

ESM7000 Cortex-M4 技术开发参考手册

1. 概述

ESM7000 是英创基于 i.MX7D 处理器开发的低功耗高性能工控主板，支持双网口、6 串口、双 CAN 总线接口、PCIe、ISA 总线等丰富的通讯接口，支持 18-bit 并行 RGB 或 LVDS 显示接口。主 CPU i.MX7D 是 NXP 推出的异构多核处理器，配置了主频高达 1GHz 的 ARM Cortex-A7 双核和一颗运行速度 240MHz、带硬件浮点运算的 ARM Cortex-M4 内核。

ESM7000 可预装正版 WEC7 或 Linux 操作系统，但对于一些实时性要求极高的应用，无论是 WEC 还是 Linux 操作系统都无法满足对中断事件的及时响应，而且频繁的中断响应也会大大的降低操作系统性能。对这类应用场合就可充分利用 i.MX7D 的异构多核结构，由高性能的 Cortex-A7 双核完成人机交互、数据处理、通讯管理等复杂运算，而对于实时的数据采集、高速的中断事件响应等实时任务交由 i.MX7D 的 Cortex-M4 完成。

对于 Cortex-M4 的应用程序开发，NXP 为 i.MX7D Cortex-M4 内核提供了 FreeRTOS 软件开发包 i.MX7D Cortex-M4 FreeRTOS BSP。FreeRTOS 是具有“固定优先级抢占式调度”的“硬实时”轻量级操作系统，在 ESM7000 上实际测试 Cortex-M4 的中断响应时间稳定的小于 500ns，就算利用 FreeRTOS 的二值信号量同步技术，将中断处理从中断服务程序(ISR)中剥离，利用信号量同步将中断事件“推迟”处理，中断响应的时间也可稳定在 2us 左右。

为了方便用户开发 Cortex-M4 程序，我们在 NXP 提供的 FreeRTOS BSP 基础上，针对 ESM7000 的硬件资源，对各个物理外设的驱动程序进行了重新封装，以期最大程度的减少用户对 i.MX7D 芯片的学习时间，以将主要精力集中在实现自己的应用业务逻辑上。

本文将详细介绍 FreeRTOS BSP 的使用方法，ESM7000 Cortex-M4 开发环境的搭建以及如何编译、启动运行 Cortex-M4 应用程序。

2. i.MX7D Cortex-M4 FreeRTOS BSP

i.MX7D Cortex-M4 FreeRTOS BSP 是一个综合的软件开发包，其中包含了 i.MX7D 处理器外设的驱动程序，已经移植好的 FreeRTOS 操作系统和多核通讯协议栈 RPMsg。下图是 FreeRTOS BSP 的基本结构。

