I.MX RT eMMC Boot and R/W Operation

Ji Cheng/Calvin Ji
North China CAS Team

March 2020

SECURE CONNECTIONS FOR A SMARTER WORLD

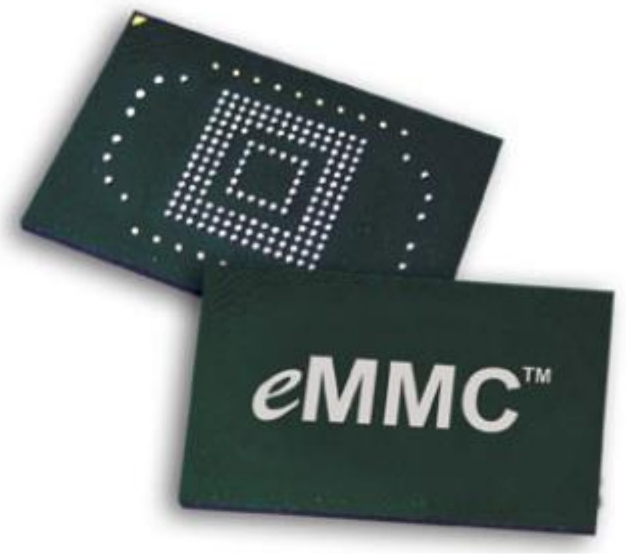
Agenda

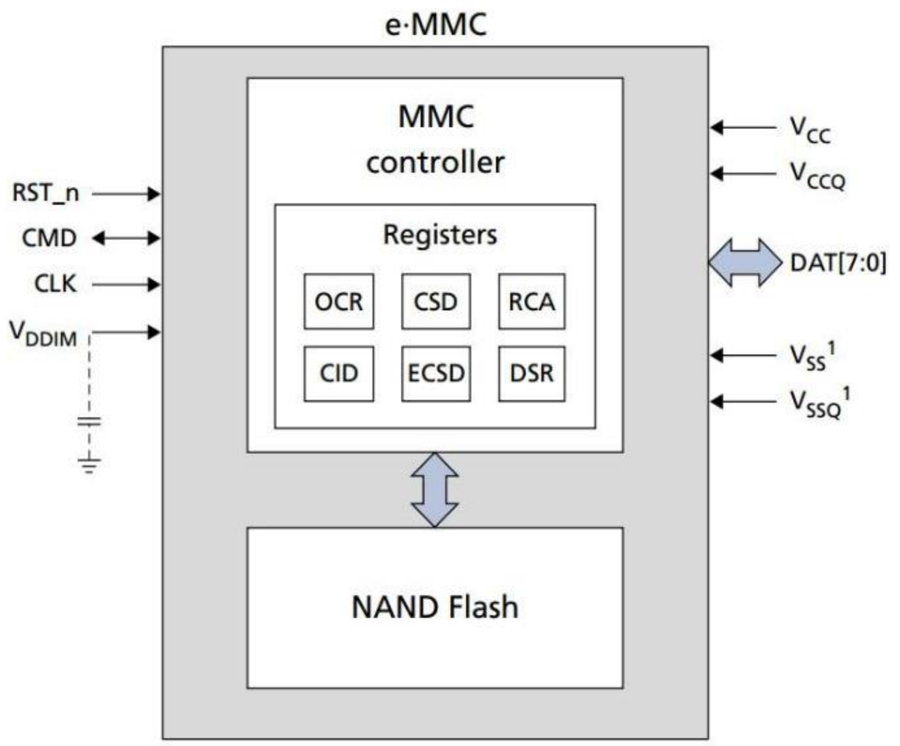
- eMMC Introduction
- eMMC Interface on I.MXRT1052
- eMMC Boot on I.MXRT1052
- eMMC Read/Write on I.MXRT1052
- Reference and FAQ

eMMC Introduction

eMMC概述

eMMC，全称为embedded MultiMediaCard。MultiMediaCard即 MMC，是一种闪存卡（Flash Memory Card）标准，它定义了 MMC 的架构以及访问Flash Memory的接口和协议，而MMC前面加了个embedded，主要就是为了突出它是embedded 在电路板上的。 eMMC 是对MMC 的一个拓展，以满足更高标准的性能、成本、体积、稳定和易用等方面的需求，近年来在手机，平板电脑以及对体积、可靠性和性能要求高的嵌入式领域得到广泛的应用。

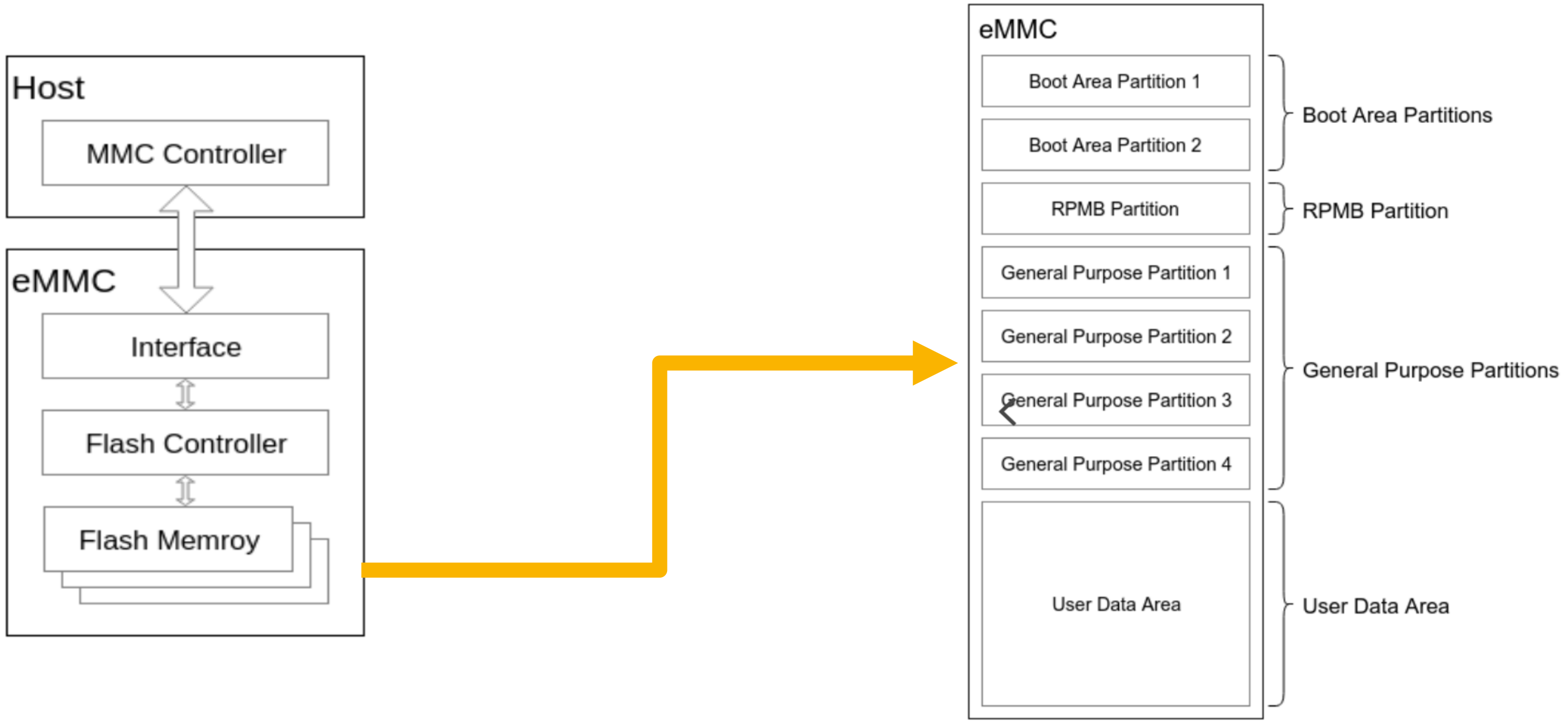




eMMC内部架构

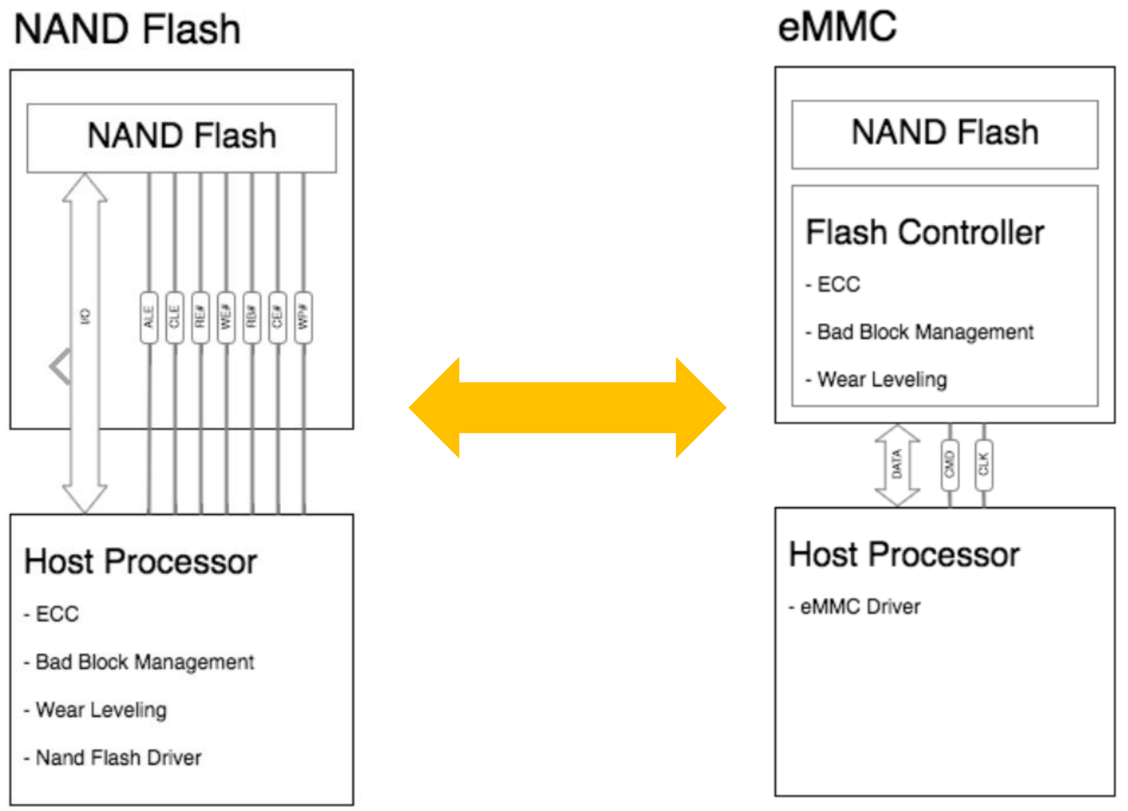
eMMC内部主要可以分为 Flash Memory、Flash Controller 以及 Host Interface 三个部分：

（1）eMMC的Flash Memory都是使用的NandFlash作为存储介质，其内部又可分为BootArea(4MB), RPMB(4MB), General Purpose(默认不存在)和User Data分区(总容量 - 其他分区空间)，注意这几个分区为硬件分区，每个存储空间都是独立寻址的，即访问地址范围为 0 ~partition size，具体访问哪个分区需要主机发送命令来选择PARTITION_ACCESS；



eMMC内部架构

（2）eMMC内部集成了Flash Controller，用于完成擦写均衡、坏块管理、ECC校验等功能。相比于直接将 NAND Flash 接入到 Host 端，eMMC 屏蔽了 NAND Flash 的物理特性，可以减少 Host 端软件的复杂度，让 Host 端专注于上层业务，省去对 NAND Flash 进行特殊的处理。同时，eMMC 通过使用 Cache和 Memory Array 等技术，在读写性能上也比 NAND Flash 要好很多；



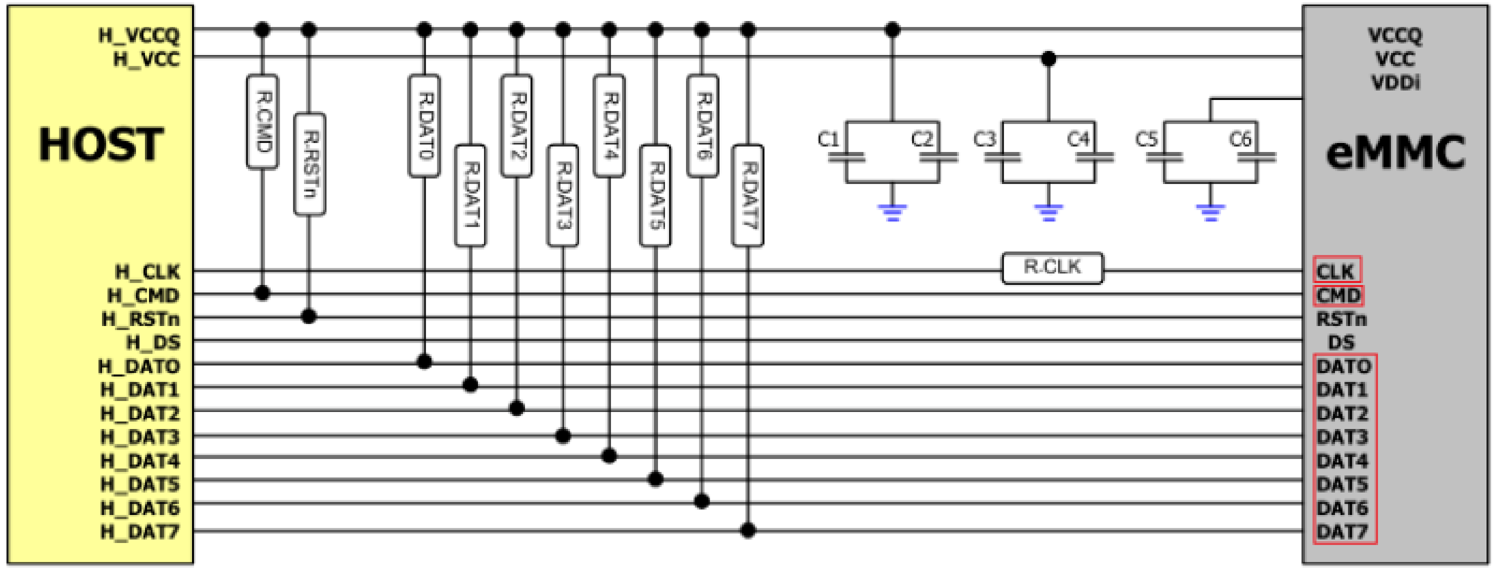
eMMC缺点：

1. 贵；
2. 封装均为BGA

eMMC内部架构

（3）eMMC与Host端的接口为8位并行接口：

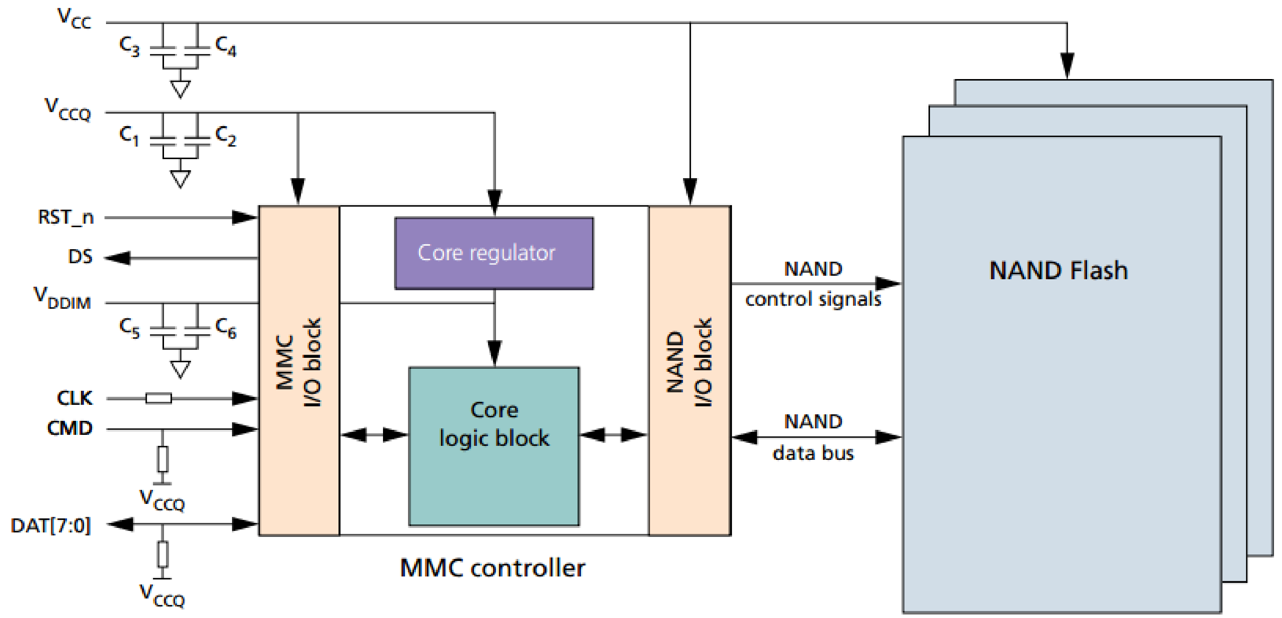
- Clk： 0~200MHz，每个cycle可以在上升沿或下降沿传输，也可以在上升沿和下降沿都传输;
- Data_Strobe：由slave device 发送给host controller，主要用在HS400 mode（5.0规范以后版本才有），频率与clk一致，启用后可以提高数据传输的稳定性，省去总线 tuning 过程;
- CMD：双向信号线，用于从host发送命令给device和device发送response给host，总线上的所有通讯都是由Host 端以一个 Command通过CMD信号发起;
- DAT[0:7]：双向信号线，工作在push pull mode，支持 1bit, 4bit, 8bit传输，默认上电或reset后只有DATA0用于数据传输，完成初始化后，可配置 为DAT0-3（4线模式）或者 DAT0-7（8线模式） 进行数据传输;

