

In this demo application you can see the low power management feature of the Cortex-M4 with CAN in stop-mode. The Cortex-M4 will enter stop-mode after the receiver program is running on the Cortex-M4 and the Cortex-A9 is booted up.

It's possible to send Cortex-A9 into suspend mode by entering

```
echo mem > /sys/power/state
```

Listing 1: suspend Cortex-A9

inside of the Linux system and then start the transmitter program on the other board. This will wake up the Cortex-A9 while the Cortex-M4 is receiving the data sent.

For further information on how to set up the two boards consult the "`Getting_Started_with_FreeRTOS_BSP_for_i.MX_6SoloX.pdf`" located in the `doc` folder in the FreeRTOS BSP package.

Modifications made

Changed `/armgcc/CMakeLists.txt` to set CMAKE_EXE_LINKER_FLAGS_* to TCM instead of the default QSPI. This makes it possible to run the program directly on the Cortex-M4 memory.

Changes needed

We modified the package to use CAN2 instead of the default CAN1. Connect the cables to the corresponding pins on both boards according to Table 4(efusA9X) or Table 5(PicoCOMA9X).

Additionally, you have to disable `&flexcan1` and/or `&flexcan2` in the Device Tree by changing

```
status = okay
```

to

```
status = disabled
```

or protect them like stated in chapter “4.2.3 Protecting modules against reconfiguration(17)”

Execute binary

Run

```
mw.w 0x0091f000 0x0 4; dcache flush
```

to clear the shared memory magic numbers and then

```
setenv m4_file "can_wakeup_rx.bin"
run m4; boot
```

on board one. After the Cortex-A9 has booted up, you can execute Listing 1 on board 1 and change to board 2 and run

```
setenv m4_file "can_wakeup_tx.bin"
run m4
```

You should see sent and received packets on both Cortex-M4 Screens and the Cortex-A9 woken up.

7.3.7 func_gen

Description

Just an utility application for the gpio_toggle demo.

Modifications made

-

Changes needed

BOARD_GPIO_LED provides the output signal

Execute binary

Simply Execute

```
setenv m4_file "func_gen.bin"
run m4
```

7.3.8 gpio_toggle

Description

A demo created by F&S to show the benefits for the real-time aspect of the I.MX6 SoCs. The demo uses *BOARD_GPIO_KEY* as an interrupt-configured input GPIO and *BOARD_GPIO_LED* as a normal output GPIO. Whenever a high signal is detected on the *KEY*, the *LED* pin will toggle. This can be used to measure jitter between the two pins.

Modifications made

-

Changes needed

Connect a dupont cable with a function generator or the *BOARD_GPIO_LED* pin from another board which runs the func_gen example and the *BOARD_GPIO_KEY* from the board which runs this example.

The toggled signal will be provided under the *BOARD_GPIO_LED* pin.

Refer to Table 2 and Table 3 for pin connection information.

Execute binary

After preparing the board and GPIO connections, you should run

```
setenv m4_file "gpio_toggle.bin"
run m4
```

7.3.9 i2c_extension_board_demo

Description

This demo shows how to use I2C to control devices via master/slave-communication. The demo was designed by F&S to use the I2C Extension Board with FreeRTOS and a PCA9555 driver provided by F&S instead of onboard sensors. You can choose between a polling and interrupt demo on boot up.

Modifications made

A PCA9555 driver was implemented by F&S, the main.c was also reworked to communicate with the I2C Extension Board. The key for switching the LEDs has been debounced.

Changes needed

You have to connect the I2C pins on the StarterKit with the corresponding ones on the I2C Extension Board:

I2C-Clk with SCL,

I2C-Irq with INT and

I2C-Dat with SDA.

Refer to Table 8(efusA9X), Table 9(PicoCOMA9X) and Table 10(I2C Extension Board) for locating the connection pins on the two boards.

If you want to try the interrupt demo you will need to connect a button to the default *BOARD_GPIO_KEY*. See Table 2 for more information on GPIO pins.

Execute binary

Connect the pins on the I2C Extension Board and the StarterKit like stated in the paragraph above and run

```
setenv m4_file "i2c_extension_demo_imx6sx.bin"
run m4
```

After the demo launched you will get the following screen:

```
---------- i.MX 6SoloX i2c extension board example -----------


Please select the i2c demo you want to run:
[1].PCA9555 I2C Extension Board Polling Demo
[2].PCA9555 I2C Extension Board Interrupt Demo
-----------------------------------------
```

Now you can choose between a polling and interrupt version by pressing the corresponding key on the keyboard.