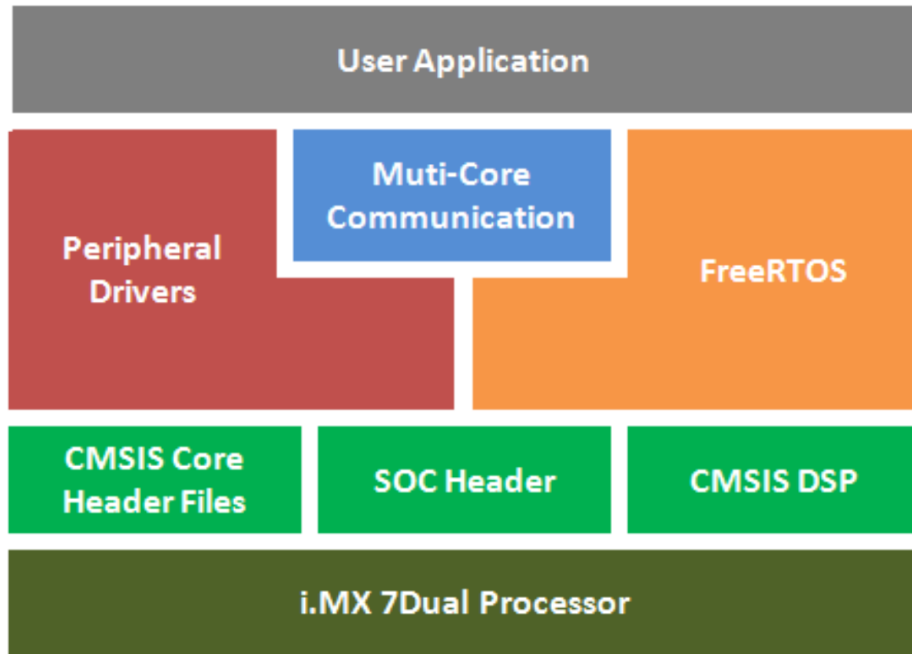


图 1: FreeRTOS BSP for i.MX 7Dual architecture

开发包中的示例程序演示了 FreeRTOS 内核、PRMsg 以及 i.MX7D/ESM7000 外设的使用方法。

NXP 提供了两个 FreeRTOS BSP 安装包：一个是在 Windows 操作系统上的自解压.exe 安装包，另一个是在 Linux 系统上安装的 tarball。两个安装包解压后具有完全相同的目录结构。英创在 NXP 提供的 FreeRTOS BSP 基本上增加了 ESM7000 的相关例程，同时对 ESM7000 所使用的 i.MX7DL 外设硬件资源的驱动进行了重新封装。与 NXP 提供的驱动程序相比，NXP 更关注操作的灵活性，提供寄存器一级的封装，英创提供的外设驱动程序与 ESM7000 的硬件资源紧密结合、从外设功能的角度出发，提供了更加简洁易用的驱动 API 函数。

英创只提供在 Linux 系统上安装的 tarball 包，所包含的文件目录大致如下：

描述	位置
例程	<install_dir>/examples/...
例子程序	<install_dir>/examples/<board_name>/demo_apps/...
驱动使用方法例程	<install_dir>/examples/<board_name>/driver_examples/...
文档	<install_dir>/doc/...

中间层	<install_dir>/middleware/...
外设驱动、启动代码等	<install_dir>/platform/...
CMSIS ARM Cortex-M 头文件, DSP 库文件	<install_dir>/platform/CMSIS/...
处理器头文件	<install_dir>/platform/devices/<devices_name>/include/...
所支持工具链的链接脚本	<install_dir>/platform/devices/<devices_name>/linker/...
CMSIS 兼容的启动代码	<install_dir>/platform/devices/<devices_name>/startup/...
外设驱动	<install_dir>/platform/drivers/...
工具程序, 比如调试终端	<install_dir>/platform/utilities/...
FreeRTOS 内核代码	<install_dir>/rots/FreeRTOS/...
工具	<install_dir>/tools/...

2.1 FreeRTOS BSP 目录结构

FreeRTOS BSP 提供了两种类型的例子程序:

- **Demos:** 基于 FreeRTOS 操作系统的实际应用的例子, 通常会同时使用 ESM7000 的多个外设。
- **Examples:** 简单的“裸机”例子程序, 用于演示 ESM7000 外设驱动的用法。

打开任何一个 Demo/example 程序, 它们都引用了很多其它文件, 而且这些引用的文件大多都是共享文件, 一般不建议用户修改。在开发 Cortex-M4 程序之前, 需要对 FreeRTOS BSP 的目录结构有一个全局的认识, 并理解各源文件的存放位置。我们建议用户拷贝一个我们提供的 Demo 例程, 这此基础上修改开发自己的专用应用程序。