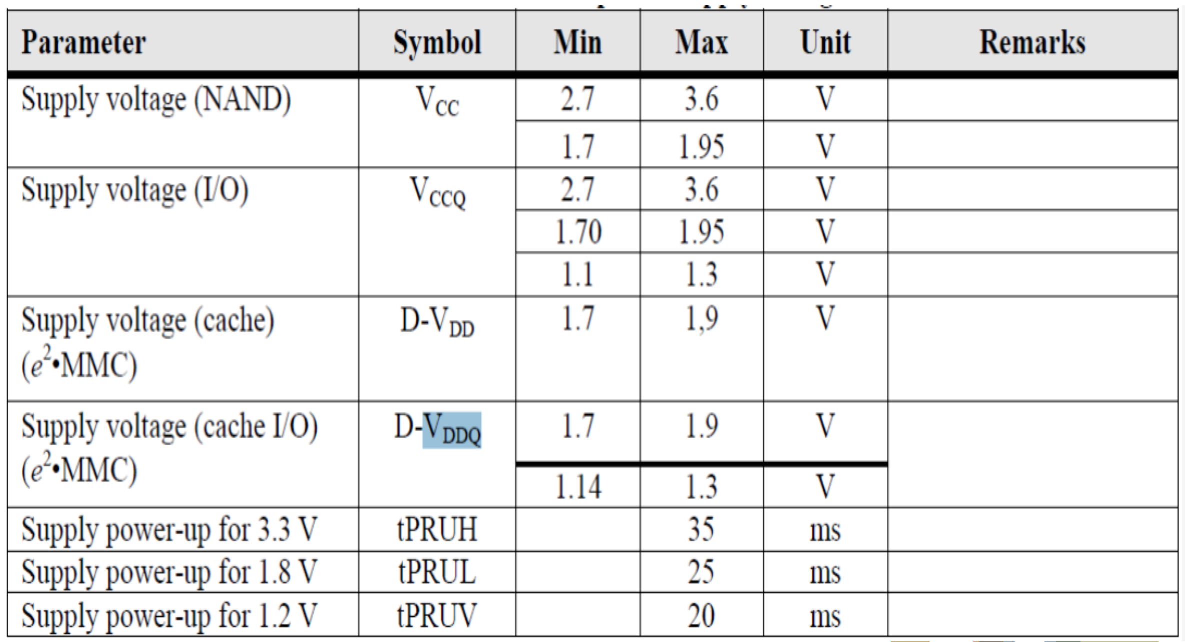


eMMC电源

　　eMMC有两组电源域，其中VCCQ主要用于eMMC IO BLOCK的供电也就是与host接口IO部分的供电，同时也给eMMC内部的Flash控制器供电，VCC则主要给eMMC内部的NandFlash存储介质以及eMMC Flash控制器与NandFlash的接口IO供电；

　　VCC支持两种电压，2.7v~3.6v和1.7~1.95v。而VCCQ则支持三种电压，2.7v~3.6v，1.7v~1.95v和1.1v~1.3v，其中VCCQ（2.7v~3.6v）范围下不支持HS200和HS400速度模式；





Parameter	Symbol	Min	Max	Unit	Remarks
Supply voltage (NAND)	V_{CC}	2.7	3.6	V	
		1.7	1.95	V	
Supply voltage (I/O)	V_{CCQ}	2.7	3.6	V	
		1.70	1.95	V	
		1.1	1.3	V	
Supply voltage (cache) (e^2•MMC)	D-V_{DD}	1.7	1,9	V	
Supply voltage (cache I/O) (e^2•MMC)	D-V_{DDQ}	1.7	1.9	V	
		1.14	1.3	V	
Supply power-up for 3.3 V	tPRUH		35	ms	
Supply power-up for 1.8 V	tPRUL		25	ms	
Supply power-up for 1.2 V	tPRUV		20	ms	

eMMC速率

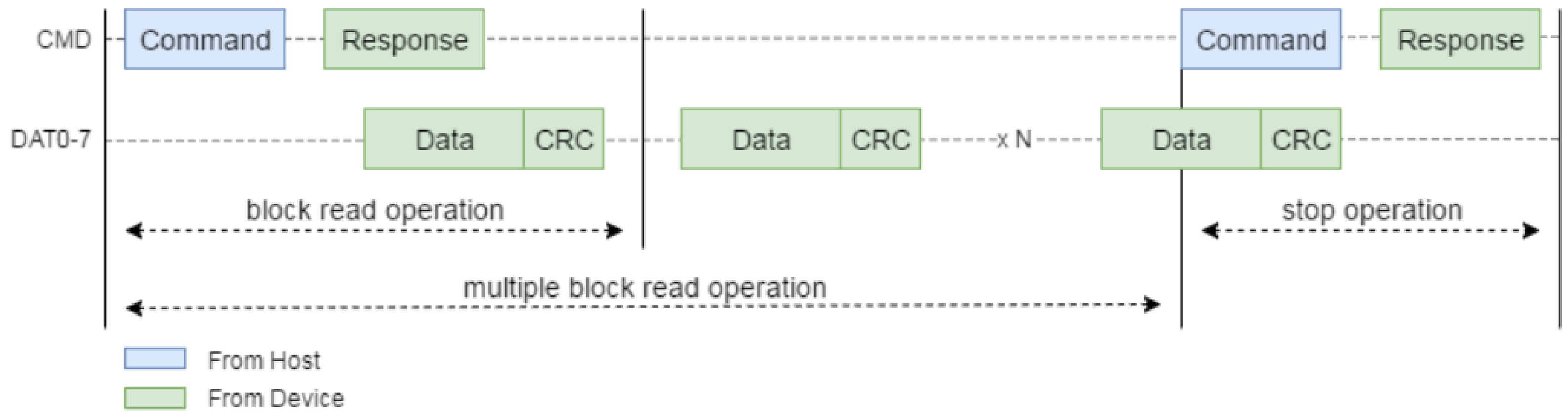
随着eMMC 协议的版本迭代，eMMC 总线的速率越来越高。为了兼容旧版本的 eMMC Device，所有 Devices 在上电启动或者 Reset 后，都会先进入兼容速率模式（Backward Compatible Mode）。在完成 eMMC Devices 的初始化后，Host 可以通过特定的流程，让 Device 进入其他高速率模式，目前支持以下的几种速率模式

Mode	Data Rate	Bus Width	Frequency	Max Data Transfer (x8)
Backward Compatible	Single	x1, x4, x8	0-26 MHz	26 MB/s
High Speed SDR	Single	x1, x4, x8	0-52 MHz	52 MB/s
High Speed DDR	Dual	x4, x8	0-52 MHz	104 MB/s
HS200 eMMC4.5以后支持	Single	x4, x8	0-200 MHz	200 MB/s
HS400 eMMC5.0以后支持	Dual	x8	0-200 MHz	400 MB/s

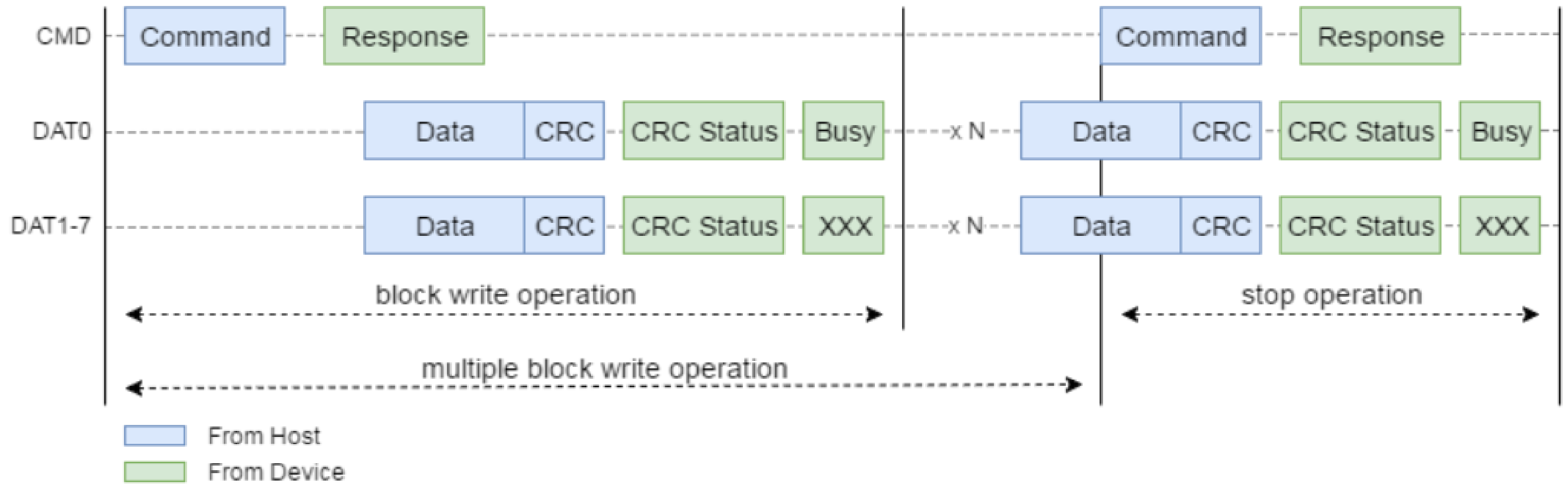
Note:
Extended CSD byte[185] HS_TIMING 寄存器可以配置总线速率模式
Extended CSD byte[183] BUS_WIDTH 寄存器用于配置总线宽度和 Data Strobe

eMMC读写时序

Read Data



Write Data

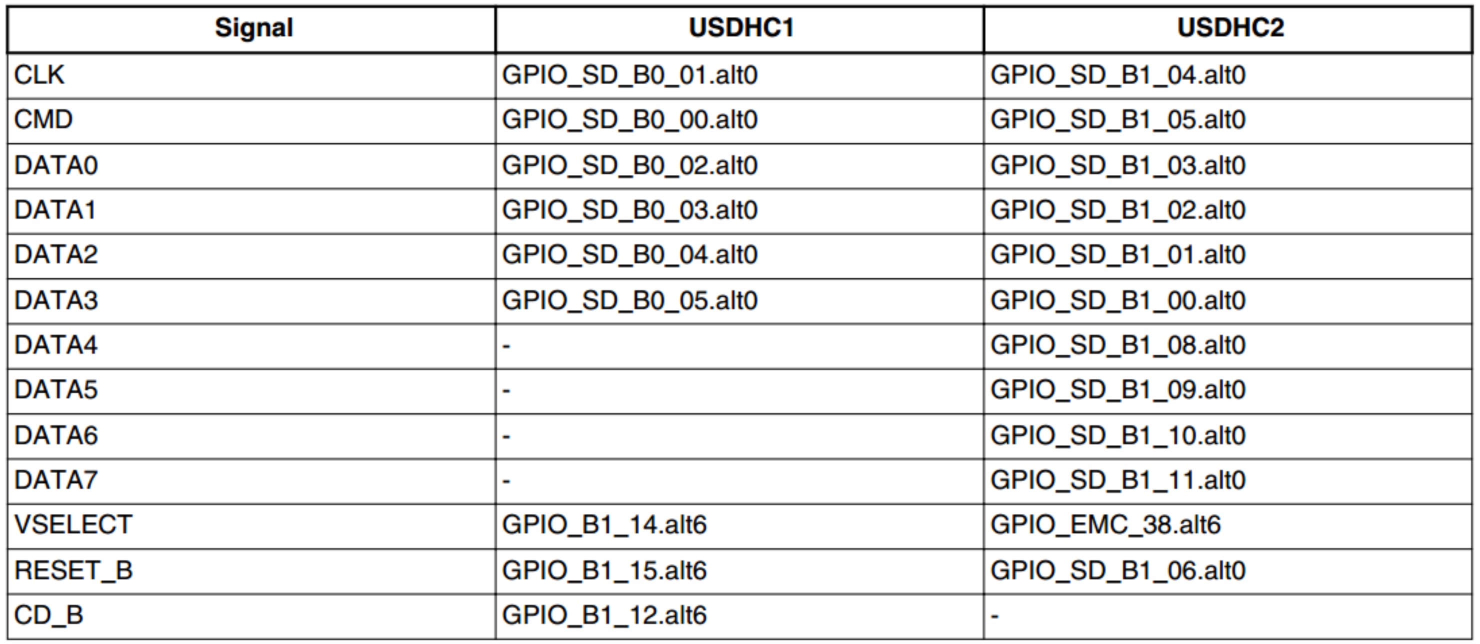


eMMC Interface on I.MXRT1052

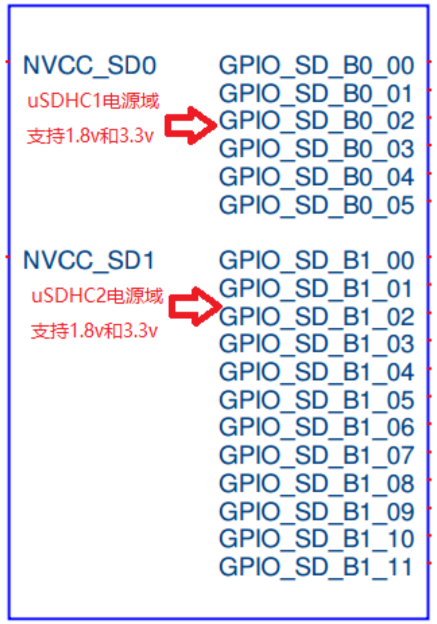
uSDHC databus and power domain

1. i.MXRT1052有两个uSDHC接口都可以接入eMMC device，uSDHC1和uSDHC2，其中uSDHC1支持1-bit和4bit databus模式，而uSDHC2支持1-bit，4-bit和8-bit databus模式，而IO接口则支持1.8v和3.3v电源，其中uSDHC1的接口电平由NVCC_SD0电源域管理，uSDHC2接口电平由NVCC_SD1管理，具体电平由外接设备决定也可以动态切换；

Table 9-33. IOMUX configuration for SD/MMC

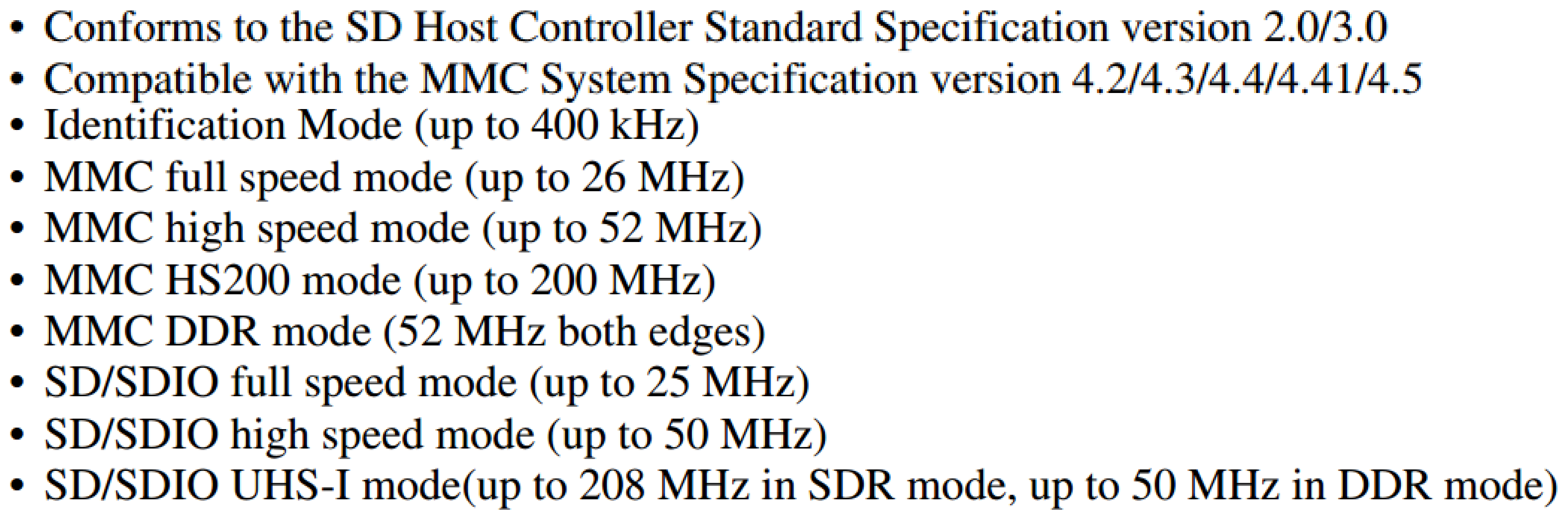


Signal	USDHC1	USDHC2
CLK	GPIO_SD_B0_01.alt0	GPIO_SD_B1_04.alt0
CMD	GPIO_SD_B0_00.alt0	GPIO_SD_B1_05.alt0
DATA0	GPIO_SD_B0_02.alt0	GPIO_SD_B1_03.alt0
DATA1	GPIO_SD_B0_03.alt0	GPIO_SD_B1_02.alt0
DATA2	GPIO_SD_B0_04.alt0	GPIO_SD_B1_01.alt0
DATA3	GPIO_SD_B0_05.alt0	GPIO_SD_B1_00.alt0
DATA4	-	GPIO_SD_B1_08.alt0
DATA5	-	GPIO_SD_B1_09.alt0
DATA6	-	GPIO_SD_B1_10.alt0
DATA7	-	GPIO_SD_B1_11.alt0
VSELECT	GPIO_B1_14.alt6	GPIO_EMC_38.alt6
RESET_B	GPIO_B1_15.alt6	GPIO_SD_B1_06.alt0
CD_B	GPIO_B1_12.alt6	-



uSDHC supported spec. and speed mode

2. i.MXRT1052的uSDHC最高支持SD2.3和MMC4.5规范，速度模式不支持MMC HS400模式（后续RT家族新产品会支持HS400模式，MMC规范升级到5.0）；

