

## ESM8000 MCU 技术开发参考手册

## 1. 概述

ESM8000 是英创基于 i.MX8MM 处理器开发的 64 位高性能工控主板，支持双网口、11 串口、双 CAN FD 总线、PCIe、等丰富的通讯接口，支持双 LVDS 全高清显示接口。主 CPU i.MX8MM 是 NXP 推出的异构多核处理器，配置了主频高达 1.6GHz 的四核 ARM Cortex-A53 和一颗运行速度 400MHz、带硬件浮点运算的 MCU——ARM Cotex-M4 内核。

ESM8000 预装 Linux 操作系统，但对于一些实时性要求极高的应用，无论是 WCE 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.MX8MM 的异构多核结构，由高性能的 Cortex-A53 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.MX8MM 的 Cotex-M4 完成。

我们提供了 ESM8000 MCU SDK 帮助用户开发 i.MX8M MCU 应用程序，ESM8000 MCU SDK 提供了 ESM8000 的外设驱动程序以及驱动使用例程，SDK 还包含了 FreeRTOS 内核和 RPMsg 多核通讯协议栈。

本文将详细介绍 ESM8000 MCU 程序的开发方法，包括开发环境的搭建，如何编译、启动运行 MCU 应用程序等内容。

## 2. ESM8000 MCU SDK

ESM8000 MCU SDK 是一个综合的软件开发包，其中包含了 i.MX8M 系列处理器 MCU 初始化代码，外设驱动程序，已经移植好的 FreeRTOS 操作系统和多核通讯协议栈 RPMsg 等。

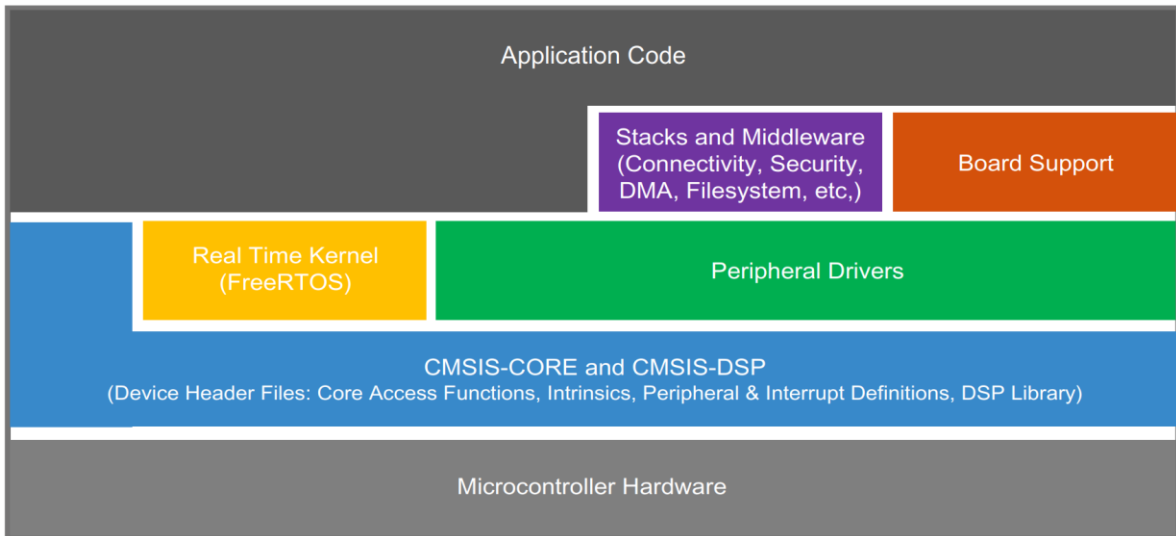


图 1: ESM8000 MCU SDK Layers

ESM8000 MCU SDK 包含了对基于 i.MX8M 系列处理器的多个主板的支持，各主板对应的硬件定义和例程位于 <install\_dir>/boards/<board\_name> 目录中。目前 SDK 中包含了对下面 4 个主板的支持：

- esm8000\_m4 → 英创 ESM8000 工控主板，基于 i.MX8MM(Cortex A53 + Cortex M4)
- esm8200\_m7 → 英创 ESM8200 工控主板，基于 i.MX8MN(Cortex A53 + Cortex M7)
- evkmimx8mm → NXP i.MX8MM 官方评估板
- evkmimx8mn → NXP i.MX8MN 官方评估板