If you press `'1'`, you will get

```
---------- i.MX 6SoloX i2c extension board example -----------


Please select the i2c demo you want to run:
[1].PCA9555 I2C Extension Board Polling Demo
[2].PCA9555 I2C Extension Board Interrupt Demo
-----------------------------------------
Your choice: Polling Demo

The LEDs have been successfully turned off!
The LEDs have been successfully turned on!
```

```
The LEDs have been successfully turned off!
The LEDs have been successfully turned on!
The LEDs have been successfully turned off!
```

And the LEDs on the I2C Extension board will be toggled every 1 s.

If you press `'2'`, you will get

```
---------- i.MX 6SoloX i2c extension board example -----------


Please select the i2c demo you want to run:
[1].PCA9555 I2C Extension Board Polling Demo
[2].PCA9555 I2C Extension Board Interrupt Demo
----------------------------------------
Your choice: Interrupt Demo

The LEDs have been successfully turned off!
The LEDs have been successfully turned on!
```

And the LEDs will be toggled if you press the button connected to the *gpioPwm5*.

7.3.10 periodic_wfi_tcm

Description

This demo also highlights the low power management feature of the Cortex-M4 by setting himself in sleep mode, informing Cortex-A9 about its power state who then can shut down all peripherals which are not needed.

Modifications made

Commented out lines `222 to 223` in `main.c` since the demo now uses TCM instead of QSPI.

Changed CMAKE_EXE_LINKER_FLAGS_* to use `*tcm_lowpower.ld` instead of `*qspi2b.ld`

Changes needed

none

Execute binary

Run

```
setenv m4_file "periodic_wfi.bin"
run m4
```

to kick off the demo.

7.3.11 pingpong_bm

Description

The Master peer on Linux side sends an integer to the Cortex-M4 application, which adds one and transfers it back. This demo works on bare metal base.

Modifications made

Added

```
LMEM_FlushSystemCache(LMEM);
LMEM_InvalidateSystemCache(LMEM);
```

to `middleware/multicore/open-amp/rpmsg/rpmsg_core.c`, `multicore/open-amp/porting/env/bm/rpmsg_porting.c/` and `rtos/FreeRTOS/Source/queue.c`

Changes needed

You might have to change the allocation address for the VRINGs and the *BOARD_SIZE*. See chapter 5.6 for more information.

Execute binary

First run

```
setenv m4_file "rpmsg_pingpong_bm_example.bin"
run m4; boot
```

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

```
modprobe imx_rpmsg_pingpong
```

to load the pingpong master side module. After this you should see

```
Get Data From Master Side : 0
Get Data From Master Side : 2
Get Data From Master Side : 4
```

on Cortex-M4 and

```
get 1 (src: 0x0)
get 3 (src: 0x0)
get 5 (src: 0x0)
```

FreeRTOS examples

on Cortex-A9 side.

7.3.12 pingpong_freertos

Description

The Master peer on Linux side sends an integer to the Cortex-M4 application, which adds one and transfers it back. The FreeRTOS RPMsg API is used here.

Modifications made

See Modifications made in subchapter 7.3.11.

Changes needed

You might have to change the allocation address for the VRINGs and the *BOARD_SIZE*. See chapter 5.6 for more information.

Execute binary

First run

```
setenv m4_file "rpmsg_pingpong_freertos_example.bin"
run m4; boot
```

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

```
modprobe imx_rpmsg_pingpong
```

to load the pingpong master side module. After this you should see

```
Get Data From Master Side : 0
Get Data From Master Side : 2
Get Data From Master Side : 4
```

on Cortex-M4 and

```
get 1 (src: 0x0)
get 3 (src: 0x0)
get 5 (src: 0x0)
```

on Cortex-A9 side.

7.3.13 str_echo_bm

Description

This demo demonstrate the RPMsg extension API by creating a channel from Cortex-A9 to Cortex-M4 via /dev/ttyRPMSG. After initialization, you can enter a string which then will be replied back and can be read from /dev/ttyRPMSG. This demo works on bare metal base.

Modifications made

See Modifications made in subchapter 7.3.11.

Changes needed

You might have to change the allocation address for the VRINGs. See chapter 5.6 for more information.

Execute binary

First run

```
setenv m4_file "rpmsg_str_echo_bm_example.bin"
run m4; boot
```

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

```
modprobe imx_rpmsg_tty
```

to load the tty master side module.