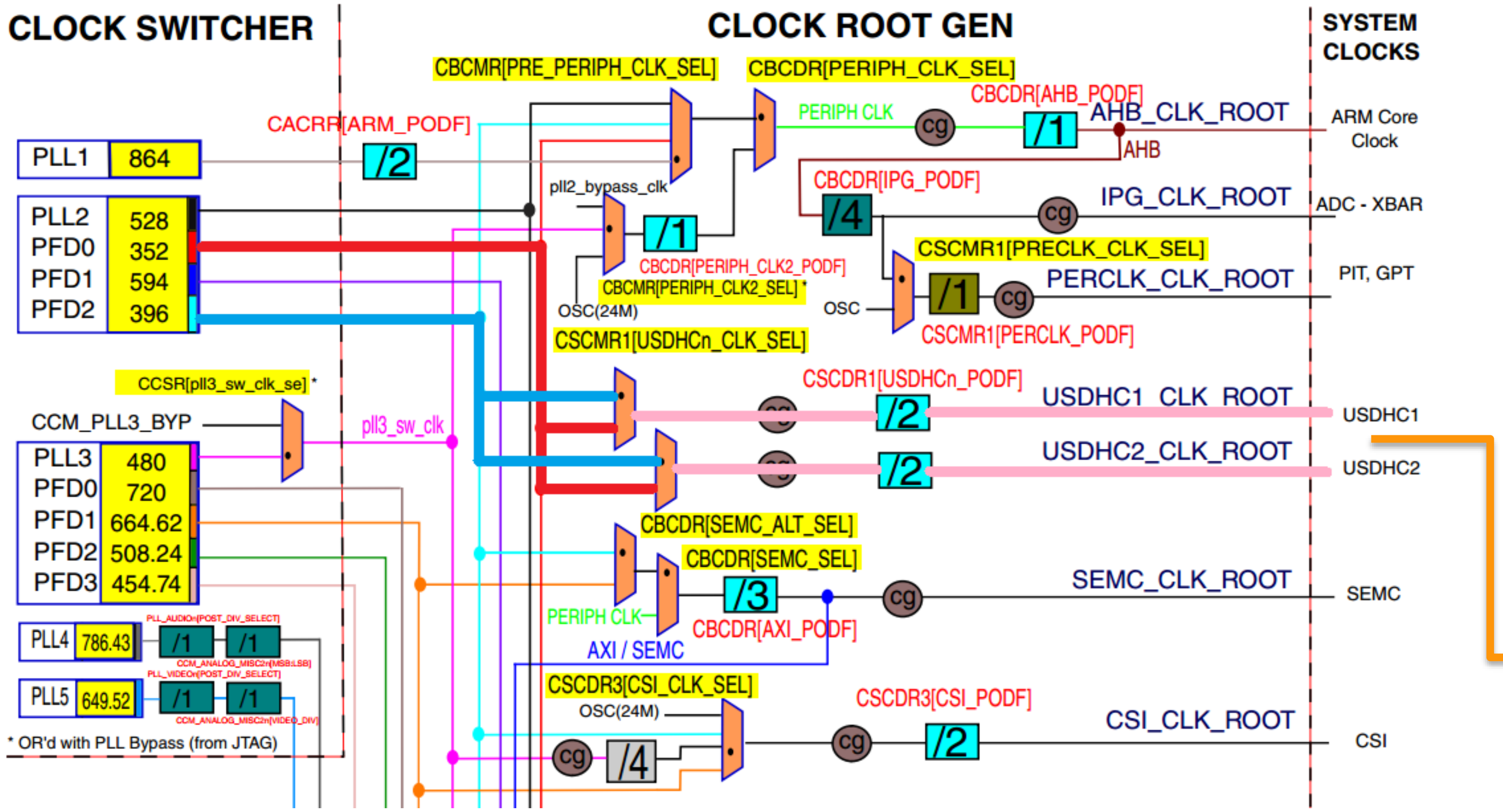


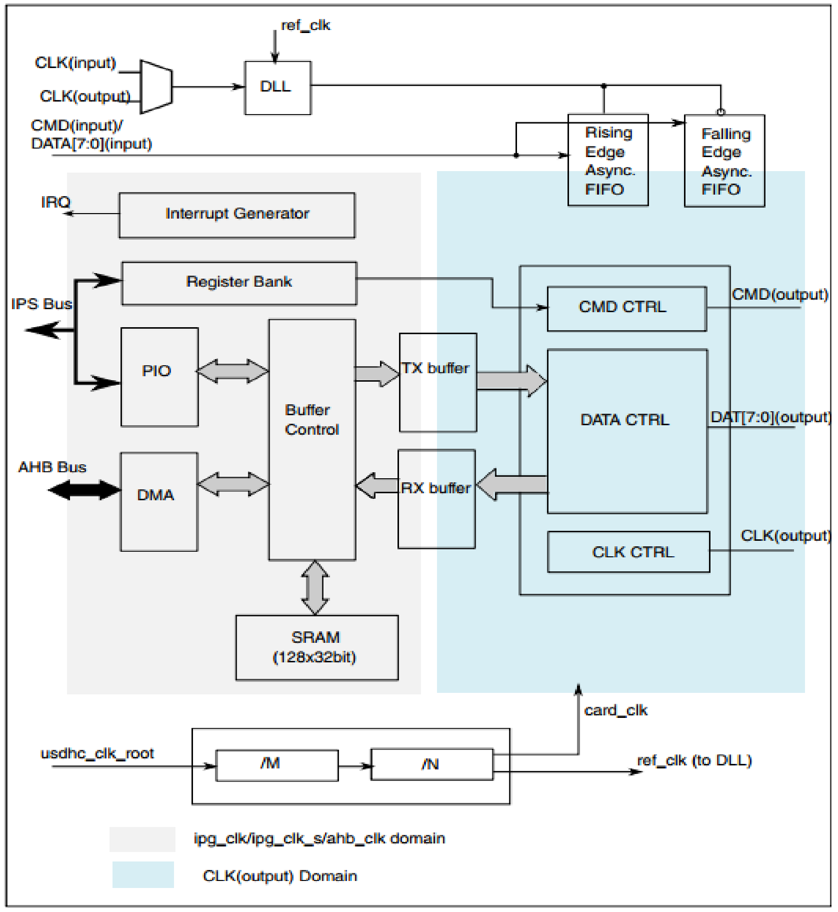
- Conforms to the SD Host Controller Standard Specification version 2.0/3.0
- Compatible with the MMC System Specification version 4.2/4.3/4.4/4.41/4.5
- Identification Mode (up to 400 kHz)
- MMC full speed mode (up to 26 MHz)
- MMC high speed mode (up to 52 MHz)
- MMC HS200 mode (up to 200 MHz)
- MMC DDR mode (52 MHz both edges)
- SD/SDIO full speed mode (up to 25 MHz)
- SD/SDIO high speed mode (up to 50 MHz)
- SD/SDIO UHS-I mode(up to 208 MHz in SDR mode, up to 50 MHz in DDR mode)

uSDHC clock domain

3. uSDHC的时钟源可选择PLL2_PFD0和PLL2_PFD2，最高时钟频率为208MHz；

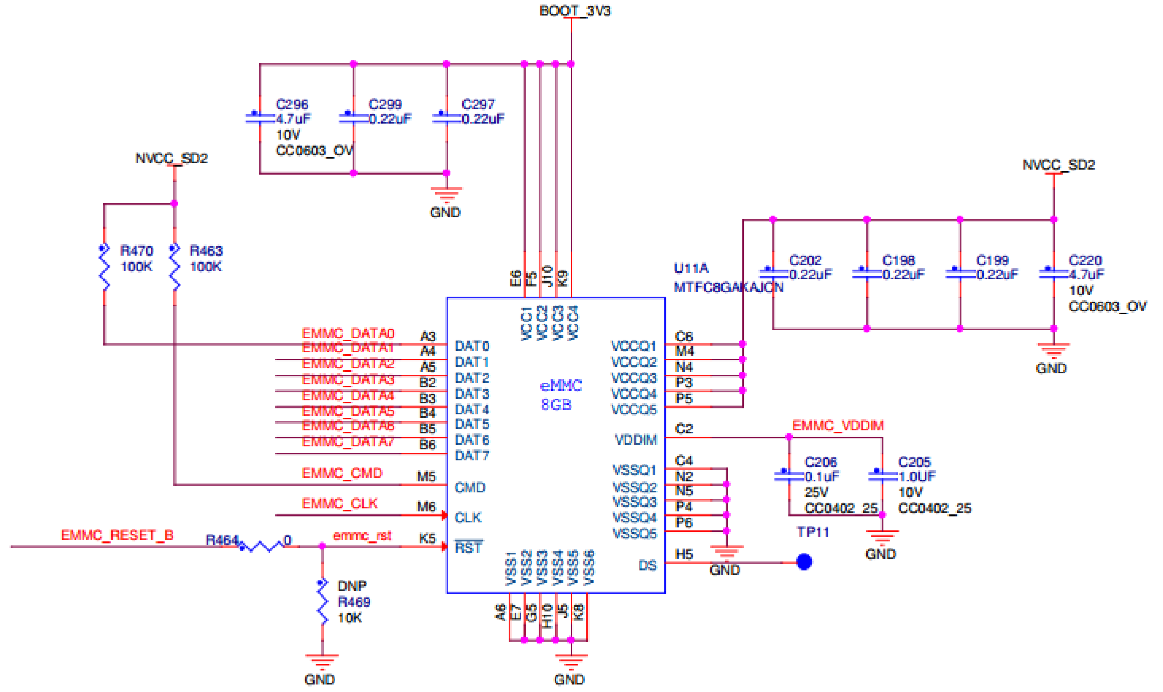
Note：uSDHC时钟推荐选择PFD0，因为RT1052的代码里默认都把PFD2作为SEMC的时钟源，而SDRAM最高为166MHz，这样uSDHC如果选择同一个时钟源则速度受限。

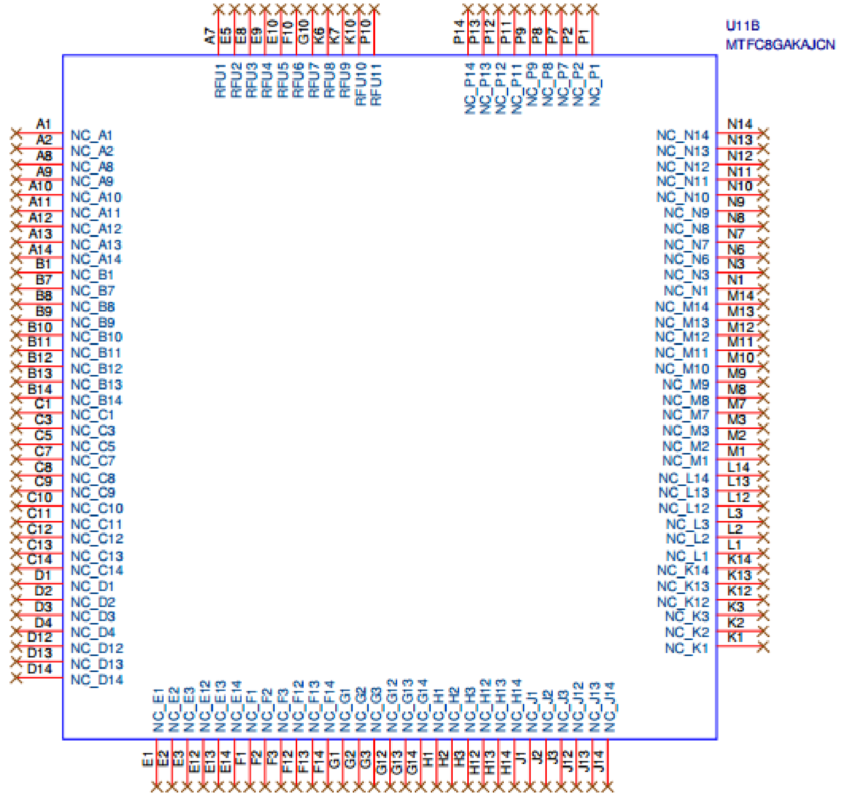




ultra Secure Digital Host Controller Block Diagram

Reference design of eMMC interface with uSDHC





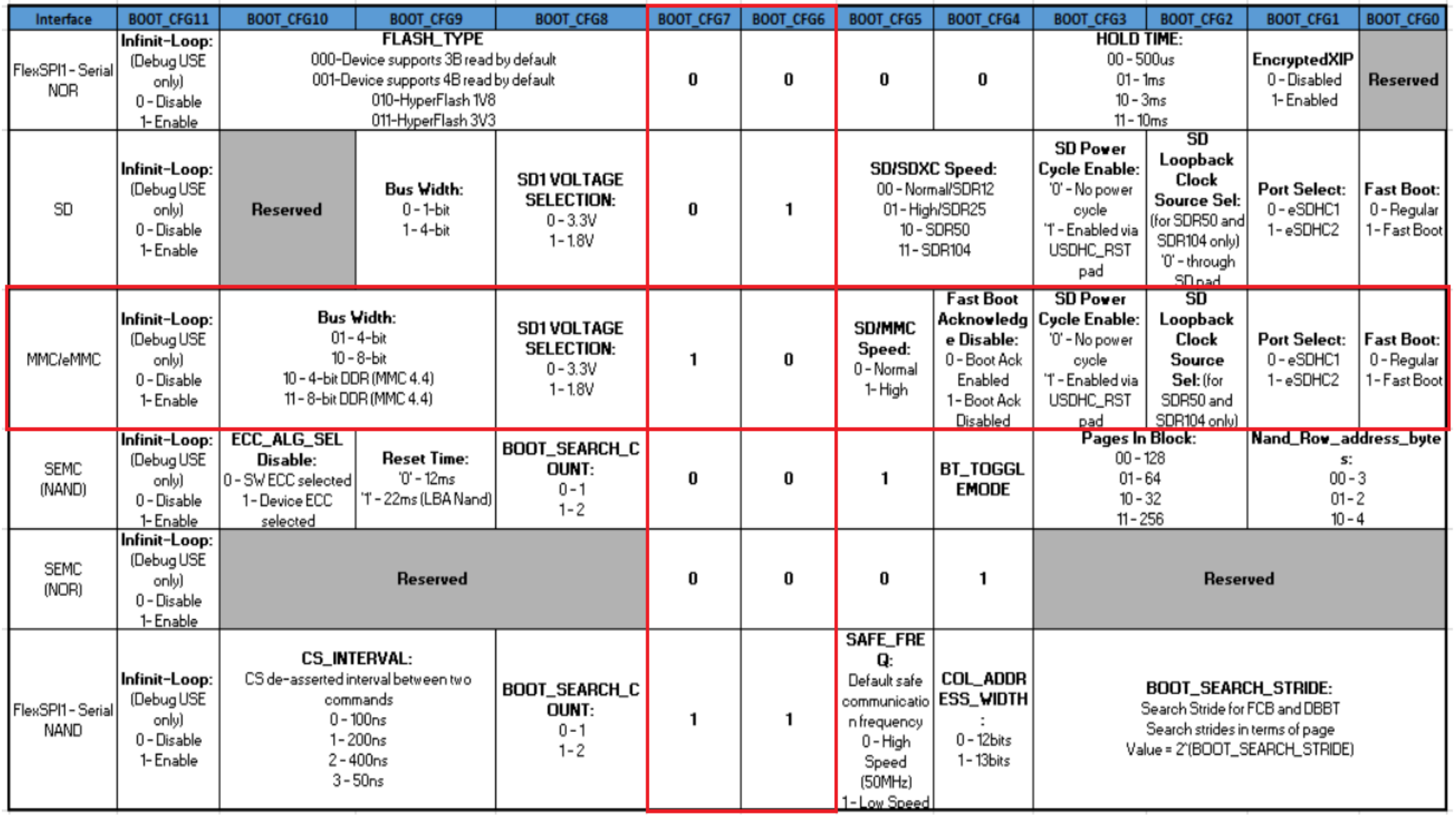
EMMC MEMORY

eMMC Boot on I.MXRT1052

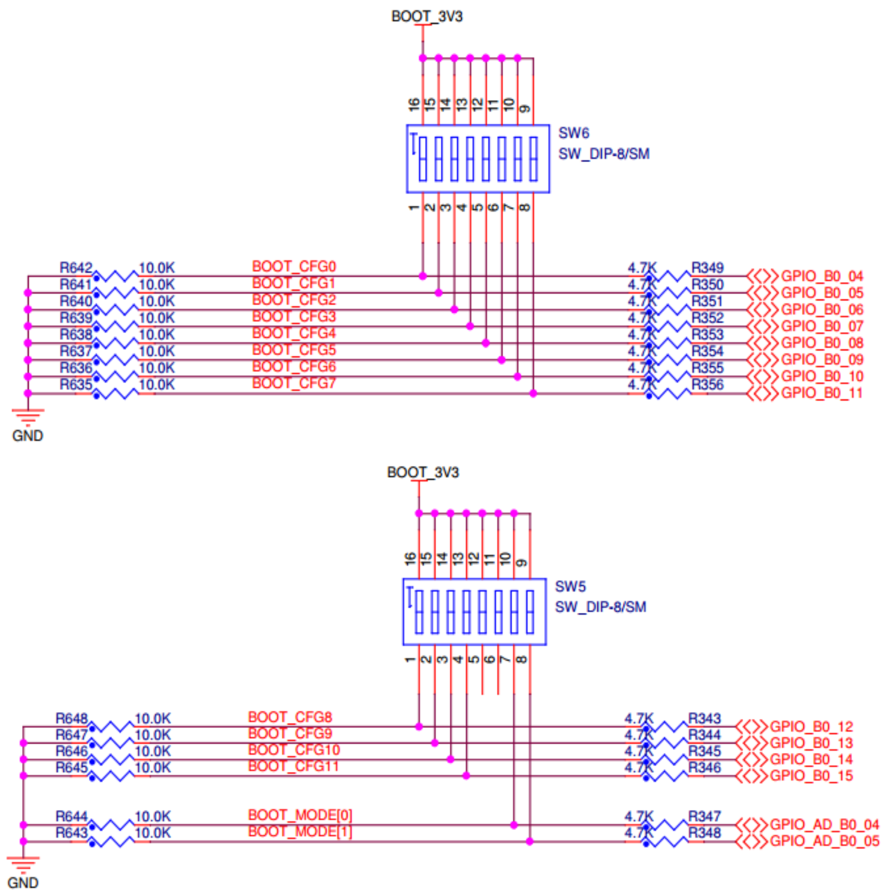
eMMC Boot Config Map

1. 支持MMC4.4及以下规范的eMMC设备可以通过RT1052的uSDHC1或uSDHC2接口启动，具体启动配置可以通过外部GPIO的Boot_Cfg管脚（见下图）或者内部Fuse（见RT1050RM Table 9-30）来决定;

