

# EMMC

- 13. EMMC

# 13. EMMC

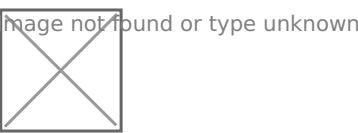
[https://linux.codingbelief.com/zh/storage/flash\\_memory/emmc/](https://linux.codingbelief.com/zh/storage/flash_memory/emmc/)

00\_UserManual\ \EMMC \JESD84-B50-1eMMCStandard.pdf

## 1.1 EMMC

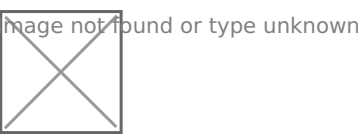
### 1.1.1 EMMC

eMMC (Embedded Multi Media Card   MMC eMMC



### 1.1.2 EMMC

,CPU eMMC   CLK CMD DATA[0-7] RST



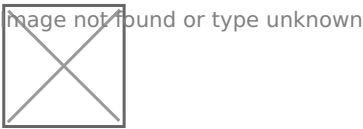
### 1.1.3 eMMC

eMMC5.0   5   MMC   SDR   DDR   4.5



1.1.5.2 (2)

CMD CRC 48



					CRC	
	47	46	[45:40]	[39:8]	[7:1]	0
	"0"	"1"	x	x	x	"1"

0 1 eMMC 6 0~63 CRC7

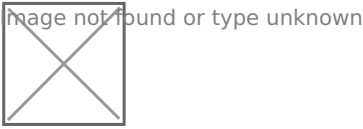
1.1.5.3 (3)

eMMC 56 12 class class class CSD CSD[95:84]

class 0	basic	
class 1	Obsolete	
class 2	block read	
class 3	obsolete	
class 4	block write	
class 5	erase	
class 6	write protection	
class 7	Lock device	
class 8	Application-specific	
class 9	I/O mode	
class 10	security protocols	
class 11	reserved	

1.1.6 Response

eMMC CMD CRC

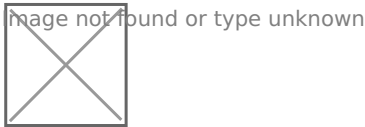
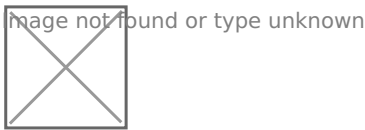


eMMC      6                      R1 R1b R2 R3 R4 R5

- R1 R1b

R1      48bit    [45:40]                      [39:8]                      R1b      R1                      Busy

					CRC	
	47	46	[45:40]	[39:8]	[7:1]	0
	1	1	6	32	7	1
	“0”	“0”	x	x	CRC	“1”



- R2

R2                      CID CSD                      CID                      CMD2 CMD10 CSD                      CMD9

					CRC	
	135	134	[133:128]	[127:8]	[7:1]	0
	1	1	6	120	7	1
	“0”	“0”	“111111”	x	CRC	“1”

- R3

R3                      ORC                      CMD1                      R3

				ORC		
	47	46	[45:40]	[39:8]	[7:1]	0
	1	1	6	32	7	1
	“0”	“0”	“111111”	x	“1111111”	“1”

- R4

R4**CMD39R4**

			39		CRC	
	47	46	[45:40]	[39:8]	[7:1]	0
	1	1	6	32	7	1
	“0”	“0”	“100111”	X	“1111111”	“1”

[39:8]				
	16	1	7	8

- R5

R5

			39		CRC	
	47	46	[45:40]	[39:8]	[7:1]	0
	1	1	6	32	7	1
	“0”	“0”	“100111”	X	“1111111”	“1”

[39:8]		
	16	16

# 1.1.7 eMMC

EMMC 5

Boot mode	eMMC      0xF0F0F0F0    CMD0    eMMC
Card identification	eMMC                      CMD3
Interrupt mode	
Data transfer mode	eMMC      RCA
Inactive mode	eMMC                      CMD1    eMMC

eMMC

--	--	--

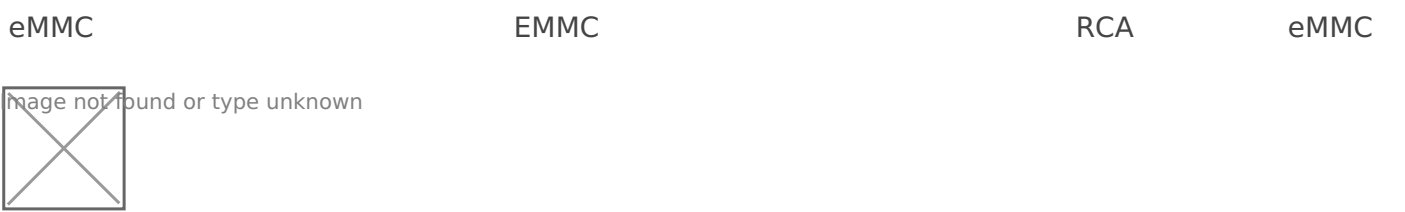
Inactive State		
Pre-Idle State		
Pre-Boot State		
Idle State		
Ready State		
Identification State		
Stand-by State		
Sleep State		
Transfer State		
Bus-Test State		
Sending-data State		
Programming State		
Disconnect State		
Boot State		
Wait-IRQ State		

