SMARC-iMX8MM_mcuxpresso_sdk_2.8.0-8mm

Build and Install FreeRTOS for *SMARC-iMX8MM* (*Cortex-M4* core)

This document provides instructions for advanced users how Embedian offers patches and builds a customized version of FreeRTOS for Embedian's *SMARC-iMX8MM* product platform and how to install the images to bring the evaluation board up and running.

Our aim is to fully support our hardware through device drivers. We also provide unit tests so that testing a board is easy and custom development can start precisely. The recommended host environment is Ubuntu 16.04.

Before using this guide, we assume you understand how to setup a Yocto, Debian or Ubuntu working SD card. This guild apply to U-boot 2019.04 and Linux kernel 4.19.35 or higher. Here we are not mention Yocto/Debian/Ubuntu setup and compilation. There are well document available from Embedian.

There are several options in both processors, code can be located in one of the following:

- TCM (Tightly Coupled Memory): 128kB available
- DDR: up to 1MB available (can be increased, set in the device tree)

Note that the TCM is the preferred option when possible since it offers the best performances since it is an internal memory dedicated to the Cortex-M4.

External memories, such as the DDR, offer more space but are also much slower to access.

In this article, it is assumed that every application runs from the TCM.

Availability

SMARC-iMX8MM from Embedian

Carrier Board

EVK-STD-CARRIER-S20 (universal carrier board for all SMARC 2.0 modules) from Embedian

Basic Resources

- AArch64 Cross Compiler
  - Linaro: https://launchpad.net/linaro-toolchain-binaries
- Bootloader
  - Das U-Boot – the Universal Boot Loader http://www.denx.de/wiki/U-Boot
  - Source – http://git.denx.de/?p=u-boot.git;a=summary
- Linux Kernel
  - Linus's Mainline tree: http://git.kernel.org/?p=linux/kernel/git/torvalds/linux.git;a=summary
  - Freescale Linux source tree: git://git.freescale.com/imx/linux-imx.git
  - Freescale BSP meta layer: git://git.freescale.com/imx/meta-fsl-bsp-release
  - OpenEmbedded/Yocto BSP layer for Freescale's ARM platform git://git.yoctoproject.org/meta-fsl-arm
  - Embedian SMARC-iMX8MM kernel source tree for linux smarc-8mm_imx_4.19.35_1.0.0: git@git.embedian.com:developer/smarc-fsl-linux-kernel.git
- ARM based rootfs
  - Debian Squeeze: http://www.debian.org/

ARM Cross Compiler: GCC

To build Embedian's *SMARC-iMX8MM* FreeRTOS, you will need to install the following ARM GNU gcc compiler first:

For **MCUXpresso 2.8.0 FreeRTOS**, you need to use the following ARM GNU GCC coss compilier.

```
$ wget -c https://developer.arm.com/-/media/Files/downloads/gnu-rm/9-2019q4/gcc-arm-none-eabi-9-2019-q4-major-x86_64-linux.tar.bz2

$ sudo tar -C /opt -xvf gcc-arm-none-eabi-9-2019-q4-major-x86_64-linux.tar.bz2

$ export ARMGCC_DIR=/opt/gcc-arm-none-eabi-9-2019-q4-major
```

Test:

If this test fails, verify that you have the 32bit libraries installed on your development system.

```
$ ${ARMGCC_DIR}/bin/arm-none-eabi-gcc --version

arm-none-eabi-gcc (GNU Tools for Arm Embedded Processors 9-2019-q4-major) 9.2.1 20191025 (release) [ARM/arm-9-branch revision 277599]
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

To download the source codes, you need to register at Embedian's git server first and put your ssh public key there.

Generating SSH Keys

We recommend you use SSH keys to establish a secure connection between your computer and Embedian Gitlab server. The steps below will walk you through generating an SSH key and then adding the public key to our Gitlab account.

Step 1. Check for SSH keys

First, we need to check for existing ssh keys on your computer. Open up Git Bash and run:

```
$ cd ~/.ssh
$ ls
# Lists the files in your .ssh directory
```

Check the directory listing to see if you have a file named either `id_rsa.pub` or `id_dsa.pub`. If you don't have either of those files go to **step 2**. Otherwise, you already have an existing keypair, and you can skip to **step 3**.

Step 2. Generate a new SSH key

To generate a new SSH key, enter the code below. We want the default settings so when asked to enter a file in which to save the key, just press enter.

```
$ ssh-keygen -t rsa -C "your_email@example.com"
# Creates a new ssh key, using the provided email as a label
# Generating public/private rsa key pair.
# Enter file in which to save the key (/c/Users/you/.ssh/id_rsa): [Press enter]
$ ssh-add id_rsa
```

Now you need to enter a passphrase.