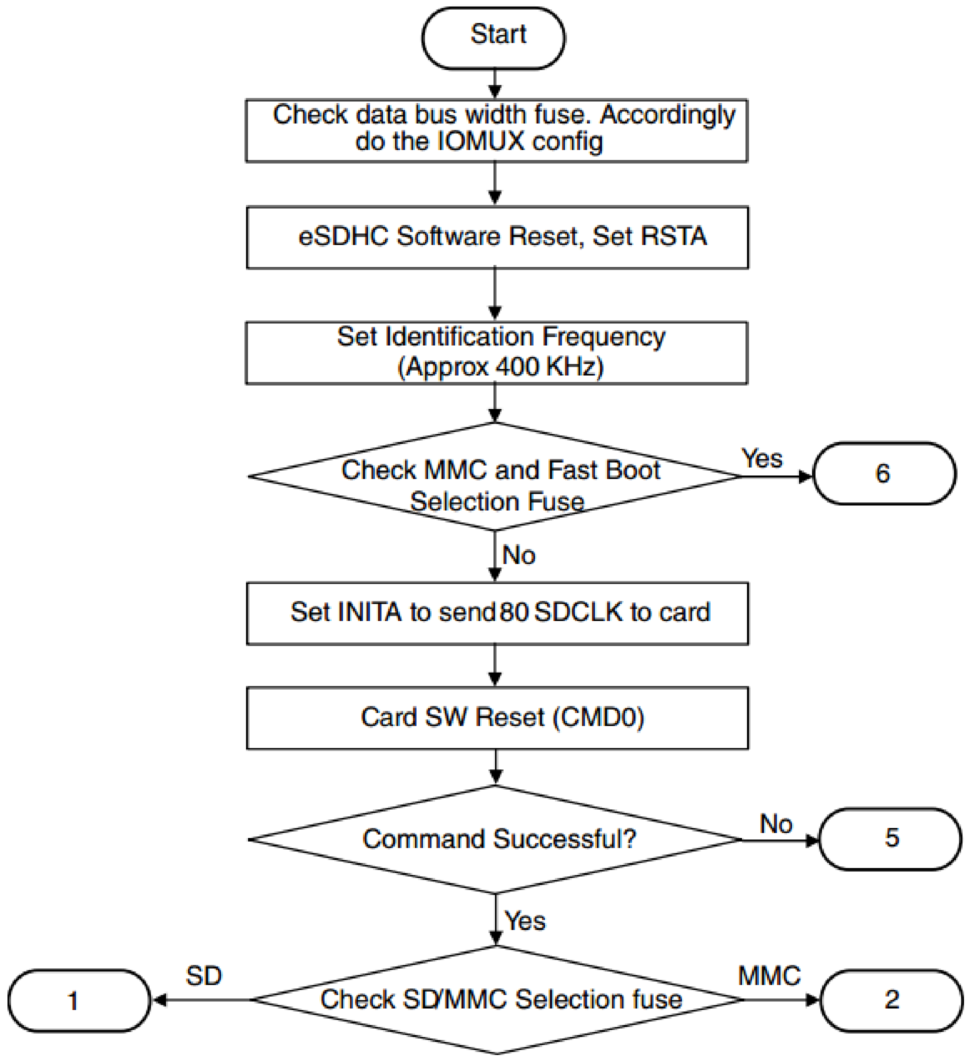
Note: Normal Speed = 20MHz, High Speed = 40MHz

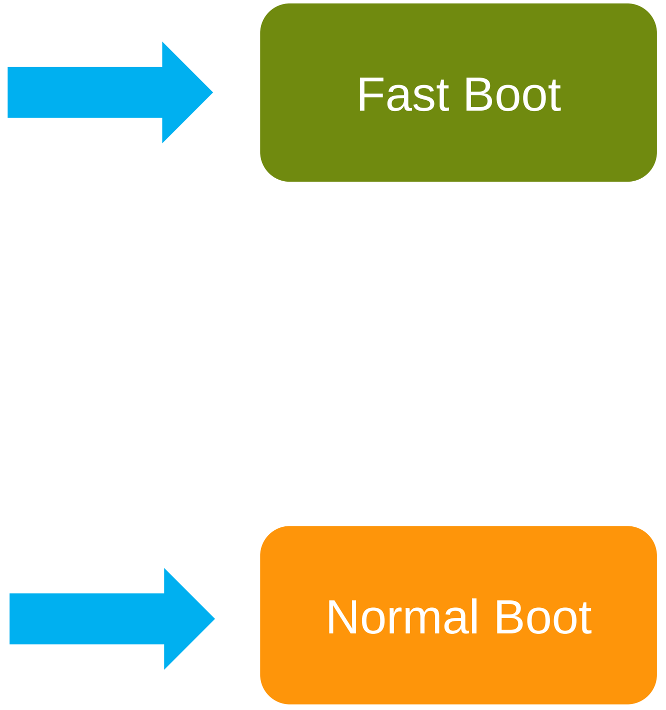


Interface	BOOT_CFG11	BOOT_CFG10	BOOT_CFG9	BOOT_CFG8	BOOT_CFG7	BOOT_CFG6	BOOT_CFG5	BOOT_CFG4	BOOT_CFG3	BOOT_CFG2	BOOT_CFG1	BOOT_CFG0
FlexSPI1 - Serial NOR	**Infinit-Loop:** (Debug USE only) 0 - Disable 1- Enable	**FLASH_TYPE** 000-Device supports 3B read by default 001-Device supports 4B read by default 010-HyperFlash 1V8 011-HyperFlash 3V3			**0**	**0**	**0**	**0**	**HOLD TIME:** 00 - 500us 01 - 1ms 10 - 3ms 11 - 10ms		**EncryptedXIP** 0 - Disabled 1- Enabled	**Reserved**
SD	**Infinit-Loop:** (Debug USE only) 0 - Disable 1- Enable	**Reserved**	**Bus Width:** 0 - 1-bit 1 - 4-bit	**SD1 VOLTAGE SELECTION:** 0 - 3.3V 1 - 1.8V	**0**	**1**	**SD/SDXC Speed:** 00 - Normal/SDR12 01 - High/SDR25 10 - SDR50 11 - SDR104		**SD Power Cycle Enable:** '0' - No power cycle '1' - Enabled via USDHC_RST pad	**SD Loopback Clock Source Sel:** (for SDR50 and SDR104 only) '0' - through SD pad	**Port Select:** 0 - eSDHC1 1 - eSDHC2	**Fast Boot:** 0 - Regular 1 - Fast Boot
MMC/eMMC	**Infinit-Loop:** (Debug USE only) 0 - Disable 1- Enable	**Bus Width:** 01 - 4-bit 10 - 8-bit 10 - 4-bit DDR (MMC 4.4) 11 - 8-bit DDR (MMC 4.4)		**SD1 VOLTAGE SELECTION:** 0 - 3.3V 1 - 1.8V	**1**	**0**	**SD/MMC Speed:** 0 - Normal 1- High	**Fast Boot Acknowledge Disable:** 0 - Boot Ack Enabled 1 - Boot Ack Disabled	**SD Power Cycle Enable:** '0' - No power cycle '1' - Enabled via USDHC_RST pad	**SD Loopback Clock Source Sel:** (for SDR50 and SDR104 only)	**Port Select:** 0 - eSDHC1 1 - eSDHC2	**Fast Boot:** 0 - Regular 1 - Fast Boot
SEMC (NAND)	**Infinit-Loop:** (Debug USE only) 0 - Disable 1- Enable	**ECC_ALG_SEL Disable:** 0 - SW ECC selected 1 - Device ECC selected	**Reset Time:** '0' - 12ms '1' - 22ms (LBA Nand)	**BOOT_SEARCH_COUNT:** 0 - 1 1 - 2	**0**	**0**	**1**	**BT_TOGGLEMODE**	**Pages In Block:** 00 - 128 01 - 64 10 - 32 11 - 256		**Nand_Row_address_bytes:** 00 - 3 01 - 2 10 - 4	
SEMC (NOR)	**Infinit-Loop:** (Debug USE only) 0 - Disable 1- Enable	**Reserved**			**0**	**0**	**0**	**1**	**Reserved**			
FlexSPI1 - Serial NAND	**Infinit-Loop:** (Debug USE only) 0 - Disable 1- Enable	**CS_INTERVAL:** CS de-asserted interval between two commands 0 - 100ns 1 - 200ns 2 - 400ns 3 - 50ns		**BOOT_SEARCH_COUNT:** 0 - 1 1 - 2	**1**	**1**	**SAFE_FREQ:** Default safe communication frequency 0 - High Speed (50MHz) 1 - Low Speed	**COL_ADDRESS_WIDTH:** 0 - 12bits 1 - 13bits	**BOOT_SEARCH_STRIDE:** Search Stride for FCB and DBBT Search strides in terms of page Value = 2^(BOOT_SEARCH_STRIDE)			

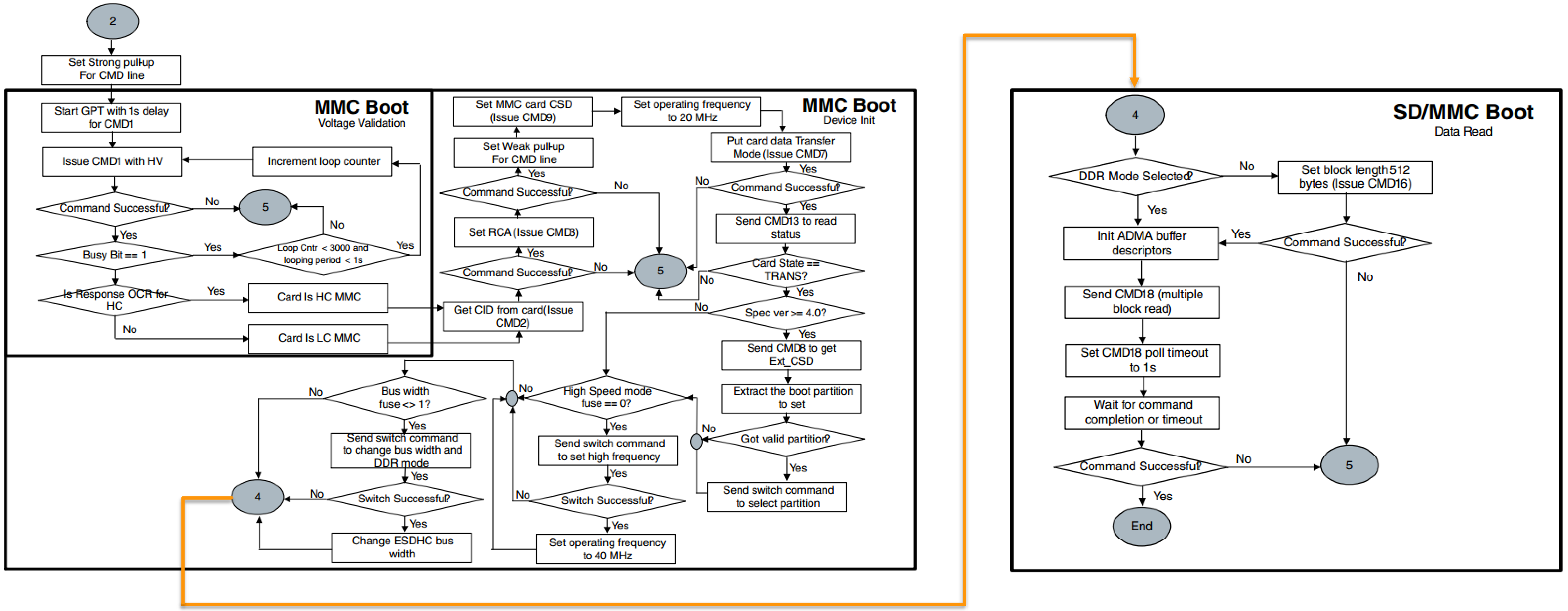


eMMC boot flow


Start
Check data bus width fuse. Accordingly do the IOMUX config
eSDHC Software Reset, Set RSTA
Set Identification Frequency (Approx 400 KHz)
Check MMC and Fast Boot Selection Fuse
Yes
6
No
Set INITA to send 80 SDCLK to card
Card SW Reset (CMD0)
Command Successful?
No
5
Yes
Check SD/MMC Selection fuse
SD
1
MMC
2



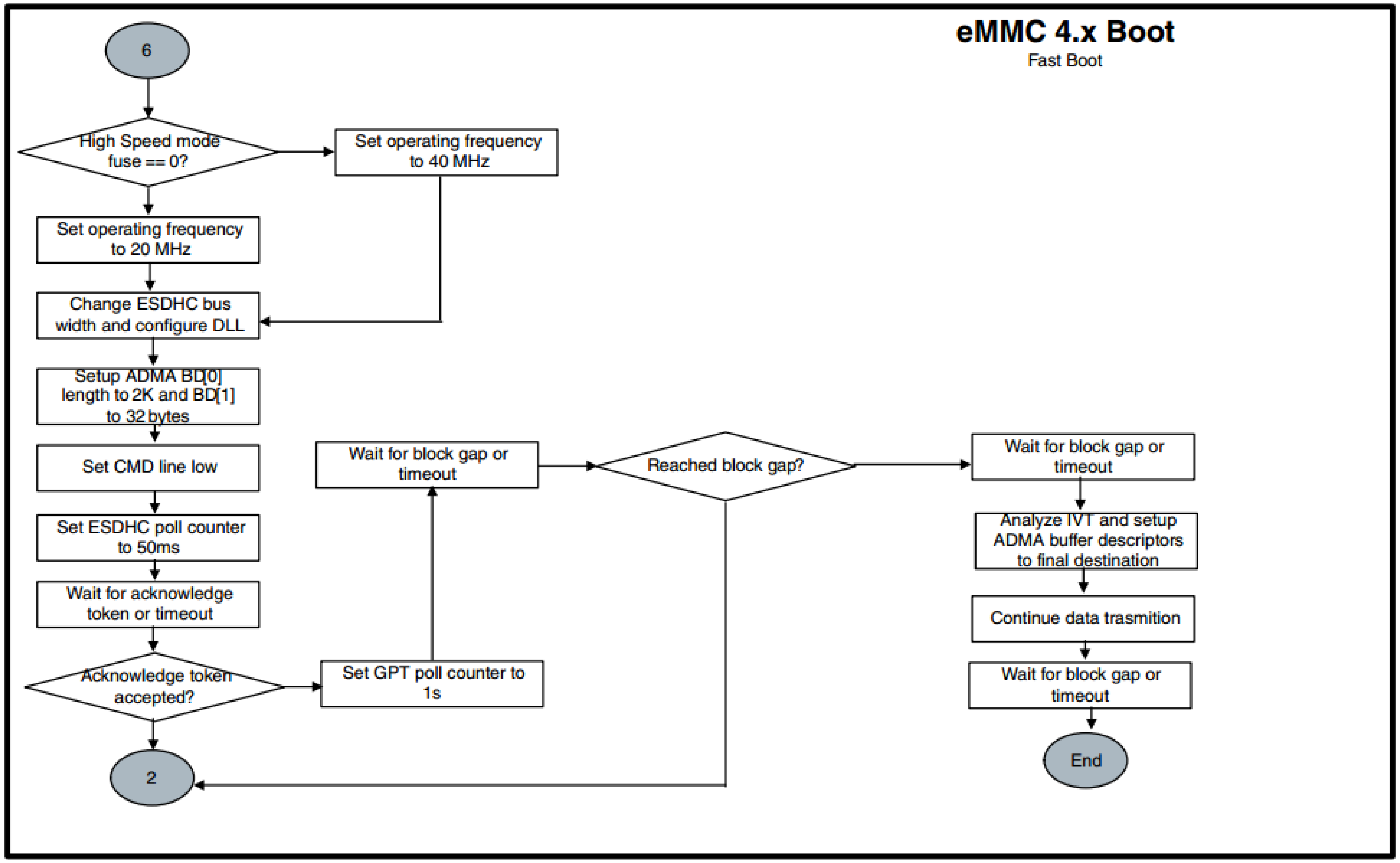
eMMC boot flow – Normal Boot



2
Set Strong pull-up For CMD line
MMC Boot
Voltage Validation
Start GPT with 1s delay for CMD1
Issue CMD1 with HV
Increment loop counter
Command Successful?
No
5
Yes
No
Busy Bit == 1
Yes
Loop Cntr < 3000 and looping period < 1s
Yes
Is Response OCR for HC
Yes
Card Is HC MMC
No
Card Is LC MMC
Set MMC card CSD (Issue CMD9)
Set operating frequency to 20 MHz
MMC Boot
Device Init
Set Weak pull-up For CMD line
Put card data Transfer Mode (Issue CMD7)
Command Successful?
No
Set RCA (Issue CMD3)
Command Successful?
5
Get CID from card (Issue CMD2)
Send CMD13 to read status
Card State == TRANS?
Spec ver >= 4.0?
Send CMD8 to get Ext_CSD
Extract the boot partition to set
Got valid partition?
Send switch command to select partition
High Speed mode fuse == 0?
Send switch command to set high frequency
Switch Successful?
Set operating frequency to 40 MHz
Bus width fuse <> 1?
Send switch command to change bus width and DDR mode
Switch Successful?
Change ESDHC bus width
4
SD/MMC Boot
Data Read
4
DDR Mode Selected?
Set block length 512 bytes (Issue CMD16)
Command Successful?
Init ADMA buffer descriptors
Send CMD18 (multiple block read)
Set CMD18 poll timeout to 1s
Wait for command completion or timeout
Command Successful?
5
End

Note:
RT1052的ROM支持eMMC不同分区的启动，通过eMMC的Extend-CSD寄存器的bit[179]即Partition Configuration来决定.