Now you can echo content via /dev/ttyRPMSG to the Cortex-M4, which then will reply this back to said device:

```
echo 'Test' > /dev/ttyRPMSG && read x < /dev/ttyRPMSG && echo $x
```

This will send the string “Test” to the Cortex-M4, read out /dev/ttyRPMSG again to a variable and print this in the terminal.

7.3.14 str_echo_freertos

Description

This demo demonstrate the FreeRTOS RPMsg extension API by creating a channel from Cortex-A9 to Cortex-M4 via /dev/ttyRPMSG. After initialization, you can enter a string which then will be replied back and can be read from /dev/ttyRPMSG. This demo uses the FreeRTOS RPMsg API.

Modifications made

See Modifications made in subchapter 7.3.11.

Changes needed

You might have to change the allocation address for the VRINGs. See chapter 5.6 for more information.

Execute binary

First run

```
setenv m4_file "rpmsg_str_echo_freertos_example.bin"
run m4; boot
```

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

```
modprobe imx_rpmsg_tty
```

to load the tty master side module.

Now you can echo content via /dev/ttyRPMSG to the Cortex-M4, which then will reply this back to said device:

```
echo 'Test' > /dev/ttyRPMSG && read x < /dev/ttyRPMSG && echo $x
```

This will send the string “Test” to the Cortex-M4, read out /dev/ttyRPMSG again to a variable and print this in the terminal.

7.3.15 sema4_demo

Description

Simple demo which shows how to implement a multicore mutex without spinning with CPU.

Modifications made

none

Changes needed

none

Execute binary

Run

```
setenv m4_file "sema4_demo.bin"
run m4
```

to kick off the demo.

7.3.16 sensor_demo

This example wasn't ported because the efusA9X does not provide any onboard sensors.

You can still find the demo in

`/examples/efusa9x/demo_apps/not_tested/sensor_demo`,

but you should use the `i2c_extension_board_demo` with the I2C Extension Board from F&S instead.

7.4 driver_examples

7.4.1 adc_imx6sx

Description

This example demonstrate the usage of the ADC on the efusA9X by measuring the AD input on the ADC1_IN0, converting and printing the result every 5 second to the console.

Modifications made

none

Changes needed

The efusA9X does not have the RN3 resistor network (22R) connected on the board by default, so keep in mind to add this if you want to run the example.

Execute binary

Run

```
setenv m4_file "adc_imx6sx_example.bin"
run m4
```

7.4.2 ecspi_interrupt

Description

In this example the master board transfers an array to the slave board, which is then send back to the master. This demo uses interrupts.

Modifications made

Inserted function calls

```
ECSPI_SetDataInactiveState(BOARD_ECSPI_MASTER_BASEADDR,
BOARD_ECSPI_MASTER_CHANNEL, ecspiDataLineStayLow);
```

and

```
ECSPI_SetDataInactiveState(BOARD_ECSPI_SLAVE_BASEADDR,
BOARD_ECSPI_SLAVE_CHANNEL, ecspiDataLineStayLow);
```

to the master and slave `main.c` to keep MOSI low between assertion of CS and starting og the clock.

Changes needed

You have to connect the masters MOSI, MISO, CLK and CS1 ping with their counterparts on the slave board (MOSI → MOSI, MISO → MISO, CLK → CLK and CS1 → CS1).

Refer to Table 6(efusA9X) and Table 7(PicoCOMa9X) for the correct pins.

Execute binary

First run

```
setenv m4_file "ecspi_interrupt_master_example.bin"
run m4
```

on the master board. Wait for the following line to appear on the terminal connected to the Cortex-M4:

```
Press "s" when spi slave is ready.
```

Now connect to the slave board and run

```
setenv m4_file "ecspi_interrupt_slave_example.bin"
run m4
```

If the demo kicks off on the slave board, press “s” on the terminal connected to the masters board Cortex-M4. Now you should see data transmitted between the two boards.

7.4.3 ecspi_polling

Description

In this example the master board transfers an array to the slave board, which is then send back to the master. This demo uses the polling mode for achieving it's goal.

Modifications made

Inserted function calls

```
ECSPI_SetDataInactiveState(BOARD_ECSPI_MASTER_BASEADDR,
BOARD_ECSPI_MASTER_CHANNEL, ecspiDataLineStayLow);
```

and

```
ECSPI_SetDataInactiveState(BOARD_ECSPI_SLAVE_BASEADDR,
BOARD_ECSPI_SLAVE_CHANNEL, ecspiDataLineStayLow);
```

to the master and slave `main.c` to keep MOSI low between assertion of CS and starting og the clock.