FreeRTOS BSP 安装根目录下的 4 个主要文件夹包含了每个 demo 程序所使用的所有源代码文件。



图 2: FreeRTOS BSP 根目录

2.2 FreeRTOS BSP examples 文件夹

所有 demos/examples 源代码都包含在 examples 目录下、以主板名称命名的文件夹中，examples 的目录结构如下图所示：

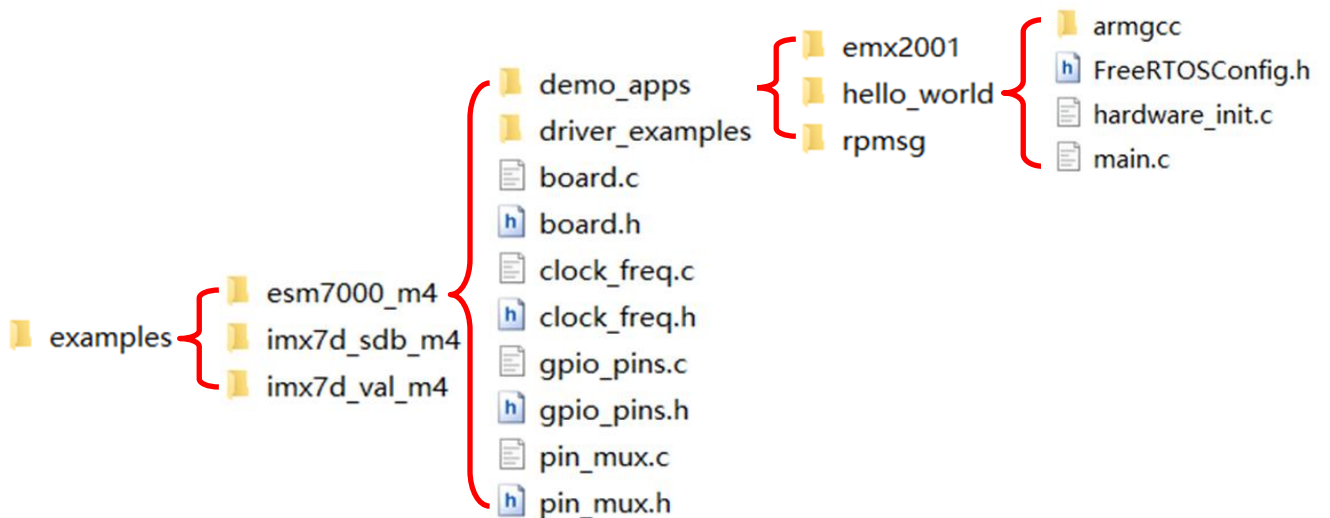


图 3: examples 目录结构

imx7d_sdb_m4 和 imx7d_val_m4 是 NXP 官方评估板的 Coretex-M4 例子程序，esm7000_m4 是 ESM7000 工控主板专用的例子程序，在 esm7000_m4 文件夹的根目录下包含了：

- **board.c/h:** board.h 包含了 ESM7000 在 Coretex-M4 侧所有可用的硬件资源宏定义，包括调试串口、SPI、PWM、可用的 GPIO 资源等。board.c 中包含了各硬件外设的时钟及 RDC 初始化代码。
- **clock_freq.c/h:** 包含了用于获取外设当前时钟频率的功能函数。

- **gpio_pins.c/h:** 包含了操作 ESM7000 GPIO 的驱动函数。虽然 ESM7000 支持 32 位 GPIO，但只有在 board.h 中同时定义了 GPIO 才可以可以在 Cortex-M4 中使用。
- **pin_mux.c/h:** CPU 管脚的复用设置。

上述文件由英创根据 ESM7000 硬件资源而提供，不建议用户修改。

2.3 FreeRTOS BSP middleware 文件夹

middleware 文件夹包含了 demos/examples 所使用的所有中间层源代码。开源的多核通讯协议栈 -RPMsg 就包含在 middleware 文件夹。