eMMC boot flow – Fast Boot


eMMC 4.x Boot
Fast Boot
6
High Speed mode fuse == 0?
Set operating frequency to 40 MHz
Set operating frequency to 20 MHz
Change ESDHC bus width and configure DLL
Setup ADMA BD[0] length to 2K and BD[1] to 32 bytes
Set CMD line low
Set ESDHC poll counter to 50ms
Wait for acknowledge token or timeout
Acknowledge token accepted?
2
Set GPT poll counter to 1s
Wait for block gap or timeout
Reached block gap?
Wait for block gap or timeout
Analyze IVT and setup ADMA buffer descriptors to final destination
Continue data trasmition
Wait for block gap or timeout
End

eMMC Boot Image Format

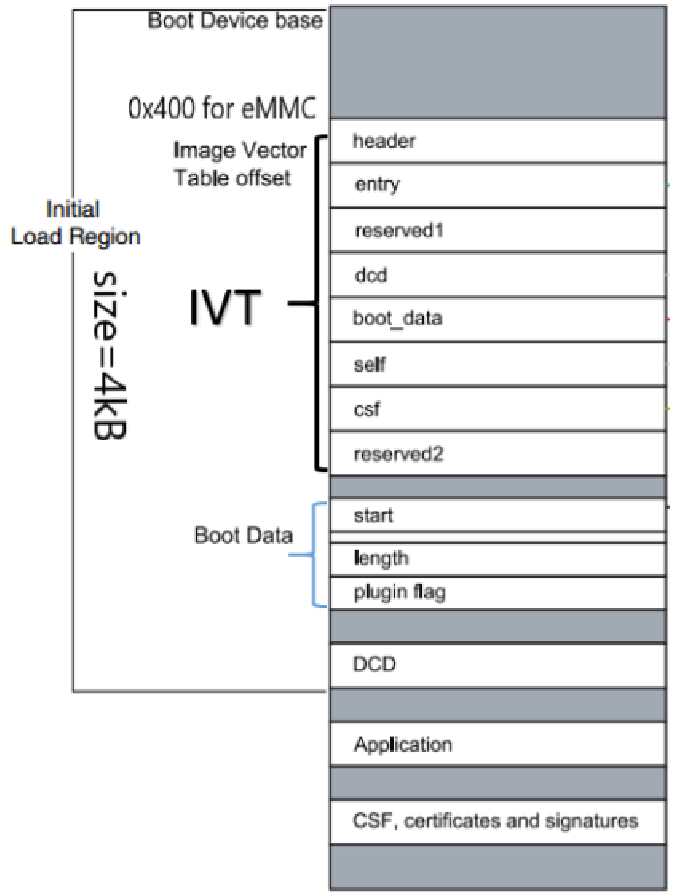
2. 完整的Bootable Image必须包含如下4个部分，IVT，BootData, DCD和用户的应用代码，其中对eMMC启动来讲IVT必须放置在其启动分区里面固定的0x400偏移地址供RT1052的ROM来读取并解析，前3个部分需预留eMMC的前4KB空间来分配它们，用户代码放在4KB后面任意地址即可。至于如何生成该完整的Bootable Image，NXP提供了一整套工具（官方和第三方都可支持），用户只需关注自己的应用代码部分，其余3个部分可由该套工具来生成并合并成为最终一个可下载可启动的完整image；

- Image vector table—a list of pointers located at a fixed address that the ROM examines to determine where the other components of the program image are located.
- Boot data—a table that indicates the program image location, program image size in bytes, and the plugin flag.
- Device configuration data—IC configuration data.
- User code and data.

Image Vector Table Offset and Initial Load Region Size

Boot Device Type	Image Vector Table Offset
FlexSPI NOR/SEMC NOR	4 Kbyte = 0x1000 bytes
SD/MMC/eSD/eMMC/SDXC	1 Kbyte = 0x400 bytes
SPI NOR/EEPROM/SEMC NAND/ FlexSPI NAND	1 Kbyte = 0x400 bytes

Boot Device Memory



eMMC Non-XIP Boot

3. 由于eMMC不支持XIP（片上执行）模式，它只能作为非易失存储介质存放代码和数据，上电后由RT1052的ROM根据IVT、BootData和DCD的配置信息将代码和数据拷贝到MCU内部或者外部RAM指定的地址空间中执行，就此引出eMMC的两种启动模式：

➢ eMMC --> 内部RAM

上电后代码和数据从eMMC拷贝到内部RAM（ITCM，DTCM和OCRAM）里执行，这种方式最简单，但是受限内部RAM的空间，RT1052的内部RAM默认配置大小128KB ITCM，128KB DTCM和256KB OCRAM，如果用户的代码超过128KB或者256KB，则可以通过如下两种方式解决：

① 烧写内部eFuse熔丝配置，改变默认的内部RAM空间大小分配（注意：OCRAM至少要保留64KB，不能全配置成TCM），此种方式只能配置一次，eFuse写完一次之后不能更改了；

② 使用DCD配置脚本动态修改内部RAM分配（通过修改IOMUXC_GPR_GPR16和IOMUXC_GPR_GPR17两个寄存器），上电后ROM会先执行DCD脚本动态重新分配内部RAM，然后再执行拷贝工作。

➢ eMMC --> 外部RAM（以SDRAM为例）

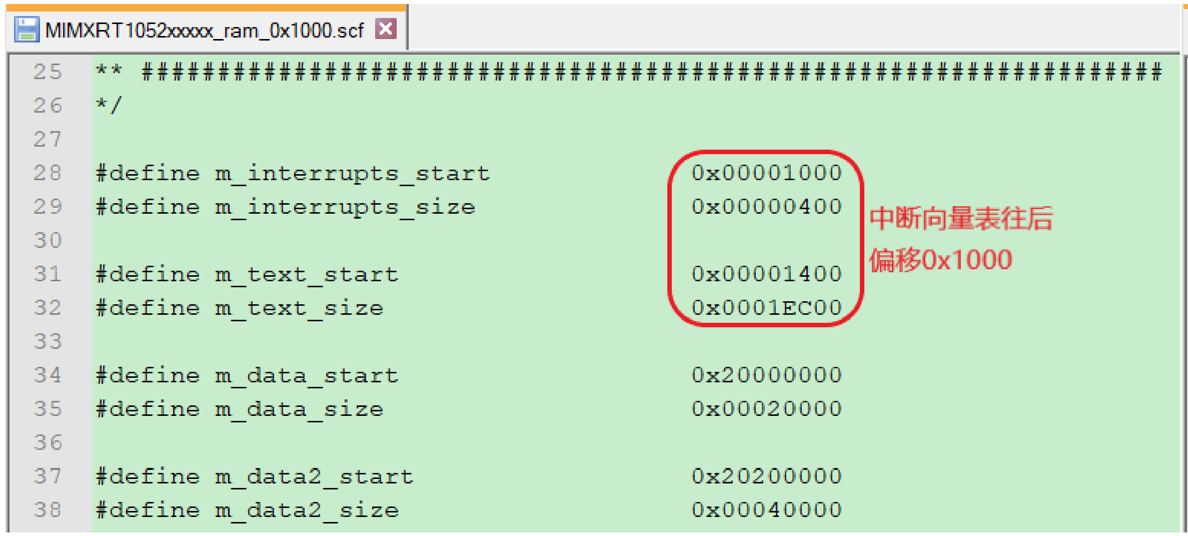
刚上电时由于外部SDRAM还没有初始化不能访问，直接拷贝的话会操作失败，也需要使用DCD配置脚本在拷贝之前对SDRAM进行初始化使能并映射好读写访问空间，再执行拷贝即可。

Note：

① 受限于RT1052 ROM可使用的uSDHC内部ADMA Buffer描述符资源，eMMC Non-XIP Boot可自动拷贝的代码大小最大为32MB；

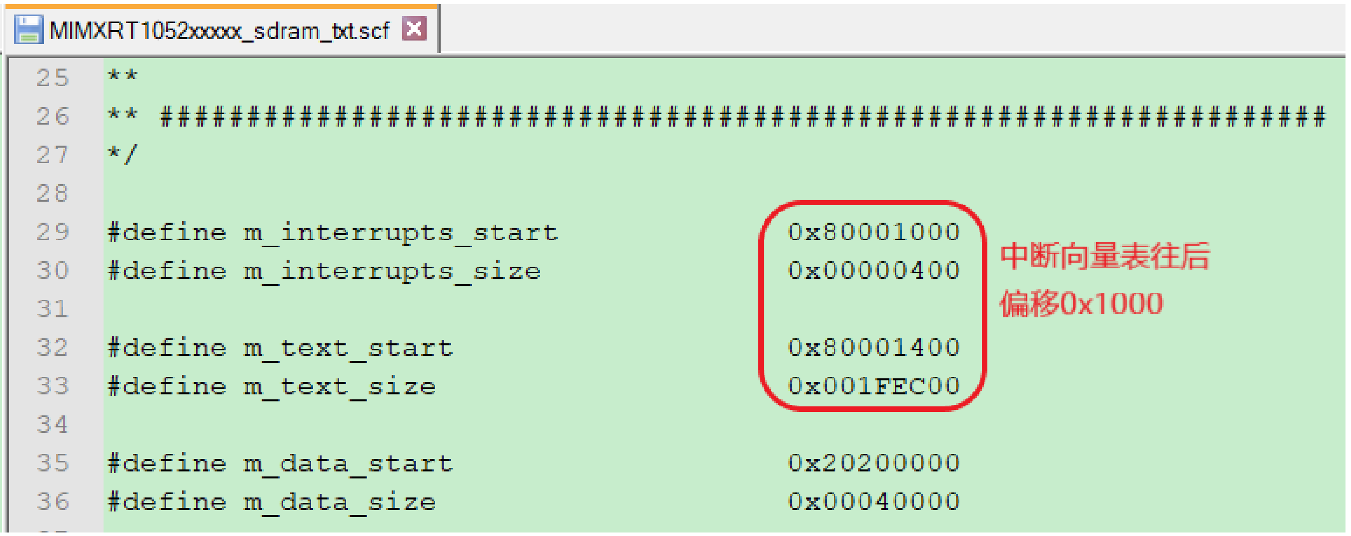
eMMC Non-XIP Boot

4. 上电拷贝时，ROM会把前4KB的IVT、BootData和DCD配置信息也拷贝到RAM里，所以用户在开发代码时要注意修改链接文件或分散加载文件，将用户代码的首地址至少往后偏移4KB的空间（给上述三部分信息预留出4KB空间），下面两图分别为代码执行在ITCM（首地址为0x0000_0000）和SDRAM（首地址为0x8000_0000）两种模式下的Keil分散加载文件配置；



MIMXRT1052xxxxx_ram_0x1000.scf

```
** ###################################################################
*/

#define m_interrupts_start             0x00001000
#define m_interrupts_size              0x00000400

#define m_text_start                   0x00001400
#define m_text_size                    0x0001EC00

#define m_data_start                   0x20000000
#define m_data_size                    0x00020000

#define m_data2_start                  0x20200000
#define m_data2_size                   0x00040000
```

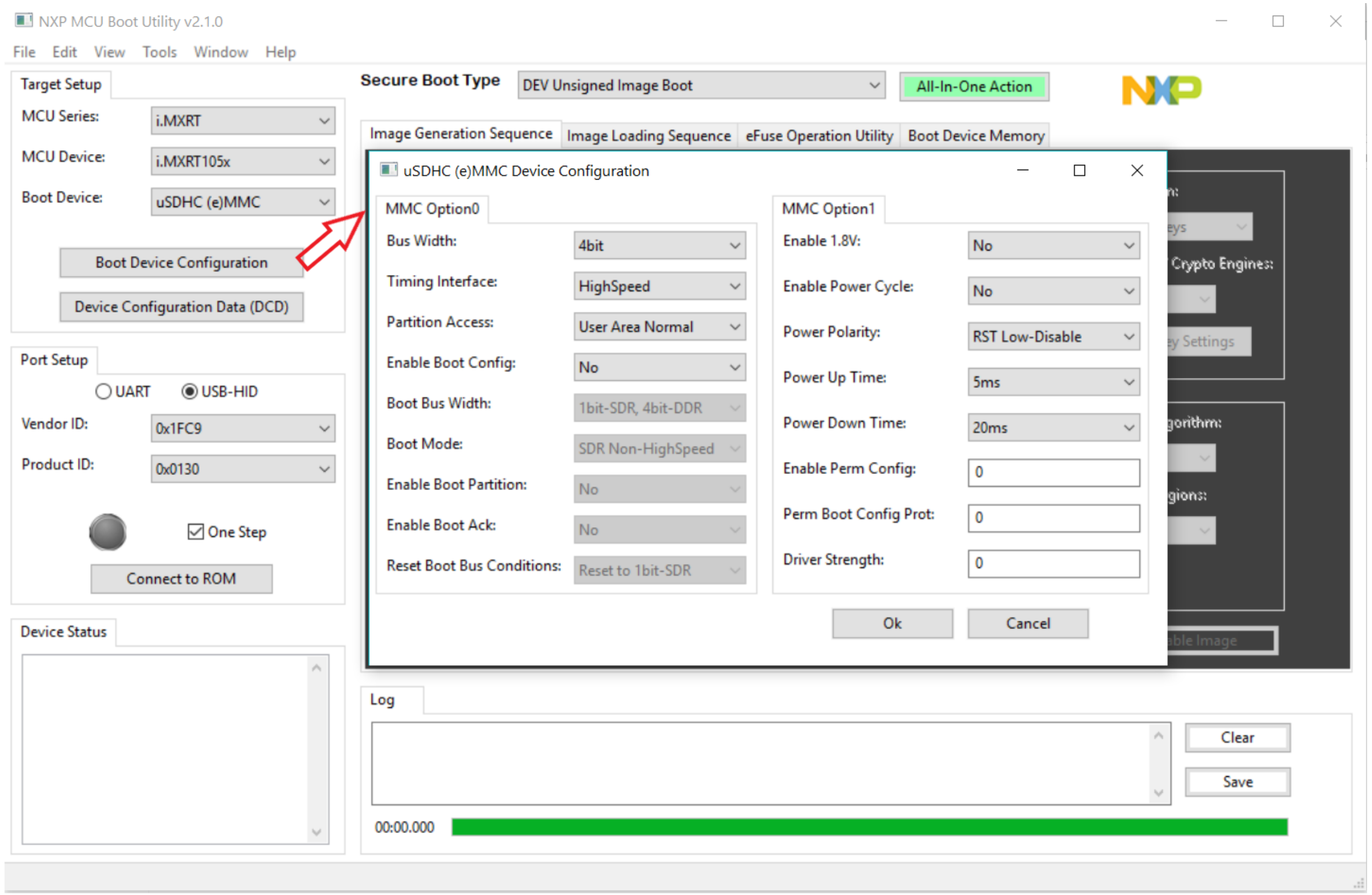


MIMXRT1052xxxxx_sdram_txt.scf

```
**
** ###################################################################
*/

#define m_interrupts_start             0x80001000
#define m_interrupts_size              0x00000400

#define m_text_start                   0x80001400
#define m_text_size                    0x001FEC00

#define m_data_start                   0x20200000
#define m_data_size                    0x00040000
```

eMMC Boot Hands-On

5. 本实验使用第三方工具MCUBootUtility来生成和下载Bootable Image（不要使用该工具原生版本，烧录eMMC有bug），可以直接从本文档最后的下载链接直接下载。打开该工具之后，MCU Device和Boot Device如下图选择，至于Boot Device Configuration则需要根据你的硬件来实际选择；


NXP MCU Boot Utility v2.1.0

Target Setup

MCU Series:	i.MXRT
MCU Device:	i.MXRT105x
Boot Device:	uSDHC (e)MMC

Port Setup: UART [✓ USB-HID]

Vendor ID:	0x1FC9
Product ID:	0x0130

Secure Boot Type: DEV Unsigned Image Boot

uSDHC (e)MMC Device Configuration

MMC Option0

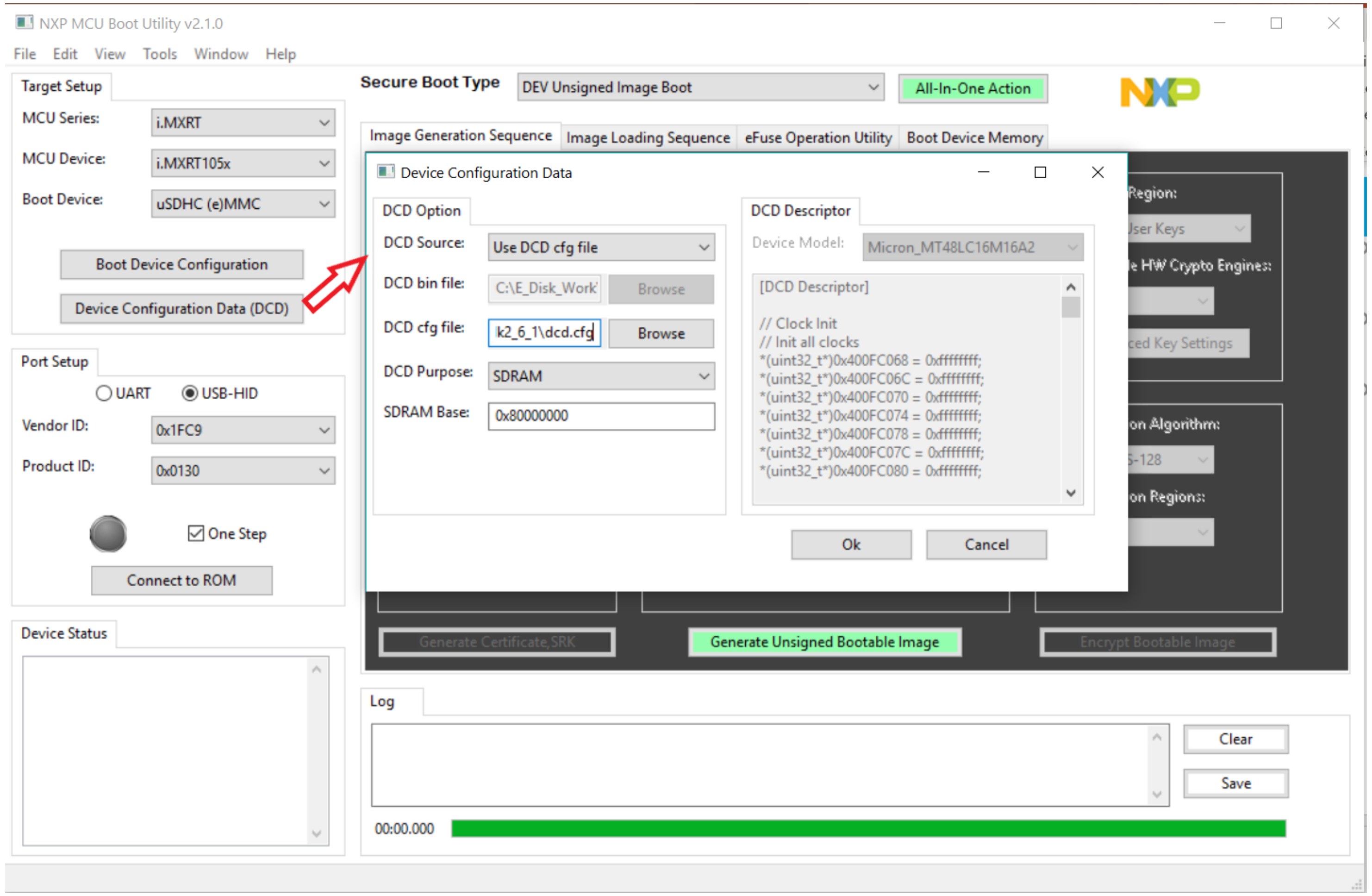
Bus Width:	4bit
Timing Interface:	HighSpeed
Partition Access:	User Area Normal
Enable Boot Config:	No
Boot Bus Width:	1bit-SDR, 4bit-DDR
Boot Mode:	SDR Non-HighSpeed
Enable Boot Partition:	No
Enable Boot Ack:	No
Reset Boot Bus Conditions:	Reset to 1bit-SDR