图 2: boards 目录

### 2.1 esm8000\_m4 文件夹

esm8000\_m4 和 esm8200\_m7 具有完全相同的目录结构，这里以 esm8000\_m4 为例进行介绍。

<install\_dir>/boards/esm8000\_m4 目录内容及简要说明如下：



图 3: esm8000\_m4 目录

esm8000\_m4 目录中除包含 demo\_apps、driver\_examples、drivers、multicore\_examples 4 个子目录外，还包含以下公共文件：

- **board.c/h:** board.h 包含了 ESM8000 在 MCU 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:** 包含了用于设置和获取外设当前时钟频率的功能函数。
- **ESM8000\_cm4\_\*.ld:** ARM GCC 链接文件，配置 MCU 程序的 CODE 及 DATA 段。
- **pin\_mux.c/h:** CPU 管脚的复用设置。
- **utilities.c/h:** 一些有用的实例程序，比如 CPU 使用率统计等。

esm8000\_m4 下的所有 examples 和 demo 都将引用上述公共文件，所以一般不建议修改。

## 3. ESM8000 MCU 软件开发环境搭建

ESM8000 MCU 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，本文仅以使用 ARM GCC 工具链为例，对开发环境的搭建进行说明。

### 3.1 Win10 安装 Linux 子系统

<install\_dir>/boards/esm8000\_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，请跳过此小节。

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 SDK\_1.0.0\_ESM8000-MCU.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 SDK 中的例程了, 进入<install\_dir>/boards/esm8000\_m4/demo\_apps/ hello\_world/armgcc 目录:

执行./build\_realse.sh, 编译工具将基于 esm8000\_cm4\_tcm.ld 文件在 armgcc 目录下自动创建 realse 子目录, 并在其下生成在 ESM8000 TCM 上运行的 hello\_world.bin 文件。

执行./build\_dds\_realse.sh, 编译工具将基于 esm8000\_cm4\_dds\_ram.ld 在 armgcc 目录下自动创建 dds\_realse 子目录, 并在其下生成在 ESM8000 DDR 上运行的 hello\_world.bin 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经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 ESM8000 MCU SDK 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 ESM8000 MCU 应用程序

在 MCU 程序开发过程中，可利用 U-Boot 直接加载并运行编译好的 MCU 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 MCU 应用程序 bin 文件固化到系统 eMMC 中，U-Boot 在启动时会先自动启动 MCU 程序，再启动 Linux 系统。