```
Enter passphrase (empty for no passphrase): [Type a passphrase]
Enter same passphrase again: [Type passphrase again]
```

Which should give you something like this:

```
Your identification has been saved in /c/Users/you/.ssh/id_rsa.
Your public key has been saved in /c/Users/you/.ssh/id_rsa.pub.
The key fingerprint is:
01:0f:f4:3b:ca:85:d6:17:a1:7d:f0:68:9d:f0:a2:db your_email@example.com
```

Step 3. Add your SSH key to Embedian Gitlab Server

Copy the key to your clipboard.

```
$ cat ~/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAABDAQABAAABAQDQUEnh8uGpfxaZVU6+uE4bsDrs/tEE5/BPW7jMAxak
6qgOh6nUrQGBWS+VxMM2un3KzwvLRJSj8G4TnTK2CSmlBvR+X8ZeXNTyAdaDxULs/StVhH+QRtFEGy4o
iMIzvIlTyORY89jzhIsgZzwr01nqoSeWWASd+59JWtFjVy0nwVNVtbek7NfuIGGAPaijO5Wnshr2uChB
Pk8ScGjQ3z4VqNXP6CWhCXTqIk7EQl7yX2GKd6FgEFrzae+5Jf63Xm8g6abbE3ytCrMT/jYy5OOj2XSg
6jlxSFnKcONAcfMTWkTXeG/OgeGeG5kZdtqryRtOlGmOeuQe1dd3I+Zz3JyT your_email@example.c
om
```

Go to Embedian Git Server. At Profile Setting --> SSH Keys --> Add SSH Key

Paste your public key and press "Add Key" and your are done.

Download Source Codes

For downloading the latest source code of FreeRTOS for SMARC-iMX8MM, we have to use MCUExpresso SDK generator online tool. Below is the link:

https://mcuxpresso.nxp.com/en/select

Embedian has customized it for SMARC-iMX8MM.

```
$ git clone git@git.embedian.com:developer/freertos-embedian.git freertos -b mcuxpresso_sdk_2.8.0-8mm
```

The file structures are as below.

```
$ tree freertos -L 1
freertos
 boards
 CMSIS
 components
 COPYING-BSD-3
 devices
 docs
 EVK-MIMX8MM_manifest_v3_6.xml
 middleware
 MIR
 rtos
 SMARC-iMX8MM_manifest_v3_6.xml
 SW-Content-Register.txt
 tools

9 directories, 4 files
$
```

Install Required Packages

Install cmake.

```
$ sudo apt-get install cmake
```

Build ROMSG Sample Application

Change the directory to the application project directory, which has a path similar to the following:

```
$ cd <install_dir>/freertos/boards/smarcimx8mm/multicore_examples/rpmsg_lite_str_echo_rtos/armgcc

$ ./build_all.sh
```

This will compile and create binaries. The binaries will be available as <install_dir>freertos/boards/smarcimx8mm/multicore_examples/rpmsg_lite_str_echo_rtos/armgcc/release/rpmsg_lite_str_echo_rtos_imxcm4.bin

Same steps will be applied for other sample applications.

Test Application at U-Boot

Once you finished compilation of rtos application, you can test it at U-Boot first. Prepare a working SD card.

```
$ export DISK=/dev/sdb

$ cd <install_dir>/freertos/boards/smarcimx8mm/multicore_examples/rpmsg_lite_str_echo_rtos/armgcc/releas
e/

$ sudo mount ${DISK}1 /media/boot/

$ sudo cp -v rpmsg_lite_str_echo_rtos_imxcm4.bin /media/boot

$ sudo umount /media/boot
```

Bootup the device and stop at U-Boot Command Prompt.

```
u-boot$ load mmc 1:1 0x48000000 rpmsg_lite_str_echo_rtos_imxcm4.bin
```

```
16828 bytes read in 16 ms (1 MiB/s)
u-boot$ run cpm4mem
u-boot$ run m4boot
Booting M4 from TCM
## Starting auxiliary core at 0x007E0000 ...
u-boot$
```

You will see messages from Cortex-M4 debug port at SER1.

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

1. If your u-boot version is v2020.04, the log message will be somewhat different.

   u-boot$ load mmc 1:1 0x48000000 rpmsg_lite_str_echo_rtos_imxcm4.bin

   ```
   16828 bytes read in 16 ms (1 MiB/s)
   u-boot$ run cpm4mem
   u-boot$ run m4boot
   Booting M4 from TCM
   ## Starting auxiliary core stack = 0x20020000, pc = 0x1FFE0355...
   u-boot$
   ```

2. If your u-boot version is v2020.04, the imx-atf patch should not be applied. This UART4 should be assigned to M4 domain in RDC.

Setup SD Card

Assuming that you have a working Yocto/Debian/Ubuntu SD card.

Replace kernel device tree file:

The FreeRTOS examples uses SER1, I2C3 (I2C_GP, S48 and S49), SPI0, GPIO0 (P108) and PWM3 (GPIO5). It is therefore, we need to disable those pin for Cortex-A53 and reserve a dedicate memory area for Cortex-M4.
All available DTB files for Cortex-M4 are listed in the table below.

DTB File Name (for Cortex-M4)	Description
fsl-smarcimx8mm-m4.dtb	Device tree blob for SMARC-iMX8MM

Replacing Kernel device tree file to adopt Cortex-M4

$ export DISK=/dev/sdb

```
$ sudo mount ${DISK}1 /media/boot/
$ sudo cp -v arch/arm64/boot/dts/embedian/fsl-smarcimx8mm-m4.dtb /media/boot/dtbs/fsl-smarcimx8mm.dtb
$ sudo umount /media/boot
```

```
The device tree name in your SD card has be to fsl-smarcimx8mm.dtb
```

Pass Kernel Parameters to Kernel (uEnv.txt)

At uEnv.txt file, you need to add the following parameters.