图 4: Middleware 目录结构

2.4 FreeRTOS BSP platform 文件夹

Platform 文件夹包含了 CMSIS 头文件、外设驱动程序、startup、utilities 和 linker 文件。

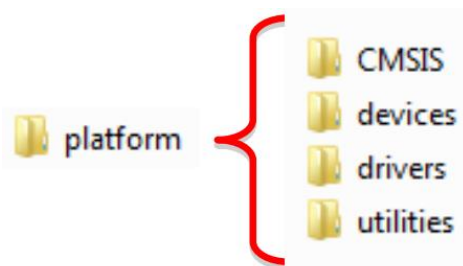


图 5: Platform 目录结构

2.5 FreeRTOS BSP rtos 文件夹

Rots 目录是 FreeROTS 操作系统相关的源文件目录。

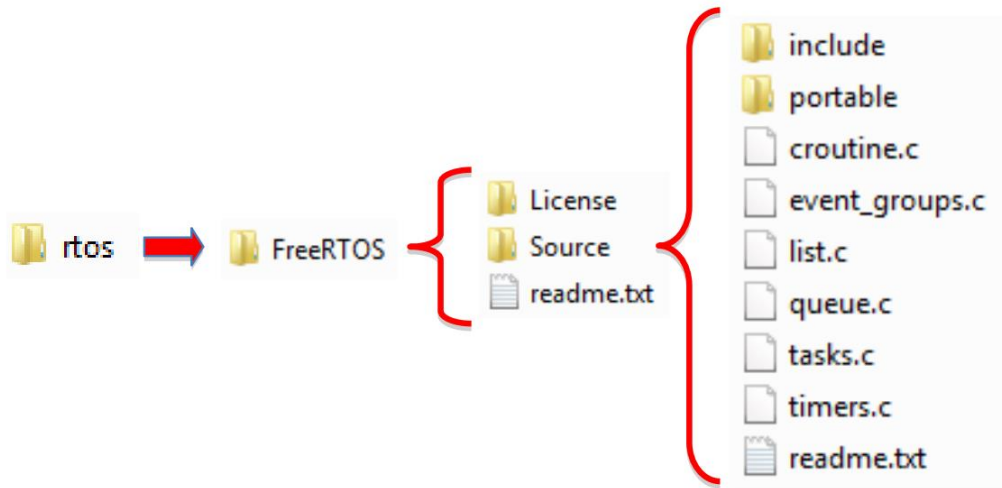


图 6: rtos 目录结构

3. i.MX7D Cortex-M4 软件开发环境搭建

i.MX7D Cortex-M4 应用程序开发与普通的 Cortex-M3/M4 单片机程序的开发完全一样，NXP 提供了 IAR Embedded Workbench IDE，DS-5 IDE 和 ARM GCC 三种工具编译程序的方法，ESM7000 只提供了使用 ARM GCC 工具链的工程文件，如果要使用 IAR Embedded Workbench IDE 或 DS-5 IDE 请参考 FreeRTOS BSP 安装目录下的<install_dir>/doc/Getting_Started_with_FreeRTOS_BSP_for_i.MX_7Dual.pdf。

3.1 Win10 安装 Linux 子系统

<install_dir>/examples/esm7000_m4 下的例子程序是在 Win10 的 Linux 子系统(WSL)环境下编译测试的，使用的 Linux 版本为 Ubuntu 18.04，如果用户已经有了 Linux 开发主机，可跳过此小节。

Win10 安装 Linux 子系统请参考：<https://docs.microsoft.com/zh-cn/windows/wsl/>。

3.2 安装 ARM GCC 及 CMake

- 根据所用开发主机的操作系统，下载对应的 GUN Arm Embedded 工具链，下载地址：<https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads>。这里以 64 位 Win10 操作系统下的 Ubuntu 18.04 Linux 子系统举例，下载对应的工具包 gcc-arm-none-eabi-10-2020-q4-major-x86_64-Linux.tar.bz2 到 D 盘根目录。

3 [gcc-arm-none-eabi-10-2020-q4-major-x86_64-linux.tar.bz2](#)

Linux x86_64 Tarball

MD5: 8312c4c91799885f222f663fc81f9a31

- 解压 tarball，执行：

```
sudo tar xjf /mnt/d/gcc-arm-none-eabi-10-2020-q4-major-x86_64-Linux.tar.bz2 -C /usr/share
```

- 建软链接，执行：

```
sudo ln -s /usr/share/gcc-arm-none-eabi-10-2020-q4-major/bin/arm-none-eabi-gcc /usr/bin/arm-none-edbi-gcc
```

```
sudo ln -s /usr/share/gcc-arm-none-eabi-10-2020-q4-major/bin/arm-none-eabi-g++ /usr/bin/arm-none-edbi-g++
```

```
sudo ln -s /usr/share/gcc-arm-none-eabi-10-2020-q4-major/bin/arm-none-eabi-gdb /usr/bin/arm-none-edbi-gdb
```



```
sudo ln -s /usr/share/gcc-arm-none-eabi-10-2020-q4-major/bin/arm-none-eabi-size  
/usr/bin/arm-none-edbi-size
```

- 可通过--version 命令查看 gcc 版本: arm-none-edbi-gcc --version
- 安装 CMake, 执行: sudo apt-get install cmake

3.3 使用 ARM GCC 编译 Hello_world

解压英创提供的 ESM_FreeRTOS_BSP_1.0.1_iMX7D_Linux.tar.gz 软件包, 在编译任何工程之前, 需要先设置一次编译工具链的环境变量, 执行:

```
export ARMGCC_DIR=/usr/share/gcc-arm-none-eabi-10-2020-q4-major
```