# 1.1.8 eMMC

6

	Byte	
CID	16	
RCA	2	
DSR	2	
CSD	16	
OCR	4	
EXT_CSD	512	

# 1.1.9 eMMC



eMMC

Idle State

OCR Busy

CMD1 CMD58

CMD1

eMMC

OCR

BUSY

BUSY 1

e

Ready State

CMD2

CID

eMMC

Identification State

CMD3

eMMC

R

## 1.1.10

EMMC

Image not found or type unknown



## 1.2 IMX6ULL EMMC

### 1.2.1 IMX6ULL USDHC

IMX6ULL 2 USDHC

1) MMC 4.2/4.3/4.4/4.41/4.5

2) 208MHz

3) 1 /4 /8 SD SDIO MMC

4) SDR 4 SDIO 832Mbps

5) DDR 4 SDXC 400Mbps

6) SDXC 4 SDXC 832Mbps

7) SDXC 4 400Mbps / 1~4096

Chapter 58 Ultra Secured Digital Host Controller (uSDHC)

### 1.2.2 IMX6ULL USDHC

IMX6ULL USDHC 29

USDHC

USDHC



image not found or type unknown



image not found or type unknown



image not found or type unknown



### 1.2.2.1 1 uSDHC2\_BLK\_ATT

uSDHC2_BLK_ATT	BLKSIZE[12:0]	4096	0	BLKCNT
----------------	---------------	------	---	--------

image not found or type unknown



### 1.2.2.2 2 uSDHC2\_CMD\_ARG

uSDHC2\_CMD\_ARG

image not found or type unknown



### 1.2.2.3 3 uSDHC2\_CMD\_XFR\_TYP

uSDHC2\_CMD\_XFR\_TYP

image not found or type unknown



### 1.2.2.4 4 uSDHC2\_CMD\_RSP0-3

uSDHC2_CMD_RSP0-3	32	eMMC
-------------------	----	------

)

### 1.2.2.5 5 uSDHC2\_DATA\_BUFF\_ACC\_PORT

```

```

uSDHC2\_PRES\_SETATE USDHC USDHC CIHB CDIHB SDSTB BR

uSDHC2\_PROT\_CTRL

uSDHC2\_SYS\_CTRL

uSDHC2\_INT\_STATUS



```
1  uSDHC2_INT_STATUS
```

)



WR	WML	RD	WML	1
----	-----	----	-----	---

### 1.2.2.12 12 uSDHC2 MIX CTRL

uSDHC2\_MIX\_CTRL

)

# 1.3 EMMC

## 1.3.1 eMMC

eMMC	IMX6ULL	USDHC	EMMC
1	USDHC		
eMMC	IMX6-NAND	ALE nRE nWE DATA0-7	IMX6UUL USDHC2
2	USDHC2		

### 1.3.1.1 1 USDHC2

Chapter4	External Signals and Pin Multiplexing	USDHC2	USDHC2
image-20220111	194240136		

Image not found or type unknown

- GPIO4\_IO10 USDHC2\_RESET\_B
- )
- )
- IOMUXC\_SW\_MUX\_CTL\_PAD\_NAND\_ALE[MUX\_MODE] 0x01
- GPIO4\_IO00 USDHC2\_CLK
- )
- )
- IOMUXC\_SW\_MUX\_CTL\_PAD\_NAND\_RE\_B[MUX\_MODE] 0x01