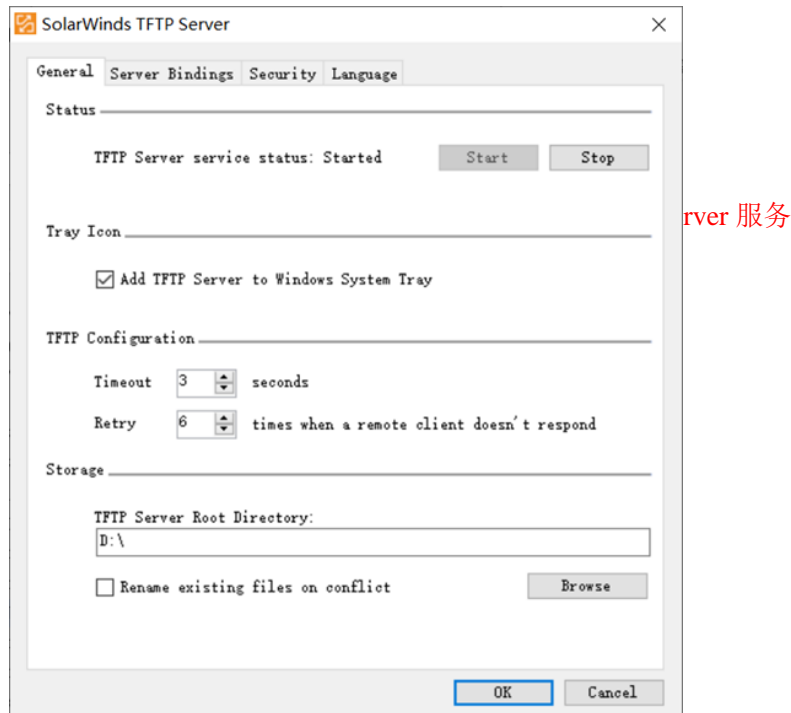
### 4.1 硬件连接

ESM8000 开发环境的硬件连接请参考[《ESM8000 工控主板使用必读》](#)，最基本的需要连接并配置好网络、连接 ESM8000 控制台串口（调试串口）和给系统提供 5V 电源。ESM8000 的 ttys2 为 MCU 程序的调试串口，在 MCU 程序中调用 PRINTF 将从此串口输出信息。ESM8000 主板 ttys2 缺省配置为 TTL 电平，在 ESMARC 评估底板上可通过跳线将 ttys2 转换为 RS232 电平。ttys2 作为 MCU 调试串口时，通讯格式与 ESM8000 的控制台串口完全一样，默认为 115200-8-N-1。为了查看 MCU 输出的打印信息，需要再用一条串口线将 ESM8000 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 MCU 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 ESM8000 MCU SDK 解压在 WSL Linux 文件系统中，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
if [ -d "CMakeFiles" ];then rm -rf CMakeFiles; fi
if [ -f "Makefile" ];then rm -f Makefile; fi
if [ -f "cmake_install.cmake" ];then rm -f cmake_install.cmake; fi
if [ -f "CMakeCache.txt" ];then rm -f CMakeCache.txt; fi
cmake -DCMAKE_TOOLCHAIN_FILE="../../../../tools/cmake_toolchain_files/armgcc.cmake" -G "Unix Makefiles" -
DCMAKE_BUILD_TYPE=release .
make -j 2>&1 | tee build_log.txt
cp release/ hello_world.bin /mnt/d #增加自动拷贝动作
```

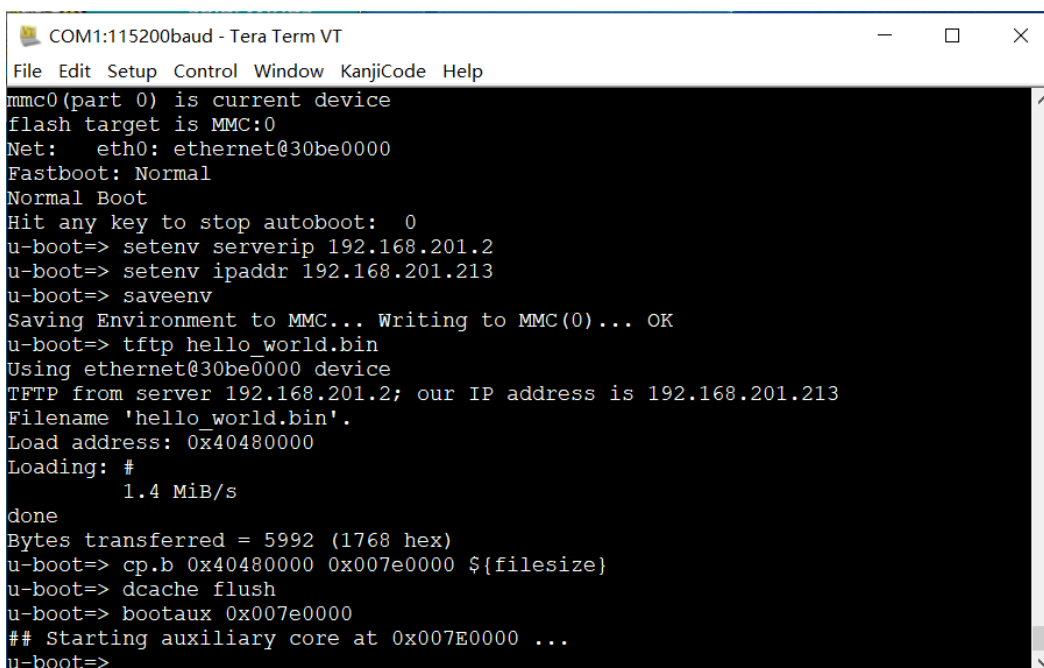
## 通过 U-boot 在 TCM 上运行 MCU 应用程序

TCM(Cortex-M Core's Tightly Coupled Memory)是 i.MX8M CPU 片内为 Cortex-M 内核使用的一块专用内存，包含 128KBytes 的代码存储空间(CODE)和 128KBytes 运行内存空间(DATA)。连接好网线、ESM8000 控制终端串口(这里连接到开发主机的 COM1)、MCU 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 ESM8000 运行在调试模式，然后给 ESM8000 上电，在开发主板 COM1 串口工具上按空格键



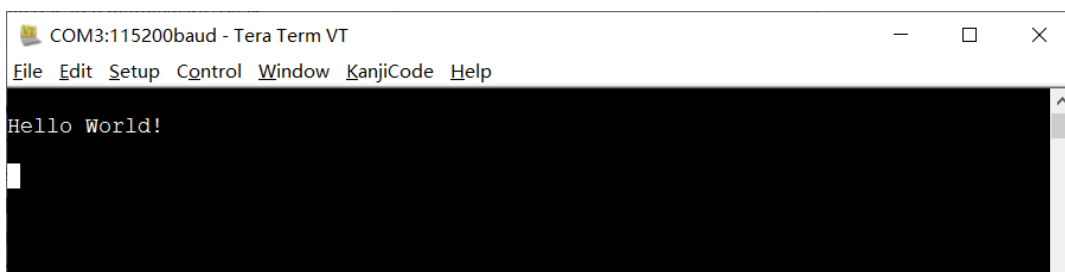
进入 ESM8000 U-boot，依次执行如下命令：

- 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 ESM8000 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 hello_world.bin` // 从开发主机上下载 `hello_world.bin` 到 ESM8000 内存中
- e. `cp.b 0x40480000 0x007e0000 ${filesize}` // 将下载的内容 copy 到 i.MX8M 的 TCM
- f. `dcache flush` // 清除 dcache
- g. `bootaux 0x007e0000` // 从 TCM 启动 MCU 程序