MMC Option1

Enable 1.8V:	No
Enable Power Cycle:	No
Power Polarity:	RST Low-Disable
Power Up Time:	5ms
Power Down Time:	20ms
Enable Perm Config:	0
Perm Boot Config Prot:	0
Driver Strength:	0

eMMC Boot Hands-On

6. 如果是使用eMMC --> SDRAM启动模式，则需要对DCD进行配置，建议使用.cfg文件来进行DCD的配置，该文件在文档最后下载链接里的示例工程根目录下，里面配置需要针对不同板载型号的SDRAM进行配置，默认配置为RT1050_EVK上的SDRAM；



Target Setup
MCU Series: i.MXRT
MCU Device: i.MXRT105x
Boot Device: uSDHC (e)MMC

Port Setup
○ UART ● USB-HID
Vendor ID: 0x1FC9
Product ID: 0x0130
☑ One Step

Secure Boot Type: DEV Unsigned Image Boot

Device Configuration Data

DCD Option
DCD Source: Use DCD cfg file
DCD bin file: C:\E_Disk_Work
DCD cfg file: k2_6_1\dcd.cfg
DCD Purpose: SDRAM
SDRAM Base: 0x80000000

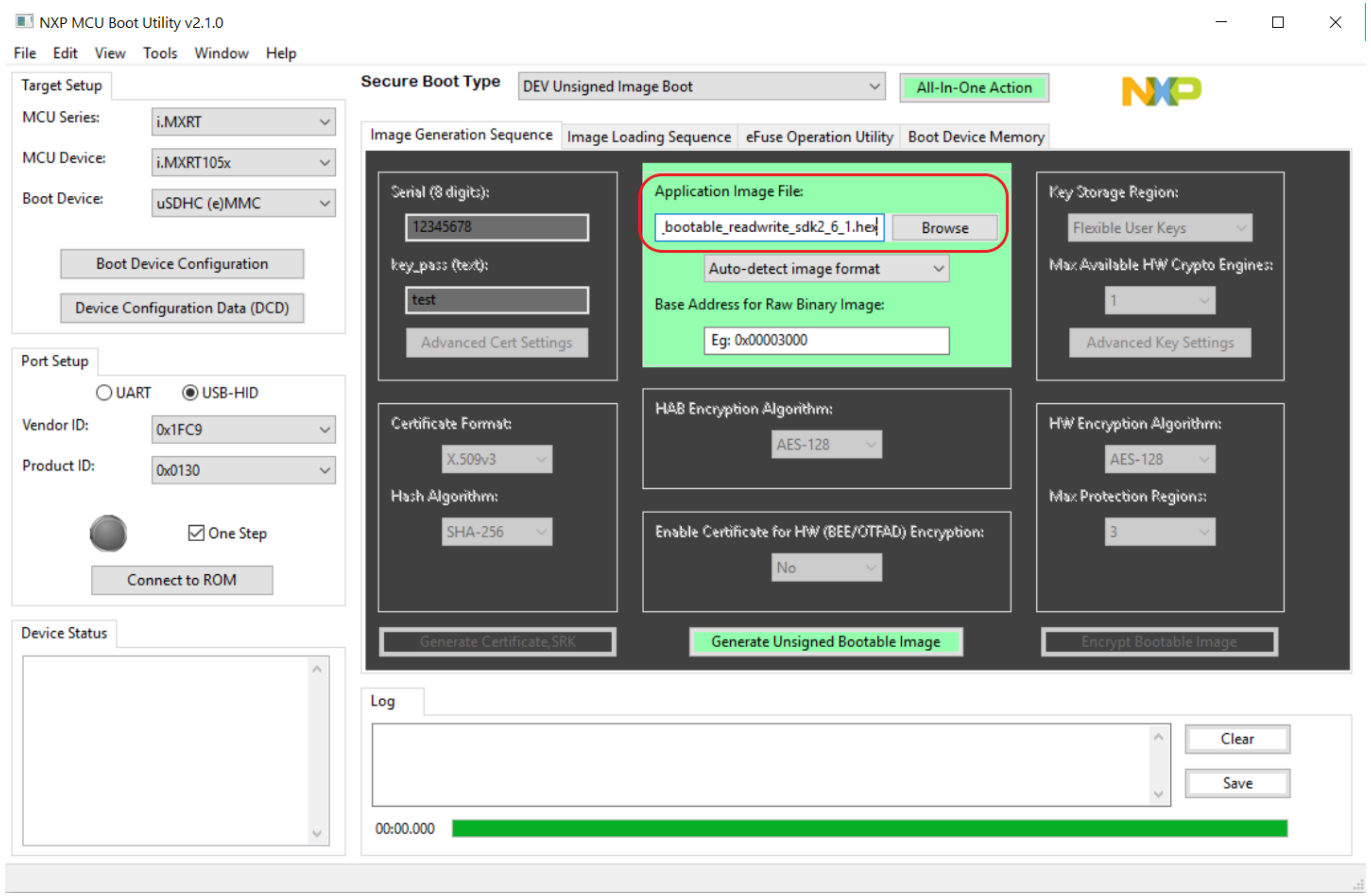
DCD Descriptor
Device Model: Micron_MT48LC16M16A2

```
[DCD Descriptor]

// Clock Init
// Init all clocks
*(uint32_t*)0x400FC068 = 0xffffffff;
*(uint32_t*)0x400FC06C = 0xffffffff;
*(uint32_t*)0x400FC070 = 0xffffffff;
*(uint32_t*)0x400FC074 = 0xffffffff;
*(uint32_t*)0x400FC078 = 0xffffffff;
*(uint32_t*)0x400FC07C = 0xffffffff;
*(uint32_t*)0x400FC080 = 0xffffffff;
```

00:00.000

eMMC Boot Hands-On

7. 使用第4点说明的链接文件或者分散加载文件修改后生成的hex、bin或者srec文件作为用户代码文件，然后让RT1052进入Serial Download模式，待识别到VID和PID之后点击左下角的Connect to ROM，可以看到读取的eMMC基本信息表明连接成功，最后点击上面的All-In-One Action即会先生成完整的Bootable Image文件并擦写和烧录文件到eMMC，下载成功后将RT1052启动模式拨回Internal Boot即可看到程序正常执行起来了。



Target Setup
MCU Series: i.MXRT
MCU Device: i.MXRT105x
Boot Device: uSDHC (e)MMC

Port Setup
UART [✓ USB-HID]
Vendor ID: 0x1FC9
Product ID: 0x0130
[✓ One Step]

Secure Boot Type: DEV Unsigned Image Boot

Serial (8 digits): 12345678
key_pass (text): test
Application Image File: bootable_readwrite_sdk2_6_1.hex
Auto-detect image format
Base Address for Raw Binary Image: Eg: 0x00003000
Key Storage Region: Flexible User Keys
Max Available HW Crypto Engines: 1
Certificate Format: X.509v3
Hash Algorithm: SHA-256
HAB Encryption Algorithm: AES-128
Enable Certificate for HW (BEE/OTFAD) Encryption: No
HW Encryption Algorithm: AES-128
Max Protection Regions: 3

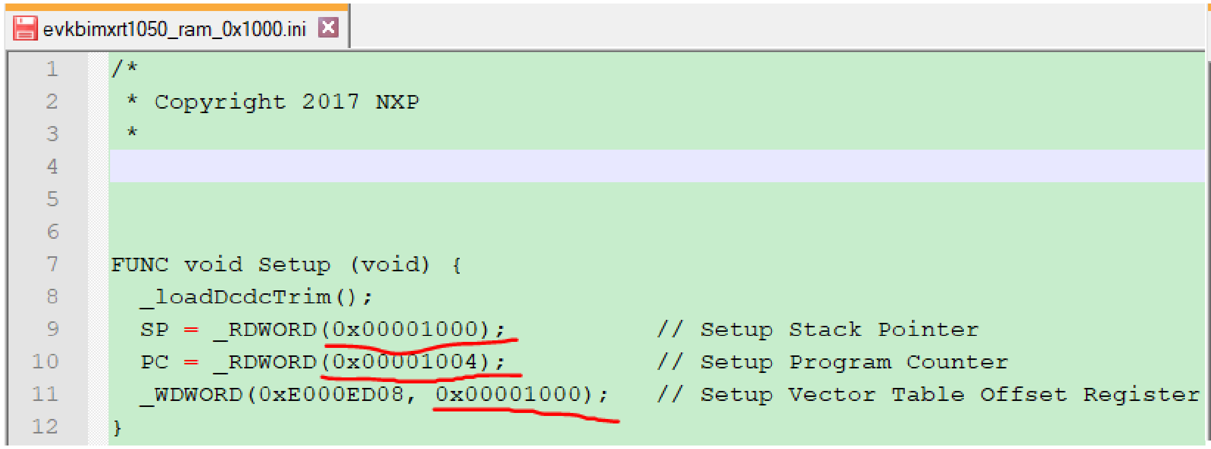
00:00.000

Code Debug on Non-XIP Mode

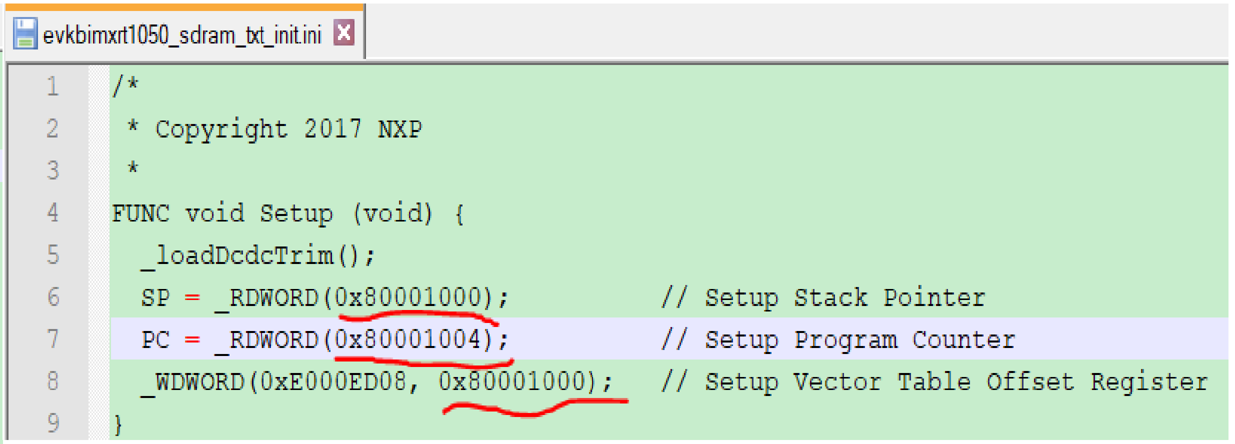
8. 在Non-XIP模式下，代码不是运行在外部存储设备上而是执行在片上或者片外RAM上，所以代码的调试则可以在IDE环境下直接通过仿真器下载到目标RAM地址上仿真（直接点Debug而不能再点Download），不过需要注意的是由于我们在链接文件或分散加载文件里给IVT等配置信息预留了4KB空间，那么IDE下的初始化脚本（Keil下为ini文件，IAR下为.mac文件）则也应如下修改好偏移地址才可以正常跳转仿真。

在仿真调试环境下，如果能确保初始化文件（ini或者mac文件）的配置与前面提到的dcd.cfg配置一致的话，那么无论是下载到eMMC里上电运行还是通过仿真器直接调试它们得到的最终结果理论上是一致的。

Note：
关于Non-XIP模式下的仿真调试，我的建议和经验是在调试时将RT1052 Boot Mode拨码拨到Serial Download模式，每次仿真调试时按一次复位或者重新上下电让CPU并在ROM等待，然后通过仿真器将代码下载到指定的RAM空间后强制跳转到代码的第一条入口那里（Reset_handler）,这种操作可以让CPU复位完整且完全可控。


evkbimxrt1050_ram_0x1000.ini

```
/*
 * Copyright 2017 NXP
 *



FUNC void Setup (void) {
  _loadDcdcTrim();
  SP = _RDWORD(0x00001000);          // Setup Stack Pointer
  PC = _RDWORD(0x00001004);          // Setup Program Counter
  _WDWORD(0xE000ED08, 0x00001000);   // Setup Vector Table Offset Register
}
```


evkbimxrt1050_sdram_txt_init.ini

```
/*
 * Copyright 2017 NXP
 *
FUNC void Setup (void) {
  _loadDcdcTrim();
  SP = _RDWORD(0x80001000);          // Setup Stack Pointer
  PC = _RDWORD(0x80001004);          // Setup Program Counter
  _WDWORD(0xE000ED08, 0x80001000);   // Setup Vector Table Offset Register
}
```

eMMC Read/Write on I.MXRT1052

eMMC Read/Write on RT1052

1. 对i.MXRT系列的应用来说，eMMC的启动只是很小一部分，更多的应用场景是涉及到对eMMC的数据读和写，所以无论是裸数据读写还是通过嵌入文件系统来管理读写，都是需要打通最底层对eMMC的读、擦和写，换句话说就是向上层提供如下图所示的Init，Read, Write和Erase的API并设计好传参。

对eMMC来讲，其读写操作的最小单位与NandFlash一样为Block，每个Block的size默认为512 Bytes，而擦除则是以Group为单位，而每个Group的size则可以通过CMD命令读取eMMC的CSD寄存器来判断。

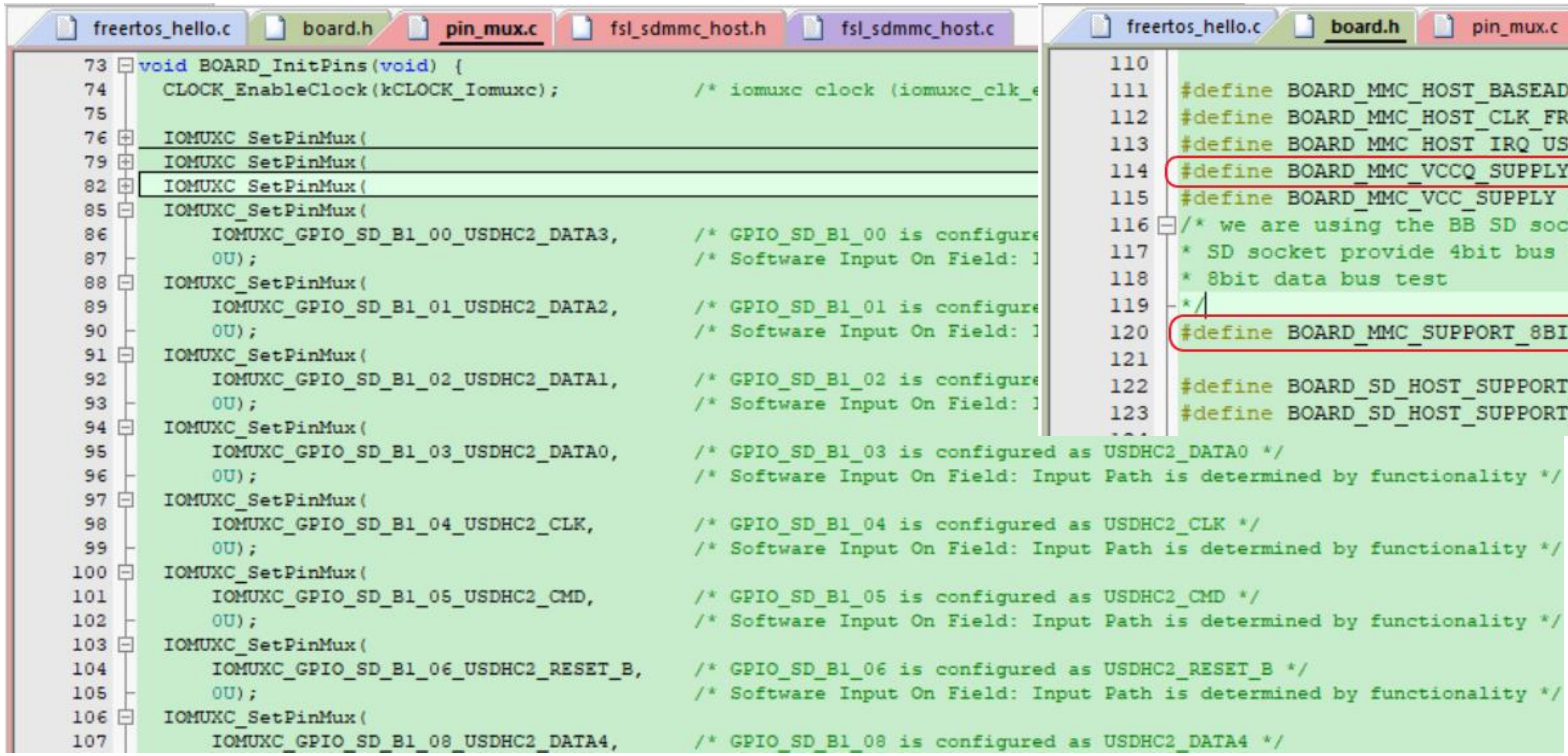
- status_t MMC_Init(mmc_card_t *card);
- status_t MMC_CardInit(mmc_card_t *card);
- status_t MMC_ReadBlocks(mmc_card_t *card, uint8_t *buffer, uint32_t startBlock, uint32_t blockCount);
- status_t MMC_WriteBlocks(mmc_card_t *card, const uint8_t *buffer, uint32_t startBlock, uint32_t blockCount);
- status_t MMC_EraseGroups(mmc_card_t *card, uint32_t startGroup, uint32_t endGroup);