- GPIO4\_IO01 USDHC2\_CMD





IOMUXC\_SW\_MUX\_CTL\_PAD\_NAND\_WE\_B[MUX\_MODE] 0x01

- USDHC2\_DATA0-7















DATA0-

7 ALT1 IOMUXC\_SW\_MUX\_CTL\_PAD\_NAND\_DATA0 IOMUXC\_SW\_MUX\_CTL\_PAD\_NAND\_DATA1 IOMUXC\_SW\_MUX\_CTL\_PAD\_NAND\_DATA2 IOMUXC\_SW\_MUX\_CTL\_PAD\_NAND\_DATA3 IOMUXC\_SW\_MUX\_CTL\_PAD\_NAND\_DATA4 IOMUXC\_SW\_MUX\_CTL\_PAD\_NAND\_DATA5 IOMUXC\_SW\_MUX\_CTL\_PAD\_NAND\_DATA6 IOMUXC\_SW\_MUX\_CTL\_PAD\_NAND\_DATA7 0x01

```

61 //
62 #IOMUXC_SW_MUX_CTL_PAD_NAND_RE_B = (volatile unsigned int *) (0x20E0178);
63 #IOMUXC_SW_MUX_CTL_PAD_NAND_WE_B = (volatile unsigned int *) (0x20E017C);
64 #IOMUXC_SW_MUX_CTL_PAD_NAND_DATA0 = (volatile unsigned int *) (0x20E0180);
65 #IOMUXC_SW_MUX_CTL_PAD_NAND_DATA1 = (volatile unsigned int *) (0x20E0184);
66 #IOMUXC_SW_MUX_CTL_PAD_NAND_DATA2 = (volatile unsigned int *) (0x20E0188);
67 #IOMUXC_SW_MUX_CTL_PAD_NAND_DATA3 = (volatile unsigned int *) (0x20E018C);
68 #IOMUXC_SW_MUX_CTL_PAD_NAND_DATA4 = (volatile unsigned int *) (0x20E0190);
69 #IOMUXC_SW_MUX_CTL_PAD_NAND_DATA5 = (volatile unsigned int *) (0x20E0194);
70 #IOMUXC_SW_MUX_CTL_PAD_NAND_DATA6 = (volatile unsigned int *) (0x20E0198);
71 #IOMUXC_SW_MUX_CTL_PAD_NAND_DATA7 = (volatile unsigned int *) (0x20E019C);
72 #IOMUXC_SW_MUX_CTL_PAD_NAND_ALE = (volatile unsigned int *) (0x20E01A0);

```

```

73
74 *IOMUXC_SW_MUX_CTL_PAD_NAND_RE_B = 0x1U;
75 *IOMUXC_SW_MUX_CTL_PAD_NAND_WE_B = 0x1U;
76 *IOMUXC_SW_MUX_CTL_PAD_NAND_DATA0 = 0x1U;
77 *IOMUXC_SW_MUX_CTL_PAD_NAND_DATA1 = 0x1U;
78 *IOMUXC_SW_MUX_CTL_PAD_NAND_DATA2 = 0x1U;
79 *IOMUXC_SW_MUX_CTL_PAD_NAND_DATA3 = 0x1U;
80 *IOMUXC_SW_MUX_CTL_PAD_NAND_DATA4 = 0x1U;
81 *IOMUXC_SW_MUX_CTL_PAD_NAND_DATA5 = 0x1U;
82 *IOMUXC_SW_MUX_CTL_PAD_NAND_DATA6 = 0x1U;
83 *IOMUXC_SW_MUX_CTL_PAD_NAND_DATA7 = 0x1U;
84 *IOMUXC_SW_MUX_CTL_PAD_NAND_ALE = 0x1U;

```

## 1.3.1.2 2 USDHC2

IMX6      UART USDHC      1      GPIO

)

- USDHC2\_CLK

IOMUXC\_USDHC2\_CLK\_SELECT\_INPUT [DAISY]    0x2

)

)

- USDHC2\_CMD

IOMUXC\_USDHC2\_CMD\_SELECT\_INPUT [DAISY]    0x2

)

- USDHC2\_DATA0

IOMUXC\_USDHC2\_DATA0\_SELECT\_INPUT [DAISY]    0x2

)

- USDHC2\_DATA1

IOMUXC\_USDHC2\_DATA1\_SELECT\_INPUT [DAISY]    0x2

)

)

- USDHC2\_DATA2

IOMUXC\_USDHC2\_DATA2\_SELECT\_INPUT [DAISY] 0x1

)

- USDHC2\_DATA3

IOMUXC\_USDHC2\_DATA3\_SELECT\_INPUT [DAISY] 0x2

)

- USDHC2\_DATA4-7

USDHC2\_DATA4-7 DAISY 0x2 IOMUXC\_USDHC2\_DATA4\_SELECT\_INPUT  
[DAISY] IOMUXC\_USDHC2\_DATA5\_SELECT\_INPUT [DAISY] IOMUXC\_USDHC2\_DATA6\_SELECT\_INPUT  
[DAISY] IOMUXC\_USDHC2\_DATA7\_SELECT\_INPUT [DAISY] 0x2