接下来就可编译 FreeRTOS BSP 里的例程了, 进入 <install_dir>/examples/esm7000_m4/demo_apps/hello_world/armgcc 目录, 执行 ./build_realse.sh 或 ./build_debug.sh, 编译工具将在 armgcc 目录下自动创建 realse/debug 目录, 并在其下生成 hello_world 的 .bin、.elf、.hex 和 .map 文件。

3.4 在 WSL2 中编译 vs 在 Windows 文件系统中编译

WSL2 使用了最新的虚拟技术在量化的虚拟机中运行 Linux 内核, WSL2 在多个方面与微软最初推出的 WSL1 相比都更具优势, 但除了跨操作系统文件系统性能。详细说明请参考: <https://docs.microsoft.com/zh-cn/windows/wsl/compare-versions>

我们将 FreeROTS BSP 分别解压到 WSL2 Linux 文件系统中 和 Windows 文件系统中进行测试, 与将文件存放在 Windows 中跨文件系统编译相比, 把文件存放在 WSL 中直接编译速度要快 5 倍以上。

4. 运行 i.MX7D Cortex-M4 应用程序

在 Cortex-M4 程序开发过程中,可利用 U-Boot 直接加载并运行编译好的 Cortex-M4 应用程序,通过 PRINTF 输出打印信息来调试应用程序。程序发布时可利用 Linux 的 flash_opt 工具,将要发布的 Cortex-M4 应用程序 bin 文件固到系统 eMMC 中, U-Boot 在启动时会先自动启动 Cortex-M4 程序,再启动 Linux 系统。

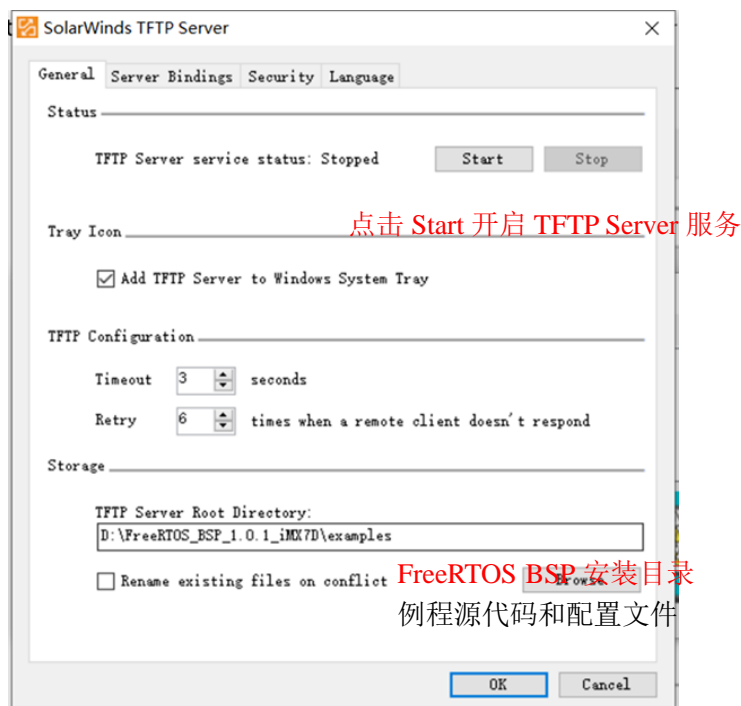
4.1 硬件连接

ESM7000 开发环境的硬件连接请参考《ESM7000 工控主板使用必读》,最基本的需要连接并配置好网络、连接 ESM7000 控制台串口(调试串口)和给系统提供 5V 电源。ESM7000 的 ttys2(COM3)为 Cortex-M4 程序的调试串口,在程序中调用 PRINTF 将从 ESM7000 的 ttys2(COM3)输出信息, Cortex-M4 调试串口(ESM7000-ttys2)缺省为 RS232 电平,通讯格式与 ESM7000 的控制台串口完全一样,默认为 115200-8-N-1。为了查看 Cortex-M4 输出的打印信息,需要再用一条串口线将 ESM7000 开发评估底板上的 ttys2(COM3)与开发主机的串口相连。

4.2 在 U-boot 中通过 TFTP 下载并运行 Cortex-M4 应用程序

安装 TFTP Server

用户可以在开发主板上选择自己熟悉的 TFTP Server 工具,这里以在 Win10 下使用 SolarWinds TFTP Server 配置为例,只需要设置好目录,点击 start 启动 TFTP 服务就可以了。(可能需要合理配置开发主机的防火墙)