```
$ sudo mount ${DISK}1 /media/boot/
```

```
$ sudo vim /media/boot/uEnv.txt

$ sudo umount /media/boot
```

Add the following parameter below the line "image=Image"

```
m4_bin=m4_binary_name.bin
m4_addr=0x7e0000
m4_addr_tmp=0x48000000
```

Add the following parameters in the "uenvcmd" and before "run mmcboot".

```
uenvcmd=run loadimage; run loadfdt; run loadm4bin; run cpm4mem; run m4boot; run mmcboot
```

Boot up the device and you will see "M4 is started" at dmesg.

Demo Programs

In this section, we will show how to use the demo programs.

RPMsg TTY demo

This demo application demonstrates the RPMsg remote peer stack. It works with Linux RPMsg master peer to transfer string content back and forth. The Linux driver creates a tty node to which you can write to. The MCU displays what is received, and echoes back the same message as an acknowledgement. The tty reader on ARM Cortex-A53 core can get the message, and start another transaction. The demo demonstrates RPMsg's ability to send arbitrary content back and forth.

Copy *rpmsg_lite_str_echo_rtos_imxcm4.bin* to the first partition of your SD card and add *m4_bin=rpmsg_lite_str_echo_rtos_imxcm4.bin* at uEnv.txt file. Boot up the device. You will see SER1 print out the following Cortex-M4 messages.

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

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

At device side,

```
$ modprobe imx_rpmsg_tty

$ echo "this is a test" > /dev/ttyRPMSG30
```

You will see SER1 print out the following messages.

```
Get Message From Master Side : "hello world!" [len : 12]
Get Message From Master Side : "this is a test" [len : 14]
Get New Line From Master Side
```

RPMsg Ping Pong demo

Same as previous demo, this one demonstrates the RPMsg communication. After the communication channels are created, Linux OS transfers the first integer to FreeRTOS OS. The receiving peer adds 1 to the integer and transfers it back, a hundred times and then stops.

Copy *rpmsg_lite_pingpong_rtos_linux_remote.bin* to the first partition of your SD card and add *m4_bin=rpmsg_lite_pingpong_rtos_linux_remote.bin* at uEnv.txt file. Boot up the device. You will see SER1 print out the following Cortex-M4 messages.

```
RPMSG Ping-Pong FreeRTOS RTOS API Demo...
RPMSG Share Base Addr is 0xb8000000
Link is up!
Namevice announce sent.
```

At device side,

```
$ modprobe imx_rpmsg_pingpong
```

You will see SER1 print out the following messages.

```
Waiting for ping...
Sending pong...
Waiting for ping...
Sending pong...
Waiting for ping...
Sending pong...
Waiting for ping...
Sending pong...
Waiting for ping...
Sending pong...
Waiting for ping...
Sending pong...
Waiting for ping...
Sending pong...
Waiting for ping...
```

Hello World demo

The Hello World project is a simple demonstration program that uses the BSP software. It prints the “Hello World” message to the ARM Cortex-M4 debug terminal SER1.

Copy *hello_world.bin* to the first partition of your SD card and add *m4_bin=hello_world.bin* at uEnv.txt file. Boot up the device. You will see SER1 print out the following Cortex-M4 messages.

```
hello world.
```

GPIO demo

The GPIO demo code uses GPIO0 as an example to toggle this pin.

Copy *igpio_led_output.bin* to the first partition of your SD card and add *m4_bin=igpio_led_output.bin* at uEnv.txt file. Boot up the device. You will see SER1 print out the following Cortex-M4 messages.

```
GPIO Driver example

The LED is blinking.
```

Use scope to measure GPIO0, you will see a square wave that means this pin is toggled.

PWM demo

The PWM demo code uses GPIO5 (PWM3) as an example.

Copy *ipwm.bin* to the first partition of your SD card and add *m4_bin=ipwm.bin* at uEnv.txt file. Boot up the device. You will see SER1 print out the following Cortex-M4 messages.

```
PWM driver example.
```

Use scope to measure GPIO5, you will see PWM signals

Setup eMMC

For *SMARC-iMX8MM*, the SD card is always emulated as /dev/mmcblk1 and on-module eMMC is always emulated as /dev/mmcblk0. Setting up eMMC now is nothing but changing the device descriptor.

Follow exactly the same steps as that set up SD card. This time, the first partition of eMMC will be /dev/mmcblk0p1.