)

```
86 //
87 #IOMUXC_USDHC2_CLK_SELECT_INPUT = (volatile unsigned int *) (0x020E0670);
88 #IOMUXC_USDHC2_CMD_SELECT_INPUT = (volatile unsigned int *) (0x020E0678);
89 #IOMUXC_USDHC2_DATA0_SELECT_INPUT = (volatile unsigned int *) (0x020E067C);
90 #IOMUXC_USDHC2_DATA1_SELECT_INPUT = (volatile unsigned int *) (0x020E0680);
91 #IOMUXC_USDHC2_DATA2_SELECT_INPUT = (volatile unsigned int *) (0x020E0684);
92 #IOMUXC_USDHC2_DATA3_SELECT_INPUT = (volatile unsigned int *) (0x020E0688);
93 #IOMUXC_USDHC2_DATA4_SELECT_INPUT = (volatile unsigned int *) (0x020E068C);
94 #IOMUXC_USDHC2_DATA5_SELECT_INPUT = (volatile unsigned int *) (0x020E0690);
95 #IOMUXC_USDHC2_DATA6_SELECT_INPUT = (volatile unsigned int *) (0x020E0694);
96 #IOMUXC_USDHC2_DATA7_SELECT_INPUT = (volatile unsigned int *) (0x020E0698);
97
98 #*IOMUXC_USDHC2_CLK_SELECT_INPUT = 0x2U;
99 #*IOMUXC_USDHC2_CMD_SELECT_INPUT = 0x2U;
100 #*IOMUXC_USDHC2_DATA0_SELECT_INPUT = 0x2U;
101 #*IOMUXC_USDHC2_DATA1_SELECT_INPUT = 0x2U;
102 #*IOMUXC_USDHC2_DATA2_SELECT_INPUT = 0x1U;
103 #*IOMUXC_USDHC2_DATA3_SELECT_INPUT = 0x2U;
104 #*IOMUXC_USDHC2_DATA4_SELECT_INPUT = 0x1U;
105 #*IOMUXC_USDHC2_DATA5_SELECT_INPUT = 0x1U;
```

```

106  IOMUXC_USDHC2_DATA6_SELECT_INPUT  = 0x1U;
107  IOMUXC_USDHC2_DATA7_SELECT_INPUT  = 0x1U;

```

### 1.3.1.3 3 USDHC2





0x17059

```

109 //
110 IOMUXC_SW_PAD_CTL_PAD_NAND_RE_B  = (volatile unsigned int *) (0x020E0404U);
111 IOMUXC_SW_PAD_CTL_PAD_NAND_WE_B  = (volatile unsigned int *) (0x020E0408U);
112 IOMUXC_SW_PAD_CTL_PAD_NAND_DATA00 = (volatile unsigned int *) (0x020E040CU);
113 IOMUXC_SW_PAD_CTL_PAD_NAND_DATA01 = (volatile unsigned int *) (0x020E0410U);
114 IOMUXC_SW_PAD_CTL_PAD_NAND_DATA02 = (volatile unsigned int *) (0x020E0414U);
115 IOMUXC_SW_PAD_CTL_PAD_NAND_DATA03 = (volatile unsigned int *) (0x020E0418U);
116 IOMUXC_SW_PAD_CTL_PAD_NAND_DATA04 = (volatile unsigned int *) (0x020E041CU);
117 IOMUXC_SW_PAD_CTL_PAD_NAND_DATA05 = (volatile unsigned int *) (0x020E0420U);
118 IOMUXC_SW_PAD_CTL_PAD_NAND_DATA06 = (volatile unsigned int *) (0x020E0424U);
119 IOMUXC_SW_PAD_CTL_PAD_NAND_DATA07 = (volatile unsigned int *) (0x020E0428U);
120
121 IOMUXC_SW_PAD_CTL_PAD_NAND_RE_B    = 0x17059;
122 IOMUXC_SW_PAD_CTL_PAD_NAND_WE_B    = 0x17059;
123 IOMUXC_SW_PAD_CTL_PAD_NAND_DATA00  = 0x17059;
124 IOMUXC_SW_PAD_CTL_PAD_NAND_DATA01  = 0x17059;
125 IOMUXC_SW_PAD_CTL_PAD_NAND_DATA02  = 0x17059;
126 IOMUXC_SW_PAD_CTL_PAD_NAND_DATA03  = 0x17059;
127 IOMUXC_SW_PAD_CTL_PAD_NAND_DATA04  = 0x17059;
128 IOMUXC_SW_PAD_CTL_PAD_NAND_DATA05  = 0x17059;
129 IOMUXC_SW_PAD_CTL_PAD_NAND_DATA06  = 0x17059;
130 IOMUXC_SW_PAD_CTL_PAD_NAND_DATA07  = 0x17059;