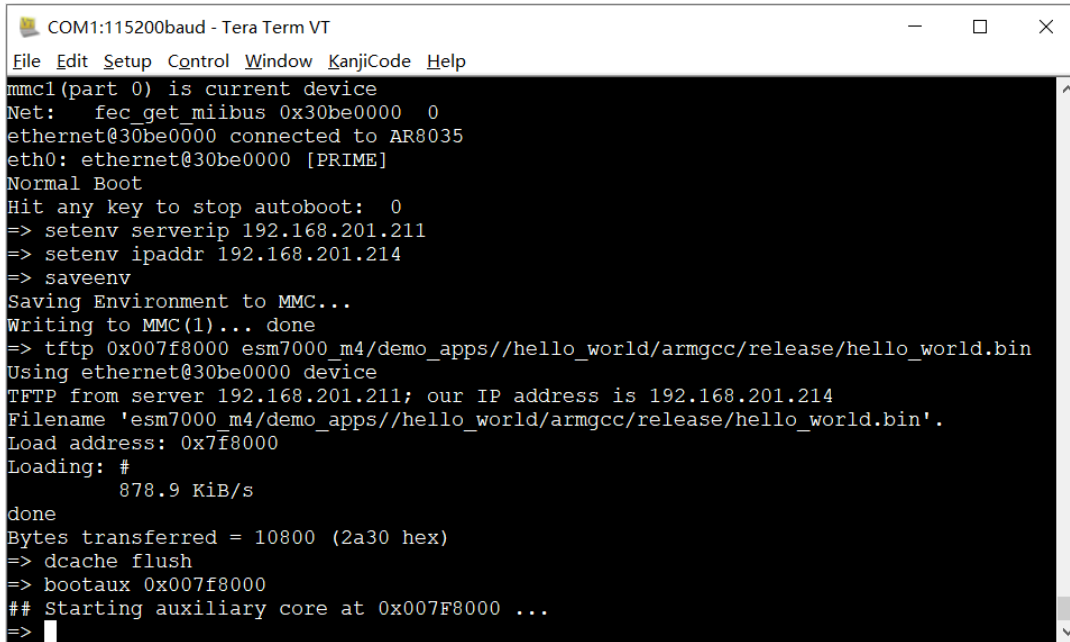
如果用户是将 FreeRTOS BSP 安装在 WSL Linux 系统中, SolarWinds TFTP Server 工具不能将 WSL 中的目录作为 TFTP Server 的 Root Directory, 简单的处理方法是将编译生成的 bin 文件拷贝到 Windows 的文件系统目录中, 拷贝动作可以通过修改相应的 build_xxxx.sh 脚本自动完成, 比如在编译完成后自动将 hello_world.bin 文件拷贝到 Windows D 盘根目录, 修改 build_release.sh 文件如下:

```
#!/bin/sh
cmake -DCMAKE_TOOLCHAIN_FILE="../../../../tools/cmake_toolchain_files/armgcc.cmake" -G "Unix Makefiles"
-DCMAKE_BUILD_TYPE=Release .
make -j4
cp release/hello_world.bin /mnt/d #增加自动拷贝动作
```

通过 U-boot 在 TCM 上运行 Cortex-M4 应用程序

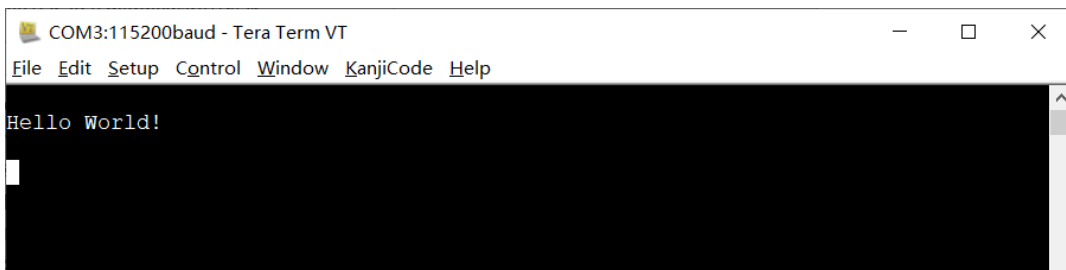
TCM(Cortex-M4 Core's Tightly Coupled Memory)是 i.MX7D CPU 片内为 Cortex-M4 内核使用的一块专用内存, 包含 32KBytes 的程序存储空间和 32KBytes 运行内存空间。连接好网线、ESM7000 控制终端串口(这里连接到开发主机的 COM1)、Cortex-M4 调试串口(这里连接到开发主机的 COM3), 短接开发评估底板上的 JP1, 以便让 ESM7000 运行在调试模式, 然后给 ESM7000 上电, 在开发主板 COM1 串口工具上按空格键进入 ESM7000 U-boot, 依次执行如下命令:

- a. setenv serverip 192.168.201.211 // 设置 TFTP Server IP 地址(开发主板 IP 地址)
- b. setenv ipaddr 192.168.201.213 // 设置 ESM7000 网口 IP 地址(与开发主机同一网段)
- c. saveenv // 可以使用 saveenv 命令保存环境变量, 系统下次启动时就不需要重复设置 IP 地址了。
- d. tftp 0x007f8000 esm7000_m4/demo_apps/hello_world/armgcc/release/hello_world.bin // 从开发主机上下载 hello_world.bin 到 i.MX7D TCM 中
- e. dcache flush // 清除 dcache
- f. bootaux 0x007f8000 //从 TCM 启动 Cortex-M4 内核



```
COM1:115200baud - Tera Term VT
File Edit Setup Control Window KanjiCode Help
mmci(part 0) is current device
Net: fec_get_miibus 0x30be0000 0
ethernet@30be0000 connected to AR8035
eth0: ethernet@30be0000 [PRIME]
Normal Boot
Hit any key to stop autoboot: 0
=> setenv serverip 192.168.201.211
=> setenv ipaddr 192.168.201.214
=> saveenv
Saving Environment to MMC...
Writing to MMC(1)... done
=> tftp 0x007f8000 esm7000_m4/demo_apps//hello_world/armgcc/release/hello_world.bin
Using ethernet@30be0000 device
TFTP from server 192.168.201.211; our IP address is 192.168.201.214
Filename 'esm7000_m4/demo_apps//hello_world/armgcc/release/hello_world.bin'.
Load address: 0x7f8000
Loading: #
      878.9 KiB/s
done
Bytes transferred = 10800 (2a30 hex)
=> dcache flush
=> bootaux 0x007f8000
## Starting auxiliary core at 0x007F8000 ...
=>
```

如果程序正常启动运行，在开发主机的 COM3 口上(连接到了 Cortex-M4 的调试串口 ESM7000-ttys2) 可以看到程序运行时输出的 Hello World! 信息。



```
COM3:115200baud - Tera Term VT
File Edit Setup Control Window KanjiCode Help
Hello World!
```

通过 U-boot 在 OCRAM/DDR 上运行 Cortex-M4 应用程序

在 FreeRTOS BSP 中一些后缀名为 “_ddr”或“_ocram” 的例子程序应该加载到 OCRAM 或 DDR 中运行。连接好网线、ESM7000 控制终端串口(这里连接到开发主机的 COM1)、Cortex-M4 调试串口(这里连接到开发主机的 COM3)，短接开发评估底板上的 JP1，以便让 ESM7000 运行在调试模式，然后给 ESM7000 上电，在开发主板 COM1 串口工具上按空格键进入 ESM7000 U-boot，依次执行如下命令：

- **OCRAM**

- setenv serverip 192.168.201.211 // 设置 TFTP Server IP 地址(开发主板 IP 地址)
- setenv ipaddr 192.168.201.213 // 设置 ESM7000 网口 IP 地址(与开发主机同一网段)
- saveenv // 可以使用 saveenv 命令保存环境变量，系统下次启动时就不需要重复设置 IP 地址了。
- tftp 0x00920000 esm7000_m4/demo_apps/hello_world_ddr/armgcc/release/hello_world_ocram.bin // 从开发主板上下载 hello_worldddr_ocram.bin 到 i.MX7D OCRAM 中