Changes needed

See 7.4.2 ecspi_interrupt for information on wiring the two boards.

Execute binary

First run

```
setenv m4_file "ecspi_polling_master_example.bin"
run m4
```

on the master board. Wait for the following line to appear on the terminal connected to the Cortex-M4:

```
Press "s" when spi slave is ready.
```

Now connect to the slave board and run

```
setenv m4_file "ecspi_polling_slave_example.bin"
run m4
```

If the demo kicks off on the slave board, press “s” on the terminal connected to the masters board Cortex-M4. Now you should see data transmitted between the two boards.

Attention

There is a bug inside the iMX6 which prevents the usage of burstlengths greater than [(32 * n) + 1], so don't set it to more than 32!

Note 6: Maximum Burstlength

7.4.4 epit

Description

Simple application demonstrating two different EPIT instances (EPIT1 and EPIT2) cathing each others counter every 0.5 s.

Modifications made

none

Changes needed

none

Execute binary

Run

```
setenv m4_file "epit_example.bin"
run m4
```

to kick off the demo.

7.4.5 flexcan_loopback_epit

Description

This example demonstrates the FlexCAN module loopback operating mode by sending data from the tx message buffer to its own rx message buffer.

Modifications made

none

Changes needed

Disable `&flexcan1` and/or `&flexcan2` in the Device Tree by changing

```
status = okay
```

to

```
status = disabled
```

in `efusa9x.dts` if you want to boot up the Cortex-A9 parallel to the Cortex-M4.

Execute binary

Run

```
setenv m4_file "flexcan_loopback_epit_example.bin"
run m4
```

7.4.6 flexcan_network_epit

Description

Like in the can_wakeup demo this one will send data packets over the CAN bus between two boards.

Modifications made

none

Changes needed

Please refer to Table 4(efusA9X) or Table 5(PicoCOMA9X) for the necessary pin configurations.

Connect CAN_*_TX to the corresponding CAN_*_TX on the second board and do the same with CAN_*_RX.

You need to compile two versions of the software. The first one need

```
#define NODE    1
```

the other must be built with

```
#define NODE    2
```

Save each `*.bin` under a different name (like `flexcan_network_epit_example_b1.bin`) before transfering them to the boards RAM.

Execute binary

Run

```
setenv m4_file "flexcan_network_epit_example_b1.bin"
run m4
```

on board 1 and

```
setenv m4_file "flexcan_network_epit_example_b2.bin"
run m4
```

on the other one.

7.4.7 gpio_imx

Description

This simple application shows how to use LED, buttons, etc. connected to the board via the GPIO interface.

Modifications made

Changed

```
configure_gpio_pin(BOARD_GPIO_KEY_CONFIG);
```

to

```
set_iomux(MX6SX_PAD_LCD1_DATA23__GPIO3_IO_24, 0x30B0);
set_iomux(MX6SX_PAD_LCD1_DATA22__GPIO3_IO_23, 0x30B0);
```

Changes needed

Connect a LED and a Button to the GPIOs configured in Table 1 (see 7).

BOARD_GPIO_KEY_CONFIG and *BOARD_GPIO_LED_CONFIG* must be properly configured (see “Changes Needed” in 27).

Execute binary

Run

```
setenv m4_file "gpio_imx_example.bin"
run m4
```

7.4.8 i2c_interrupt_extension_board_imx6sx

Description

A sample application which uses the FreeRTOS I2C API to let the board communicate as a master with other i2c slaves. It will configure the I2C Extension Board via I2C and then start a chaser light on it to check if the configuration was successful. This application uses interrupts.

Modifications made

Used the `i2c_interrupt_sensor_imx6sx` as a template and changed the purpose and functionality.

Changes needed

See 7.3.9 i2c_extension_board_demo, section `“Changes Needed”` and Table 8(efusA9X), Table 9(PicoCOMA9X) and Table 10(I2C Extension Board) for the required pin connections for the I2C bus.