```

## 1.3.2

USDHC2

EMMC

USDHC2

### 1.3.2.1 1 eMMC

eMMC	MTFC4GACAJC	MTFC4GACAJC	HS200	HS400	SDR	DDR	52MHz
	MTFC4GACAJC	EXCSD	HS_TIMING	00h			

)

eMMC5.0Spec\_JESD84-B50 HS\_TIMING HS\_TIMING 0

)

)

## 11.3.2.2 2 USDHC2

Chapter 18 Clock Controller

Module	USDHC2	PLL2	PFD0	PFD2	USDHC2	CSCDR1[USDHC2_PC
CG2]						

)

- CSCMR1[USDHC2\_CLK\_SEL]

USDHC2_CLK_SEL	0	PLL2_PFD2	PLL2_PFD0	PLL2_PFD2	CSCMR1[USDHC2_CLK_SE
----------------	---	-----------	-----------	-----------	----------------------

)

)

- CSCDR1[USDHC2\_POPF]

USDHC2_POPF	1~8	2	CSCDR1[USDHC2_POPF]	0x1
-------------	-----	---	---------------------	-----

)

)

)

- CCGR6[CG2]

)

CCM_CCGR6	CCM_CCGR6[CG2]	11	11	4.3.2 CCM	GPIO
-----------	----------------	----	----	-----------	------

USDHC2_CLK_ROOT = PLL2_PDF2 / 2	PLL2_PDF2	396MHz	USDHC2_CLK_ROOT
---------------------------------	-----------	--------	-----------------

400KHz	USDHC2_SYS_CTRL	SDCLKFS[15:8]	DVS[7:4]
--------	-----------------	---------------	----------

- USDHC2\_SYS\_CTRL









SDCLKFS uSDHC SDR 1 2 4 8 16 32 64 128 256

DDR SDR

DVS uSDHC 1 16

USDHC2\_CLK

SDR USDHC2\_CLK = USDHC2\_CLK\_ROOT / SDCLKFS \* DVS

DDR USDHC2\_CLK = USDHC2\_CLK\_ROOT / SDCLKFS \* 2\* DVS

uSDHC\_CLK 400KHz eMMC SDR SDCLKFS 40h DVS 3 uSDHC\_C

```
192 /* 400KHz. */
193 //Single Data Rate mode, PLL2_PFD2 = 396Mhz, usdhc2_clk_root = 396Mhz / 2 = 198Mhz
194 //usdhc2_clk = [usdhc2_clk_root / (prescaler * divisor)
195 [//SDCLKFS = 0x40 = 128
196 [//DVS = 0x03 = 4
197 [//usdhc2_clk = 198Mhz/(128*4) = 0.387Mhz
198 [usdhc2_reg->SYS_CTRL &= ~(0xFF00 | (0xF0));
199 [usdhc2_reg->SYS_CTRL |= (0x0040U << 8) | (0x0003U << 4);
200 [
201 [/* */
202 [while (!(usdhc2_reg->PRES_STATE & 0x8U))
203 {
204 [
205 }
```

198 SYS\_CTRL SDCLKFS DVS 0

199 SYS\_CTRL SDCLKFS DVS 0x40 0x03

202 PRES\_STATE SDSTB 1



)

01\_emmc

## 1.3.3 USDHC2

USDHC2

USDHC2\_SYS\_CTRL RSTA EMMC RSTA RSTA 0x1000000

)

)

01\_emmc mmc.c

```
209 /*****
210 *      USDHC_Reset
211 *      usdhc
212 *      unsigned int mask
213 *      unsigned int timeout
214 *      unsigned char
215 *      0      1
216 *
217 * -----
218 * 2020/02/28   V1.0   LJZ
219 *****/
220 static unsigned char USDHC_Reset(unsigned int mask, unsigned int timeout)
221 {
222     usdhc2_reg->SYS_CTRL |= mask;
223
224     while ((usdhc2_reg->SYS_CTRL & mask) != 0U)
225     {
226         if (timeout == 0U)
227         {
228             break;
229         }
230         timeout--;
231     }
232
233     return ((!timeout) ? 0 : 1);
```

```

234 }
133
134 /*****
135 *      USDHC_Init
136 *      usdhc
137 *
138 *
139 *
140 *
141 * -----
142 * 2020/02/28   V1.0   LJZ
143 *****/
144 void USDHC_Init(void)
145 {
146     CCM_CCGR6 = (volatile unsigned int *)(0x20C4080);
147
148     EMMC_PinConfig();
149
150     /*      USDHC      */
151     *CCM_CCGR6 = ((*CCM_CCGR6) & ~(3U << 0x04)) | (((unsigned int)3U) << 0x04);
152
153     /*      USDHC      */
154     if( USDHC_Reset(0x1000000U, 100U) == 1 )
155     {
156         /*      */
157         printf("Reset Success\n\r");
158     }
159     else
160     {
161         /*      */
162         printf("Reset false\n\r");
163         return;
164     }
165     .
166     .
167     .
207 }