Note：
如果你的代码不超过4MB，则我的建议是你可以把代码放到eMMMC的BootArea分区，然后需要频繁读写的用户的数据则放到User Data Area分区，这样可以做到代码和数据安全隔离，频繁读写数据的时候不会影响到用户代码，保证代码完整。当然，如果用户代码超过4MB，其实也可以在BootArea区只放一个小的Bootloader，这样可以确保每次上电该Bootloader是先允许的，然后通过它把存放在UserData Area分区主要的用户代码手动load到RAM执行，这样即使UserData Area分区的主程序被破坏了，Bootloader也还存在。

eMMC Read/Write on RT1052

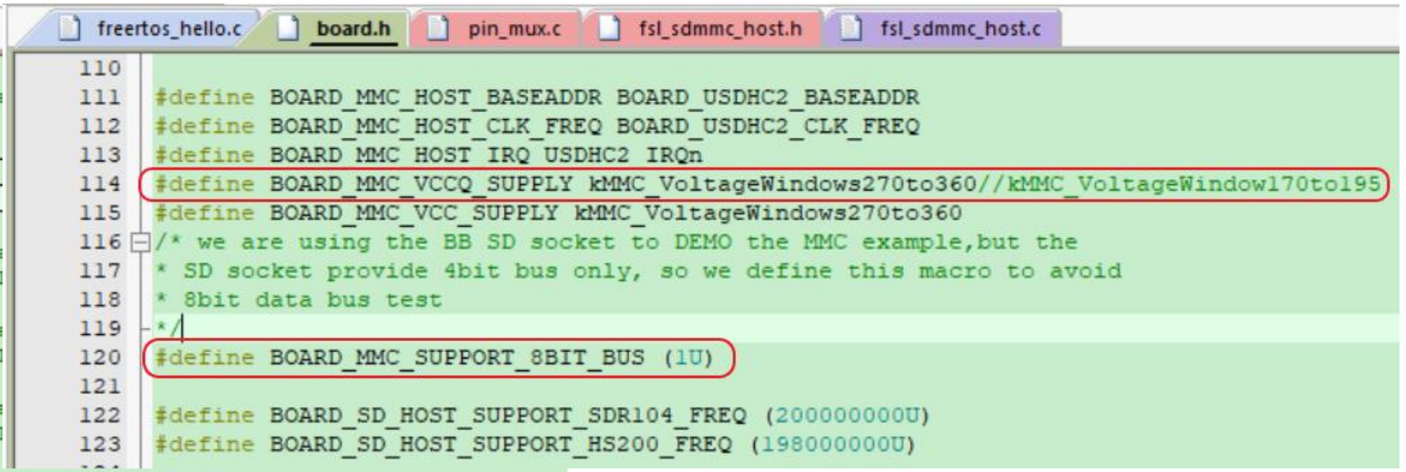
2. 本文档最后链接提供了基于RT1052的完整eMMC读写驱动示例工程，该例程基于SDK2.6.1的middleware sdmmc驱动修改而来，修补了针对eMMC的几个处理不妥之处，并做了板级的驱动移植，其中有些地方需要着重注意，列举如下，供参考：

（1）首先修改板级支持文件，pinmux.c和board.h，将其修改成板子eMMC所连接的uSDHC接口，并且配置好eMMC的VCC和VCCQ电压，VCCQ电压为3.3v时最高速度只能支持到DDR52速度模式；



freertos_hello.c | board.h | pin_mux.c | fsl_sdmmc_host.h | fsl_sdmmc_host.c

```
void BOARD_InitPins(void) {
  CLOCK_EnableClock(kCLOCK_Iomuxc);           /* iomuxc clock (iomuxc_clk_e

  IOMUXC_SetPinMux(
  IOMUXC_SetPinMux(
  IOMUXC_SetPinMux(
  IOMUXC_SetPinMux(
      IOMUXC_GPIO_SD_B1_00_USDHC2_DATA3,      /* GPIO_SD_B1_00 is configure
      0U);                                    /* Software Input On Field: I
  IOMUXC_SetPinMux(
      IOMUXC_GPIO_SD_B1_01_USDHC2_DATA2,      /* GPIO_SD_B1_01 is configure
      0U);                                    /* Software Input On Field: I
  IOMUXC_SetPinMux(
      IOMUXC_GPIO_SD_B1_02_USDHC2_DATA1,      /* GPIO_SD_B1_02 is configure
      0U);                                    /* Software Input On Field: I
  IOMUXC_SetPinMux(
      IOMUXC_GPIO_SD_B1_03_USDHC2_DATA0,      /* GPIO_SD_B1_03 is configured as USDHC2_DATA0 */
      0U);                                    /* Software Input On Field: Input Path is determined by functionality */
  IOMUXC_SetPinMux(
      IOMUXC_GPIO_SD_B1_04_USDHC2_CLK,        /* GPIO_SD_B1_04 is configured as USDHC2_CLK */
      0U);                                    /* Software Input On Field: Input Path is determined by functionality */
  IOMUXC_SetPinMux(
      IOMUXC_GPIO_SD_B1_05_USDHC2_CMD,        /* GPIO_SD_B1_05 is configured as USDHC2_CMD */
      0U);                                    /* Software Input On Field: Input Path is determined by functionality */
  IOMUXC_SetPinMux(
      IOMUXC_GPIO_SD_B1_06_USDHC2_RESET_B,    /* GPIO_SD_B1_06 is configured as USDHC2_RESET_B */
      0U);                                    /* Software Input On Field: Input Path is determined by functionality */
  IOMUXC_SetPinMux(
      IOMUXC_GPIO_SD_B1_08_USDHC2_DATA4,      /* GPIO_SD_B1_08 is configured as USDHC2_DATA4 */
```

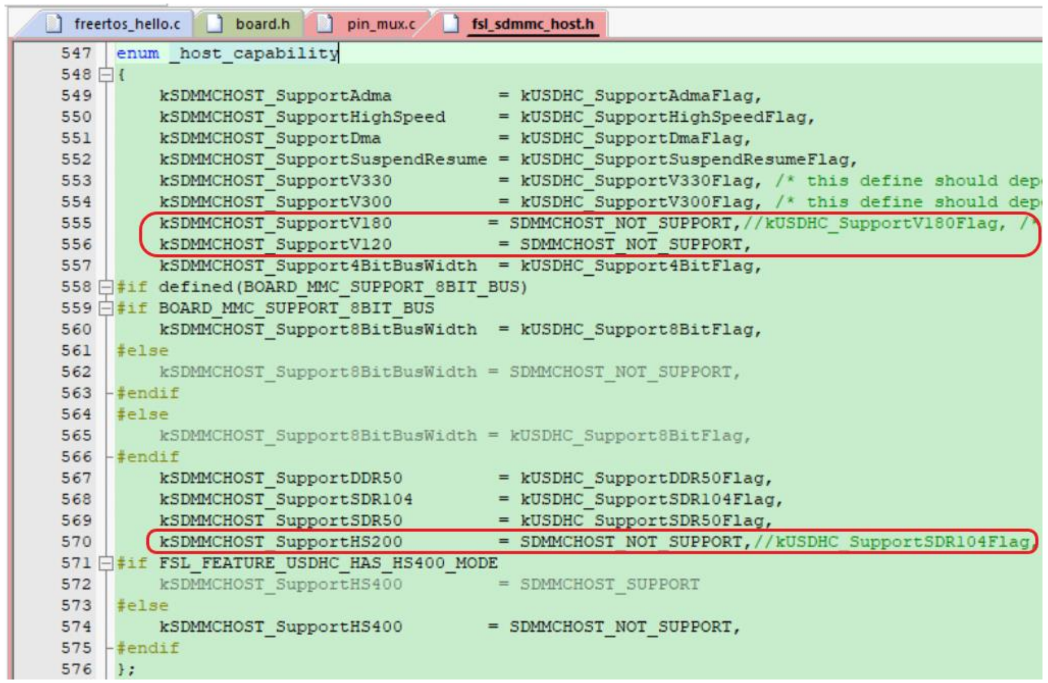


freertos_hello.c | board.h | pin_mux.c | fsl_sdmmc_host.h | fsl_sdmmc_host.c

```

#define BOARD_MMC_HOST_BASEADDR BOARD_USDHC2_BASEADDR
#define BOARD_MMC_HOST_CLK_FREQ BOARD_USDHC2_CLK_FREQ
#define BOARD_MMC_HOST_IRQ USDHC2_IRQn
#define BOARD_MMC_VCCQ_SUPPLY kMMC_VoltageWindows270to360//kMMC_VoltageWindow170to195
#define BOARD_MMC_VCC_SUPPLY kMMC_VoltageWindows270to360
/* we are using the BB SD socket to DEMO the MMC example,but the
 * SD socket provide 4bit bus only, so we define this macro to avoid
 * 8bit data bus test
 */
#define BOARD_MMC_SUPPORT_8BIT_BUS (1U)

#define BOARD_SD_HOST_SUPPORT_SDR104_FREQ (200000000U)
#define BOARD_SD_HOST_SUPPORT_HS200_FREQ (198000000U)
```

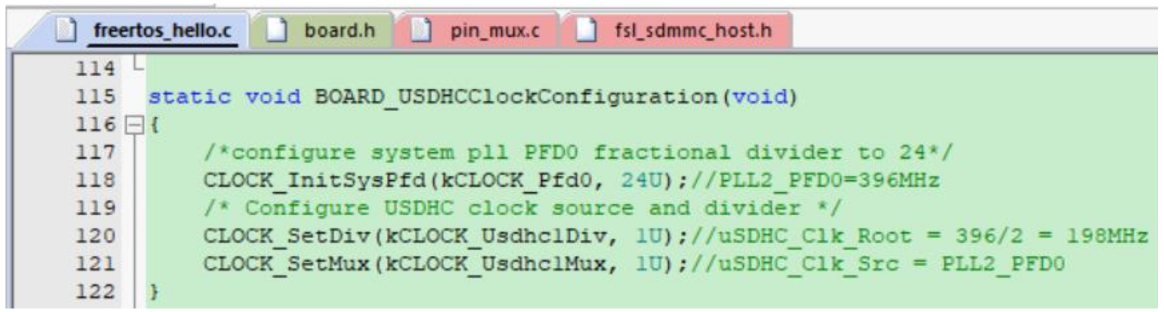
eMMC Read/Write on RT1052

（2）如果VCCQ使用3.3v供电，则需要修改fsl_sdmmc_host.h文件，将V180，V120电压模式和HS200速度模式手动修改成SDMMC_NOT_SUPPORT，避免sdmmc里面状态机切换成HS200；


freertos_hello.c | board.h | pin_mux.c | fsl_sdmmc_host.h

```
enum _host_capability
{
    kSDMMCHOST_SupportAdma           = kUSDHC_SupportAdmaFlag,
    kSDMMCHOST_SupportHighSpeed      = kUSDHC_SupportHighSpeedFlag,
    kSDMMCHOST_SupportDma            = kUSDHC_SupportDmaFlag,
    kSDMMCHOST_SupportSuspendResume = kUSDHC_SupportSuspendResumeFlag,
    kSDMMCHOST_SupportV330           = kUSDHC_SupportV330Flag, /* this define should dep
    kSDMMCHOST_SupportV300           = kUSDHC_SupportV300Flag, /* this define should dep
    kSDMMCHOST_SupportV180          = SDMMCHOST_NOT_SUPPORT,//kUSDHC_SupportV180Flag, /*
    kSDMMCHOST_SupportV120           = SDMMCHOST_NOT_SUPPORT,
    kSDMMCHOST_Support4BitBusWidth  = kUSDHC_Support4BitFlag,
#if defined(BOARD_MMC_SUPPORT_8BIT_BUS)
#if BOARD_MMC_SUPPORT_8BIT_BUS
    kSDMMCHOST_Support8BitBusWidth  = kUSDHC_Support8BitFlag,
#else
    kSDMMCHOST_Support8BitBusWidth = SDMMCHOST_NOT_SUPPORT,
#endif
#else
    kSDMMCHOST_Support8BitBusWidth = kUSDHC_Support8BitFlag,
#endif
    kSDMMCHOST_SupportDDR50          = kUSDHC_SupportDDR50Flag,
    kSDMMCHOST_SupportSDR104         = kUSDHC_SupportSDR104Flag,
    kSDMMCHOST_SupportSDR50          = kUSDHC_SupportSDR50Flag,
    kSDMMCHOST_SupportHS200          = SDMMCHOST_NOT_SUPPORT,//kUSDHC_SupportSDR104Flag,
#if FSL_FEATURE_USDHC_HAS_HS400_MODE
    kSDMMCHOST_SupportHS400          = SDMMCHOST_SUPPORT
#else
    kSDMMCHOST_SupportHS400         = SDMMCHOST_NOT_SUPPORT,
#endif
};
```

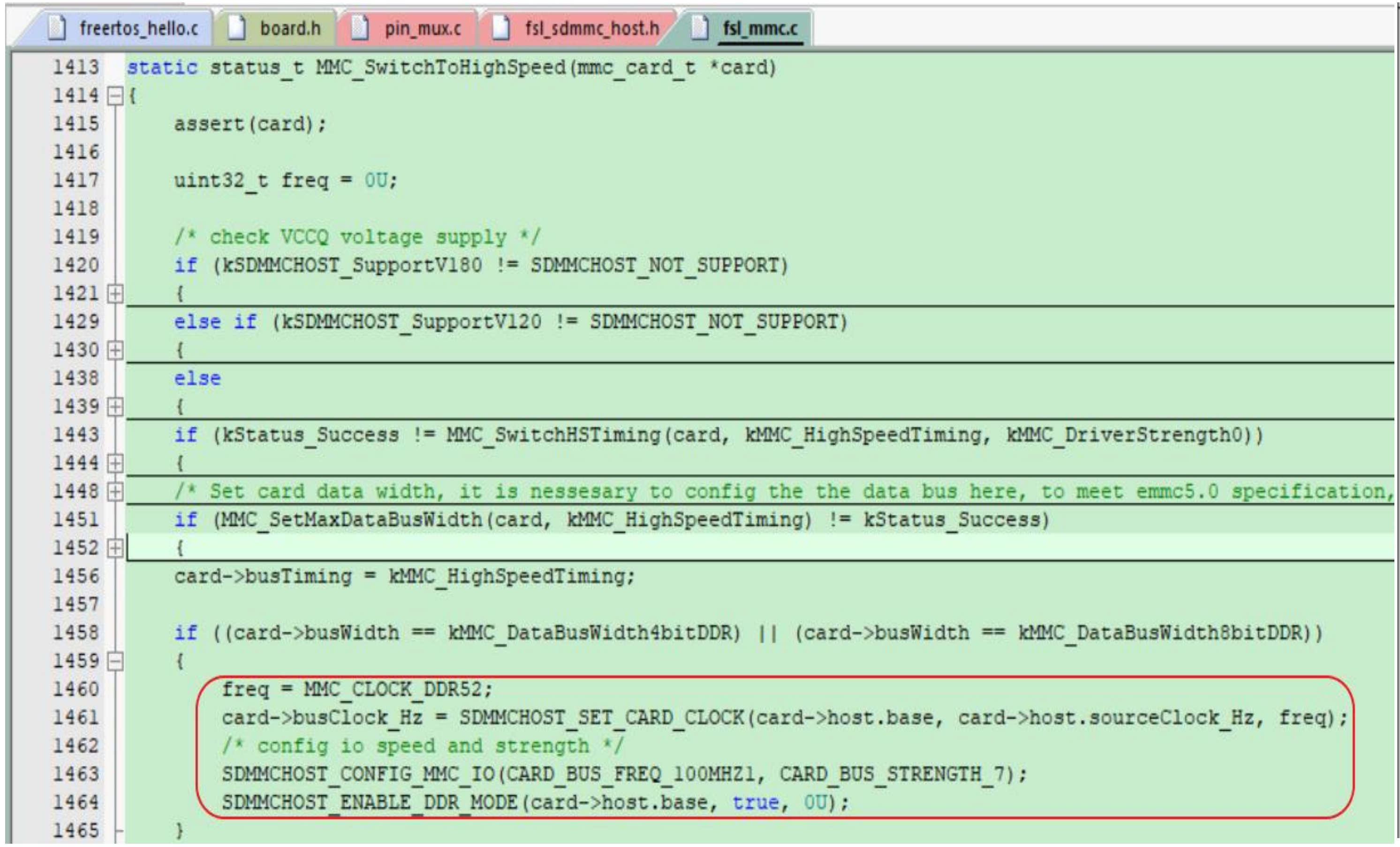
（3）uSDHC时钟部分，前面有提到过，建议把uSDHC的时钟源选择PLL2_PFD0并且2分配为198MHz（uSDHC模块最高工作频率为208MHz），避免跟SDRAM的时钟源选择同一个；


freertos_hello.c | board.h | pin_mux.c | fsl_sdmmc_host.h

```

static void BOARD_USDHCClockConfiguration(void)
{
    /*configure system pll PFD0 fractional divider to 24*/
    CLOCK_InitSysPfd(kCLOCK_Pfd0, 24U);//PLL2_PFD0=396MHz
    /* Configure USDHC clock source and divider */
    CLOCK_SetDiv(kCLOCK_Usdhc1Div, 1U);//uSDHC_Clk_Root = 396/2 = 198MHz
    CLOCK_SetMux(kCLOCK_Usdhc1Mux, 1U);//uSDHC_Clk_Src = PLL2_PFD0
}
```