Execute binary

Connect the pins as stated in “Changes needed”, then run

```
setenv m4_file
"i2c_imx_interrupt_extension_board_imx6sx_example.bin"
run m4
```

to kick of the demo. You will see the following screen and a chaser light on the I2C Extension Board, which goes from the left to the right LED and then vice versa:

```
+++++++++++++++ I2C Send/Receive interrupt Example ++++++++++++++
This example will configure the i2c extension board through I2C
Bus
```

```
and run a chaser light to see if the i2c extension board was
properly configured.
[1]. Initialize the I2C module with initialize structure.
[2]. Clear input data polarity, so that it will be retained
[3]. Configure Ports as outputs
[4]. Set PCA9555 output port 1 to 0x1
[5]. Start chaser light

Example finished!!!
```

7.4.9 i2c_polling_extension_board_imx6sx

Description

A sample application which uses the FreeRTOS I2C API to let the board communicate as a master with other i2c slaves. It will configure the I2C Extension Board via I2C and then start a chaser light on it to check if the configuration was successful. This application uses polling.

Modifications made

Used the `i2c_interrupt_sensor_imx6sx` as a template and changed the purpose and functionality.

Changes needed

See 7.3.9 i2c_extension_board_demo, section "`Changes Needed`" and Table 8(efusA9X), Table 9(PicoCOMA9X) and Table 10(I2C Extension Board) for the required pin connections for the I2C bus.

Execute binary

Connect the pins as stated in "Changes needed", then run

```
setenv m4_file
"i2c_imx_polling_extension_board_imx6sx_example.bin"
run m4
```

to kick of the demo. You will see the following screen and a chaser light on the I2C Extension Board, which goes from the left to the right LED and then vice versa:

```
++++++++++++ I2C Send/Receive polling Example +++++++++++++++
This example will configure the i2c extension board through I2C
Bus
```

```
and run a chaser light to see if the i2c extension board was
properly configured.
[1]. Initialize the I2C module with initialize structure.
[2]. Clear input data polarity, so that it will be retained
[3]. Configure Ports as outputs
[4]. Set PCA9555 output port 1 to 0x1
[5]. Start chaser light

Example finished!!!
```

7.4.10 i2c_interrupt_sensor_imx6sx

This example wasn't ported because the efusA9X does not provide any onboard sensors.

You can still find the demo in

`/examples/efusa9x/driver_examples/not_tested/i2c_interrupt_sensor_imx6sx,`

but you should use the `i2c_interrupt_extension_board_imx6sx` with the I2C Extension Board from F&S instead.

7.4.11 i2c_polling_sensor_imx6sx

See “6.2.8 i2c_interrupt_sensor_imx6sx” for more information.

Use `i2c_polling_extension_board_imx6sx` instead.

7.4.12 uart_polling

Description

This example works similar to the hello_world one except that it only echos input

and uses a polling interface.

Modifications made

none

Changes needed

If you want to run this example while booting the Cortex-A9, you have to disable the UART configuration in the Device Tree by changing the following line in *&uart3:*

```
status = "disabled";
```

Execute binary

Run

```
setenv m4_file "uart_imx_polling_example.bin"
run m4
```

7.4.13 uart_interrupt

Description

This example works similar to the hello_world one except that it only echos input

and uses interrupts.

Modifications made

none

Changes needed

If you want to run this example while booting the Cortex-A9, you have to disable the UART configuration in the Device Tree by changing the following line in *&uart3:*

```
status = "disabled";
```

Execute binary

Run

```
setenv m4_file "uart_imx_interrupt_example.bin"
run m4
```

7.4.14 wdog_imx

Description

Simple demo which enables WDOG with 1.5s timeout and a interrupt is triggered to refresh the WDOG timer four times.

Modifications made

none

Changes needed

none

Execute binary

Run

```
setenv m4_file "wdog_imx_example.bin"
run m4
```

8 Appendix

List of Figures

List of Tables

Listings

Appendix

Known Issues

As stated in the IMX6SXCE, Rev. 1 from 04/2016, there is an issue with some of the I.MX6 SoloX chips solded on the PicoCOMA9X. This prevents the usage of all RDC_* calls from the FreeRTOS API, otherwise the Cortex-A9 or Cortex-M4 will hang.

This occur on chips with date code lesser than 1524. Apparently, there is no known solution to this problem. Contact F&S for more information.

See http://cache.freescale.com/files/32bit/doc/errata/IMX6SXCE.pdf?fasp=1&WT_TYPE=Errata&WT_VENDOR=FREESCALE&WT_FILE_FORMAT=pdf&WT_ASSET=Documentation&fileExt=.pdf for a detailed description on this.

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.