```

## 1.3.4 eMMC

```
47 typedef enum
48 {
49     RESPONSE_TYPE_NONE = 0U,
50     RESPONSE_TYPE_R1    = 1U,
51     RESPONSE_TYPE_R1b   = 2U,
52     RESPONSE_TYPE_R2    = 3U,
53     RESPONSE_TYPE_R3    = 4U,
54     RESPONSE_TYPE_R4    = 5U,
55     RESPONSE_TYPE_R5    = 6U,
56     RESPONSE_TYPE_R5b   = 7U,
57     RESPONSE_TYPE_R6    = 8U,
58     RESPONSE_TYPE_R7    = 9U
59 }USDHC_Responses_Type;
60
61 typedef struct
62 {
63     unsigned int  index;
64     unsigned int  arg;
65     unsigned int  response[4];
66     USDHC_Responses_Type response_type;
67     unsigned int  mix_ctrl;
68     unsigned int  xfr_tpy;
69 }USDHC_Command;
```

USDHC_Responses_Type	USDHC_Command	index	arg	response	i
_mark	CMD_XFR_TYP				
	CMD		uSDHC2_PRES_STATE	CIHB	CDIHB

)

)

```

248  /*
249  // CIHB 0x01
250  while( usdhc2_reg->PRES_STATE & 0x01U )
251  {
252  }
253
254  /*
255  // CDIHB 0x02
256  while( usdhc2_reg->PRES_STATE & 0x02U )
257  {
258  }

```

```

248 258          CIHB 0x1U  CDIHB 0x2U          1          CIHB CDIHB 0
          CMD          uSDHC2_MIX_CTRL DDR_EN(DDR ) DTDSEL(          1          0          ) BCEN

```

)

)

```

262  /*
263  // BCEN : 0x02U
264  // DDR_EN 0x08U
265  // DTDSEL 0x10U
266  // MSBSEL 0x20U
267  usdhc2_reg->MIX_CTRL &= ~(0x20U | 0x2U | 0x10U | 0x8U );
268  usdhc2_reg->MIX_CTRL |= ((command->mix_ctrl) & (0x20U | 0x2U | 0x10U | 0x8U ));

```

```

DDR_EN 0x8U DTDSEL 0x10 BCEN 0x2U MSBSEL 0x20U

```

```

267  uSDHC2_MIX_CTRL DDR_EN DTDSEL BCEN MSBSEL 0

```

```

268  uSDHC2_MIX_CTRL mix_ctrl

```

```

          uSDHC2_CMD_ARG

```

```

270  usdhc2_reg->CMD_ARG = command->arg;

```

```

          uSDHC2_CMD_XFR_TYP          CRC

```

)

)

```
272  /*      */
273  // CDMINX  0x3F000000U
274  // DPSEL   0x200000U
275  // CICEN   0x100000U
276  // CCCEN   0x80000U
277  // PSPTYP[ 1: 0]  0x30000U
278  usdhc2_reg->CMD_XFR_TYP &= ~(0x3F000000U | 0x200000U | 0x100000U | 0x80000U | 0x30000U);
279
280  usdhc2_reg->CMD_XFR_TYP = (((command->index << 24U) & 0x3F000000U) |
281  ((command->xfr_typ) & (0x3F000000U | 0x200000U | 0x100000U |
0x80000U | 0x30000U)));
```

CMDINX DPSEL CMDTYP CICEN CCCEN RSPTYP 0x3F000000U 0x200000U 0x100000U 0x80000U 0x30000U

278 CMD\_XFR\_TYP CMDINX DPSEL CMDTYP CICEN CCCEN RSPTYP

280 CMD\_XFR\_TYP

CMD\_XFR\_TYP uSDHC2 uSDHC2\_INT\_STATUS CC

)

```
282  /*      */
283  // CC      0x01
284  while( !(usdhc2_reg->INT_STATUS & 0x1U ) )
285  {
286  }
287
288  /*      */
289  // CC      0x01
290  usdhc2_reg->INT_STATUS &= 0x01U ;
```

284-286 INT\_STATUS CC 1 1

290 INT\_STATUS CC

)