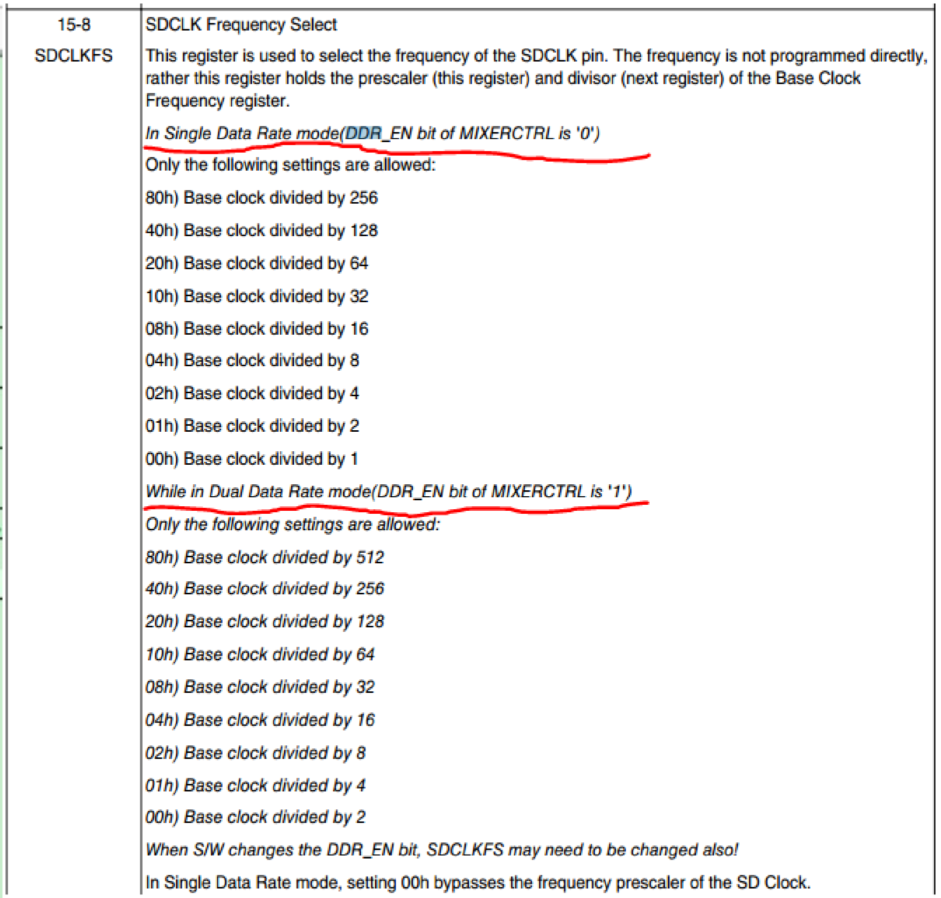
eMMC Read/Write on RT1052

（4）sdk驱动fsl_mmc.c文件在切换高速模式时配置顺序需要修改如下，即先配置52MHz频率之后，再使能DDR模式，因为如右图所示同样的SCLKFS赋值在DDR使能之后会自动把频率再次2分配，而使能DDR API函数内部会把这个多出来的2分配补偿回来；



freertos_hello.c | board.h | pin_mux.c | fsl_sdmmc_host.h | fsl_mmc.c

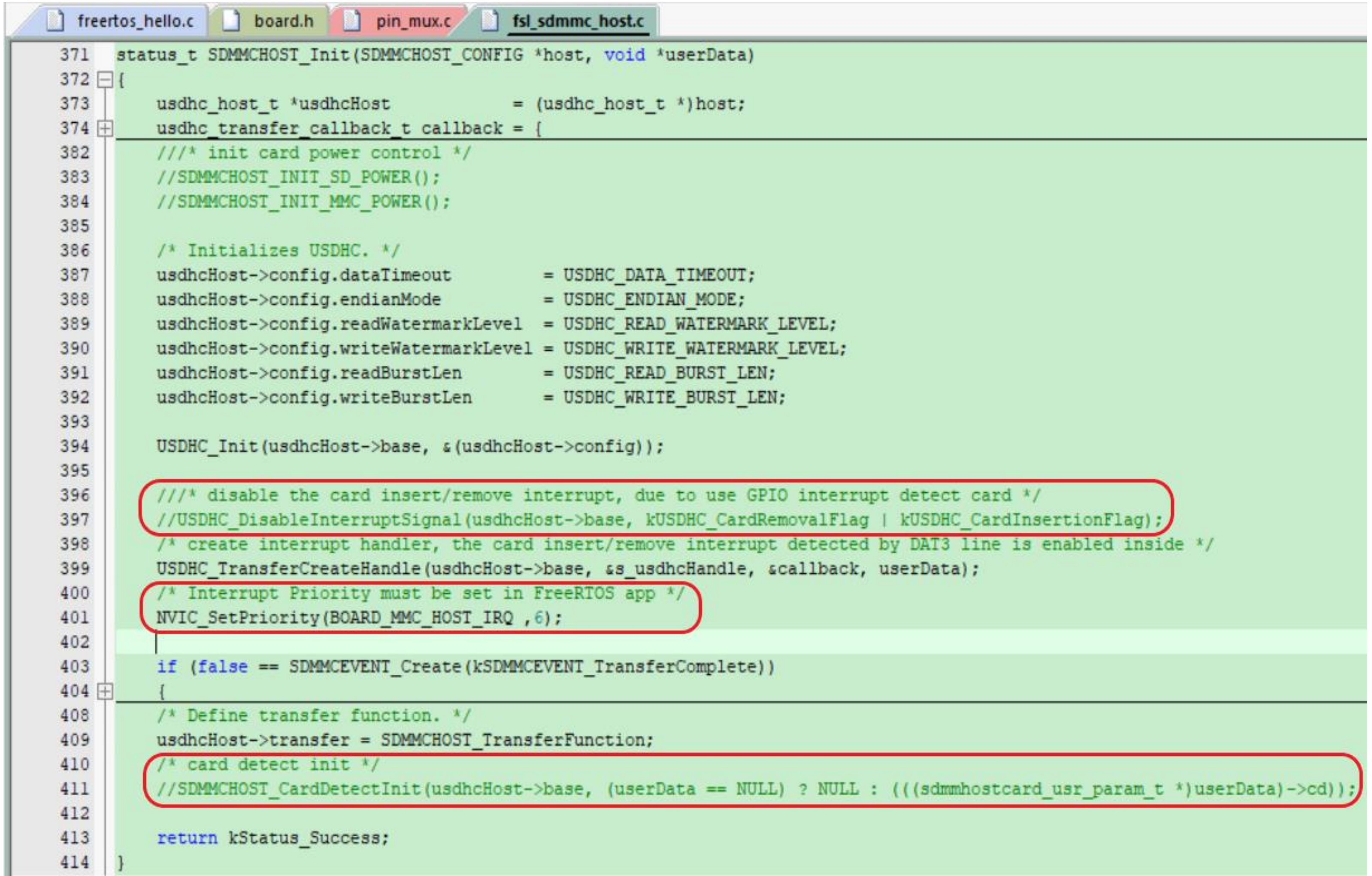
```c
static status_t MMC_SwitchToHighSpeed(mmc_card_t *card)
{
    assert(card);

    uint32_t freq = 0U;

    /* check VCCQ voltage supply */
    if (kSDMMCHOST_SupportV180 != SDMMCHOST_NOT_SUPPORT)
    {
    else if (kSDMMCHOST_SupportV120 != SDMMCHOST_NOT_SUPPORT)
    {
    else
    {
    if (kStatus_Success != MMC_SwitchHSTiming(card, kMMC_HighSpeedTiming, kMMC_DriverStrength0))
    {
    /* Set card data width, it is nessesary to config the the data bus here, to meet emmc5.0 specification,
    if (MMC_SetMaxDataBusWidth(card, kMMC_HighSpeedTiming) != kStatus_Success)
    {
    card->busTiming = kMMC_HighSpeedTiming;

    if ((card->busWidth == kMMC_DataBusWidth4bitDDR) || (card->busWidth == kMMC_DataBusWidth8bitDDR))
    {
        freq = MMC_CLOCK_DDR52;
        card->busClock_Hz = SDMMCHOST_SET_CARD_CLOCK(card->host.base, card->host.sourceClock_Hz, freq);
        /* config io speed and strength */
        SDMMCHOST_CONFIG_MMC_IO(CARD_BUS_FREQ_100MHZ1, CARD_BUS_STRENGTH_7);
        SDMMCHOST_ENABLE_DDR_MODE(card->host.base, true, 0U);
    }
```



15-8 SDCLKFS	SDCLK Frequency Select This register is used to select the frequency of the SDCLK pin. The frequency is not programmed directly, rather this register holds the prescaler (this register) and divisor (next register) of the Base Clock Frequency register. *In Single Data Rate mode(DDR_EN bit of MIXERCTRL is '0')* Only the following settings are allowed: 80h) Base clock divided by 256 40h) Base clock divided by 128 20h) Base clock divided by 64 10h) Base clock divided by 32 08h) Base clock divided by 16 04h) Base clock divided by 8 02h) Base clock divided by 4 01h) Base clock divided by 2 00h) Base clock divided by 1 *While in Dual Data Rate mode(DDR_EN bit of MIXERCTRL is '1')* *Only the following settings are allowed:* *80h) Base clock divided by 512* *40h) Base clock divided by 256* *20h) Base clock divided by 128* *10h) Base clock divided by 64* *08h) Base clock divided by 32* *04h) Base clock divided by 16* *02h) Base clock divided by 8* *01h) Base clock divided by 4* *00h) Base clock divided by 2* *When S/W changes the DDR_EN bit, SDCLKFS may need to be changed also!* In Single Data Rate mode, setting 00h bypasses the frequency prescaler of the SD Clock.

eMMC Read/Write on RT1052

（5）原SDMMCHOST_Init是为SD卡和MMC卡考虑的，所以加入了卡检测机制，但是对eMMC来讲它本身就是embeded在电路板上的，一直存在于板子上，所以可以把卡检测机制去掉。此外，由于本示例是基于FreeRTOS做的，在基于FreeRTOS开发的应用里面里面所有中断都需要定义优先级才可以，所以在该初始化函数里加入uSDHC中断的优先级配置，且该优先级要小于FreeRTOS可管理的最大优先级；

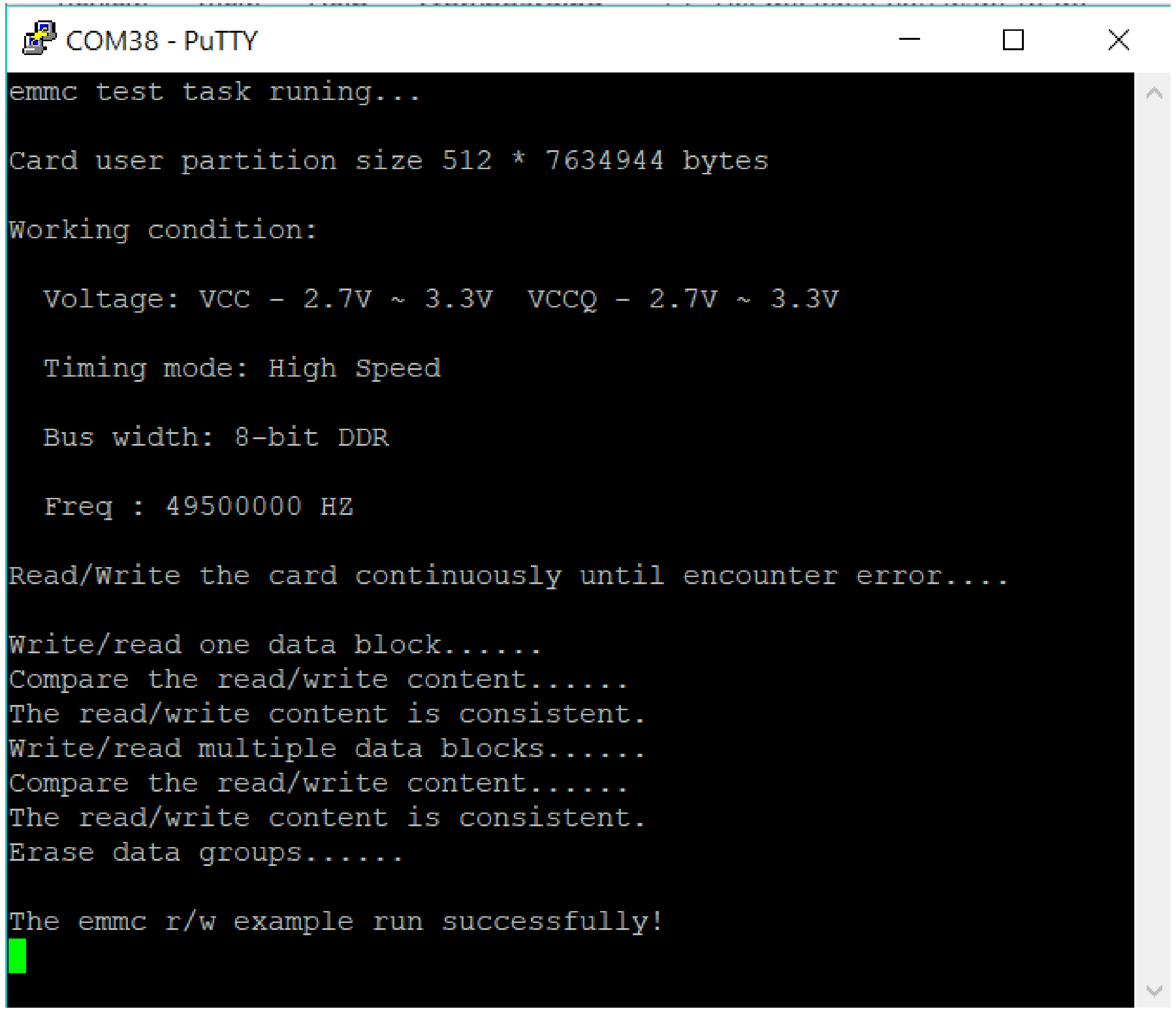


freertos_hello.c | board.h | pin_mux.c | fsl_sdmmc_host.c

```c
status_t SDMMCHOST_Init(SDMMCHOST_CONFIG *host, void *userData)
{
    usdhc_host_t *usdhcHost            = (usdhc_host_t *)host;
    usdhc_transfer_callback_t callback = {
    ///* init card power control */
    //SDMMCHOST_INIT_SD_POWER();
    //SDMMCHOST_INIT_MMC_POWER();

    /* Initializes USDHC. */
    usdhcHost->config.dataTimeout         = USDHC_DATA_TIMEOUT;
    usdhcHost->config.endianMode          = USDHC_ENDIAN_MODE;
    usdhcHost->config.readWatermarkLevel  = USDHC_READ_WATERMARK_LEVEL;
    usdhcHost->config.writeWatermarkLevel = USDHC_WRITE_WATERMARK_LEVEL;
    usdhcHost->config.readBurstLen        = USDHC_READ_BURST_LEN;
    usdhcHost->config.writeBurstLen       = USDHC_WRITE_BURST_LEN;

    USDHC_Init(usdhcHost->base, &(usdhcHost->config));

    ///* disable the card insert/remove interrupt, due to use GPIO interrupt detect card */
    //USDHC_DisableInterruptSignal(usdhcHost->base, kUSDHC_CardRemovalFlag | kUSDHC_CardInsertionFlag);
    /* create interrupt handler, the card insert/remove interrupt detected by DAT3 line is enabled inside */
    USDHC_TransferCreateHandle(usdhcHost->base, &s_usdhcHandle, &callback, userData);
    /* Interrupt Priority must be set in FreeRTOS app */
    NVIC_SetPriority(BOARD_MMC_HOST_IRQ ,6);

    if (false == SDMMCEVENT_Create(kSDMMCEVENT_TransferComplete))
    {
    /* Define transfer function. */
    usdhcHost->transfer = SDMMCHOST_TransferFunction;
    /* card detect init */
    //SDMMCHOST_CardDetectInit(usdhcHost->base, (userData == NULL) ? NULL : (((sdmmhostcard_usr_param_t *)userData)->cd));

    return kStatus_Success;
}
```

eMMC Read/Write on RT1052

3. 将示例代码根据用户不同的板子移植好之后，下载到板子上，上电运行即可得到如下log输出；



COM38 - PuTTY

```
emmc test task runing...

Card user partition size 512 * 7634944 bytes

Working condition:

  Voltage: VCC - 2.7V ~ 3.3V  VCCQ - 2.7V ~ 3.3V

  Timing mode: High Speed

  Bus width: 8-bit DDR

  Freq : 49500000 HZ

Read/Write the card continuously until encounter error....

Write/read one data block......
Compare the read/write content......
The read/write content is consistent.
Write/read multiple data blocks......
Compare the read/write content......
The read/write content is consistent.
Erase data groups......

The emmc r/w example run successfully!
```

Reference & FAQ

Reference

本文档提到的工具和示例代码可以从如下github链接下载:

https://github.com/jicheng0622/DFAE_Sharing_Git

参考文献:

1. https://linux.codingbelief.com/zh/storage/flash_memory/emmc/
2. https://github.com/JayHeng/NXP-MCUBootUtility
3. https://www.cnblogs.com/smartjourneys/p/6652388.html
4. IMXRT1050RM.pdf
5. i.MX MCU Manufacting User's Guide.pdf
6. Micro eMMC MTFC8GAKAJCN.pdf
7. SPF-29567_A_ValidationBoard.pdf

寄语

《春风》

春风如贵客，一到便繁华

来扫千山雪，归留万国花

SECURE CONNECTIONS
FOR A SMARTER WORLD