- e. `dcache flush` // 清除 dcache
- f. `bootaux 0x00920000` //从 DDR 0x00920000 地址启动 Cortex-M4 内核
- **DDR**
- a. `setenv serverip 192.168.201.211` // 设置 TFTP Server IP 地址(开发主板 IP 地址)
- b. `setenv ipaddr 192.168.201.213` // 设置 ESM7000 网口 IP 地址(与开发主机同一网段)
- c. `saveenv` // 可以使用 `saveenv` 命令保存环境变量，系统下次启动时就不需要重复设置 IP 地址了。
- d. `tftp 0x80100000 esm7000_m4/demo_apps/hello_world_ddr/armgcc/release/hello_world_ddr.bin` // 从开发主板上下载 `hello_worldld_ddr.bin` 到 ESM7000 DDR 中
- e. `dcache flush` // 清除 dcache
- f. `bootaux 0x80100000` //从 DDR 0x80100000 地址启动 Cortex-M4 内核

4.3 Cortex-M4 程序固化及开机自动运行

如前所述，Cortex-M4 程序可以加载到 i.MX7D 的 TCM、OCRAM 或 DDR 中运行，各个 memory 区域提供的程序 CODE 和 DATA 空间如下表所示：

配置	CODE	DATA	A7 & M4 Sheard Memory	CoreMark
TCM	32K (007F8000-007FFFFF)	32K (00800000-00807FFF)	64K (00910000-0091FFFF)	626.213
OCARM	128K (00920000-0093FFFF)	64K (007F8000-00807FFF)		450.288
DDR	1MB (80100000-801FFFFF)	128K (00920000-0093FFFF)		439.811

上表中的 CoreMark 是不同配置下的性能测试，可以看到程序在 TCM 中运行时性能最佳，OCRAM 次之，但提供了更大的程序运行空间。Cortex-M4 程序运行位置由 `platform/devices/MCIMX7D/linker/gcc/ESM7000_M4_*.ld` 链接文件决定，在 project 各自的 CMakeLists.txt 中指定。

对于 M4 程序的固化，在进入 Linux 系统后，可以调用 `flash_opt` 命令进行设置。`flash_opt` 在将 Coretex-M4 程序固化到系统 eMMC 的同时，会根据烧写文件的尾缀进行相应设置，以便在系统启动时能根据设置自动将 Coretex-M4 程序加载到 TCM、OCRAM 或 DDR 中运行。具体步骤为：将需要烧写的程序（以 `hello_world.bin` 为例），拷贝到主板可以访问的目录中（比如 `/mnt/mmc`），并根据程序实际运行的位置对程序重命名，如果程序是在 TCM 中运行，则需要命名为 `hello_world_tcm.bin`，如果程序在 OCRAM 中运行，则命名为

hello_world_ocram.bin, 如果是在 DDR 在运行, 则命名为 hello_world_ddr.bin。重命名只是为了满足 flash_opt 对命令格式的要求, 程序在编译时必须根据程序加载的位置选择对应的 ESM7000_M4_*.ld 文件。

执行烧写命令 (以烧写到 TCM 为例):

```
#>flash_opt m4 hello_world_tcm.bin
```

操作完成后, 就可以重启主板, 此后系统在启动时就会自动运行 M4 程序 hello_world.bin 了, 如果要更新程序, 重复上述操作即可。

M4 程序一旦通过 flash_opt 固化到系统中, 在系统每次启动时 M4 程序就会自动加载运行, 此时就不能再在 U-boot 中手动下载、启动 M4 程序进行调试。如果要重新在 U-boot 中调试 M4 程序, 需要在 Linux 中执行 #>flash_opt m4 dump.bin 来擦除先前写入的应用程序, dump.bin 实际上是一个内容全为 0xFF 的二进制文件。

5. 技术支持

成都英创信息技术有限公司是一家从事嵌入式工控主板产品研发、市场应用的专业公司。用户可通过公司网站、技术论坛、电话、邮件等方式来获得有关产品的技术支持。公司联系方式如下：

地址：成都市高新区高朋大道 5 号博士创业园 B 座 407# 邮编：610041

联系电话：028-86180660

传真：028-85141028

网址：<http://www.emtronix.com>

电子邮件：support@emtronix.com

6. 版本历史

版本	简要描述	日期
V1.0	创建 ESM7000 Cortex-M4 技术开发参考手册	2021-4

注意：本手册的相关技术内容将会不断的完善，请客户适时从公司网站下载最新版本的数据手册，恕不另行通知。