:

```

295 /*****
296 *      USDHC_ReadResponse
297 *      Response
298 *      USDHC_Command* command  USDHC
299 *
300 *
301 *      □
302 * -----
303 * 2020/02/28□      V1.0□  LJZ□
304 *****/
305 void USDHC_ReadResponse( USDHC_Command *command)
306 {
307     □unsigned int i;
308     □
309     □if( command->response_type != ResponseTypeNone )
310     □{
311         □□command->response[ 0] = usdhc2_reg->CMD_RSP0;
312         □□
313         □□if( command->response_type == ResponseTypeR2 )
314         □□{
315             □□□command->response[ 1] = usdhc2_reg->CMD_RSP1;
316             □□□command->response[ 2] = usdhc2_reg->CMD_RSP2;
317             □□□command->response[ 3] = usdhc2_reg->CMD_RSP3;
318
319             □□□i = 4;
320             □□
321             □□□do
322             □□□{
323                 □□□□command->response[ i - 1] <= 8;
324                 □□□□if ( i > 1)
325                 □□□□{
326                     □□□□□command->response[ i - 1] |= (( command->response[ i - 2] & 0xFF000000U) >> 24U);
327                 □□□□}
328             □□□} while (i--);
329     □□}

```

```
330 }
331 }
```

313-329 R2 uSDHC2\_CMD\_RSP0-3 7

1.3.4.2 1 eMMC

CMD0 CMD0 CMD0 eMMC

	0x00000000	
	0xF0F0F0F0	
-	0xFFFFFFFF	

eMMC IDLE 0x00000000 MIX\_CTRL 0x0 CMD\_XFR\_TYP 0

```
16 command.index = 0;
17 command.arg = 0;
18 command.mix_ctrl = 0;
19 command.xfr_typ = 0;
20 command.response_type = ResponseTypeNone;
21
22 // CMD0
23 USDHC_SendCommand( &command);
```

13.1.9 CMD1 eMMC OCR 31 eMMC CMD1

	0	R3

```
25 for( i = 0; i < 10; i++ )
26 {
27     // CMD1
28     // R3
29     // 48bit RSPTYP[1:0] = 0b10 = 2
30     command.index = 1;
31     command.arg = 0;
32     command.mix_ctrl = 0;
33     command.xfr_typ = 2 << 16;
```



```
34  command.response_type = ResponseTypeR3;
35
36  USDHC_SendCommand( &command);
37
38  //  EMMC
39  //  OCR 31  EMMC          1      0
40  if ( command.response[0] & (1U << 31U) )
41  {
42      printf("MMC is Ready ok \n\r");
43      break;
44  }
45  else
46  {
47      printf("MMC is Busy. \n\r");
48  }
49  }
```

1.3.4.3      4-1.4

cd 01\_emmc

1.3.4.4      4-1.4



1.3.4.5      2      CID

eMMC      Ready      Ready      CMD2      eMMC Identification      CMD2

		R2

0      ResponseTypeR2 CMD\_XFR\_TYP      CRC      CCCEN      1      RSPTYP[1:0]      1

```
57      //  CMD2
58  //      R2
59  //      136bit RSPTYP[1:0] = 0b01 = 1
60  command.index = 2;
61  command.arg = 0;
62  command.mix_ctrl = 0;
```

```
63  __command.xfr_typ = 1 << 16 | 1 << 19;
64  __command.response_type = ResponseTypeR2;
65  __
66  __JSDHC_SendCommand( &command);
```

CMD2 CID CID ManufacturerID CBX ApplicationID Product name

```
68  __printf("ManufacturerID: 0x%x \n\r", (unsigned char)((command.response[3U] &
0xFF000000U) >> 24U));
69  __printf("CBX: 0x%x \n\r", (unsigned char)((command.response[3U] & 0x00FF0000U) >> 16U));
70  __printf("ApplicationID: 0x%x \n\r", (unsigned char)((command.response[3U] & 0x0000FF00U) >>
8U));
71  __printf("Product version: 0x%x \n\r", (unsigned char)((command.response[1U] & 0xFF000000U)
>> 24U));
```

02\_emmc



1.3.4.6 3 RCA

eMMC CMD3 RCA eMMC Standby CMD3 16 RCA 16 R1

	[31:16]RCA,[15:0]	R1

RCA 1 16 0 1<<16 ResponseTypeR1 CMD\_XFR\_TYP C1CEN 1 CRC

16|1<<19|1<<20

```
73  __// Send CMD3
74  __command.index = 3;
75  __command.arg = 1 << 16;
76  __command.mix_ctrl = 0;
77  __command.xfr_typ = 2 << 16 | 1 << 19 | 1 << 20;
78  __command.response_type = ResponseTypeR1;
```