```
COM1:115200baud - Tera Term VT
File Edit Setup Control Window KanjiCode Help
mmc0(part 0) is current device
flash target is MMC:0
Net: eth0: ethernet@30be0000
Fastboot: Normal
Normal Boot
Hit any key to stop autoboot: 0
u-boot=> setenv serverip 192.168.201.2
u-boot=> setenv ipaddr 192.168.201.213
u-boot=> saveenv
Saving Environment to MMC... Writing to MMC(0)... OK
u-boot=> tftp hello_world.bin
Using ethernet@30be0000 device
TFTP from server 192.168.201.2; our IP address is 192.168.201.213
Filename 'hello_world.bin'.
Load address: 0x40480000
Loading: #
      1.4 MiB/s
done
Bytes transferred = 5992 (1768 hex)
u-boot=> cp.b 0x40480000 0x007e0000 ${filesize}
u-boot=> dcache flush
u-boot=> bootaux 0x007e0000
## Starting auxiliary core at 0x007E0000 ...
u-boot=>
```

如果程序正常启动运行，在开发主机的 COM3 口上(连接到了 MCU 的调试串口 ESM8000-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 DDR 上运行 MCU 应用程序

连接好网线、ESM8000 控制终端串口(这里连接到开发主机的 COM1)、MCU 调试串口(这里连接到开发

主机( COM3), 短接开发评估底板上的 JP1, 以便让 ESM8000 运行在调试模式, 然后给 ESM8000 上电, 在开发主板 COM1 串口工具上按空格键进入 ESM8000 U-boot, 依次执行如下命令:

- **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 ESM8000 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 hello_world.bin // 从开发主机上下载 hello_world.bin 到 ESM8000 内存中`
- e. `cp.b 0x40480000 0x80000000 ${filesize} // 将下载的内容 copy 到 i.MX8M 的系统 DDR 中`
- f. `dcache flush // 清除 dcache`
- g. `bootaux 0x80000000 //从系统 DDR 0x80000000 地址启动 MCU 程序`

### 4.3 MCU 程序固化及开机自动运行

如前所述, ESM8000 MCU 程序可以加载到 i.MX8M 的 TCM 或系统 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	Non-cache	CoreMark
TCM	128K (007E0000-007FFFFFFF)	128K (00800000-0081FFFF)	16M (80000000-0x80FFFFFFF)	1043.732
DDR	2MB (80000000-801FFFFFFF)	2MB (80200000-803FFFFFFF)	12M (80400000-0x80FFFFFFF)	733.66

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, 在 DDR 中运行损失了一些的性能, 但提供了更大的程序运行空间。MCU 程序运行位置由 `<install_dir>/boards/<board_name>/*.ld` 链接文件决定。

对于 MCU 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 MCU 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 MCU 程序加载到 TCM 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 `*.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 MCU 程序 `hello_world.bin` 了，如果要更新程序，重复上述操作即可。

MCU 程序一旦通过 `flash_opt` 固化到系统中，在系统每次启动时 MCU 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 MCU 程序进行调试。如果要重新在 U-boot 中调试 MCU 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](mailto:support@emtronix.com)

## 6. 版本历史

版本	简要描述	日期
V1.0	创建 ESM8000 MCU 技术开发参考手册	2021-7

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。