R1 13.1.6 Response CRC

```
82  __// COM_CRC_ERROR: (1U << 23U)
83  __// ILLEGAL_COMMAND: (1U << 22U)
84  __if (!(command.response[0] & ((1U << 23U) | (1U << 22U))))
```

```
85 {}
86 printf("Emmc in Stand-by State\n\r");
87 }
88 else
89 {}
90 printf("SDMMC_R1 Error\n\r");
91 return 0;
92 }
```

1.3.4.7 4 CSD EXCSD

Standby CMD9 CSD CMD9

	[31:16]RCA,[15:0]	R2

CMD9 16 RCA 16 R2 RCA 1 ResponseTypeR2 CMD\_XFR\_TYP CRC C  
CMD\_XFR\_TYP 1<<16|1<<19 CSD CSD

```
94 // Send CMD9 CSD
95 command.index = 9;
96 command.arg = 1 << 16;
97 command.mix_ctrl = 0;
98 command.xfr_typ = 1 << 16 | 1 << 19;
99 command.response_type = ResponseTypeR2;
100
101 USDHC_SendCommand( &command);
102
103 printf("CSD structure: %x \n\r", (unsigned char)((command.response[ 30] & 0xC0000000U) >>
30));
```

EXCSD 512 Standby transfer 13-1.2

	[31:16]RCA,[15:0]	R1/R1b

CMD7 16 RCA 16 R1 R1b RCA 1 ResponseTypeR1 CMD\_XFR\_TYP  
R\_TYP 2<<16|1<<19|1<<20

```
105 // Send CMD7 tranfaer
106 command.index = 7;
107 command.arg = 1 << 16;
108 command.mix_ctrl = 0;
109 command.xfr_typ = 2 << 16 | 1 << 19 | 1 << 20;
110 command.response_type = ResponseTypeR1;
111
112 USDHC_SendCommand( &command);
```

)

```
114 if (!(command.response[0] & ((1U << 23U) | (1U << 22U))))
115 {
116 printf("Emmc in Transfer State\n\r");
117 }
118 else
119 {
120 printf("SDMMC_R1 Error\n\r");
121 return 0;
122 }
```

114            22 23            0    CMD7

Transfer        CMD8   EXCSD CMD8

		R1

CMD8    0            ResponseTypeR1   CMD\_XFR\_TYP            CICEN    1   CRC            CCCEN    1    R5

9|1<<20   MIX\_CTRL            DTDSEL    1            1<<4            385   387

```
366 /*****
367 *        MMC_ReadExCsd
368 *        ECsd
369 *
370 *
371 *
372 *                           
373 * -----
```

```

374 * 2020/02/28   V1.0   LJZ
375 *****/
376 void MMC_ReadExCsd(void)
377 {
378     _USDHC_Command command;
379     _unsigned char data[512];
380
381     _command.index = 8;
382     _command.arg = 0;
383     //
384     //
385     _command.mix_ctrl = 1 << 4 | 0x1 << 1;
386     _command.xfr_type = 2 << 16 | 1 << 19 | 1 << 20 | 1 << 21;
387     _command.response_type = ResponseTypeR1;
388
389     _usdhc2_reg->BLK_ATT = 512 | 1 << 16;
390
391     _USDHC_SendCommand( &command);
392
393     _USDHC_ReadWordData((unsigned int*)data);
394
395     _printf("Card type: 0x%x \n\r", data[196U]);
396     _printf("DataBusWidth: 0x%x\n\r", data[183U]);
397     _printf("CSD structure: %x\n\r", data[194U]);
398     _printf("Extended CSD revision: %x\n\r", data[192U]);
399 }

```

389        BLK\_ATT                512        1

393        USDHC\_ReadWordData

395-398        CSD

      USDHC\_ReadWordData

```

332 /*****
333 *      USDHC_ReadWordData
334 *
335 *      unsigned int* data
336 *
337 *

```

```

338 *
339 * -----
340 * 2020/02/28 V1.0 LJZ
341 *****/
342 void USDHC_ReadWordData(unsigned int* data)
343 {
344     unsigned int i;
345
346     printf("Waitting Read buffer ready\n\r");
347
348
349     BRR    0x1<<5
350     while (!( usdhc2_reg->INT_STATUS & (1 << 5) ))
351     {
352
353     }
354
355     printf("Read buffer has readied\n\r");
356     usdhc2_reg->INT_STATUS &= (1 << 5);
357
358     for(i=0; i<512/4; i++)
359     {
360         data[i] = usdhc2_reg->DATA_BUFF_ACC_PORT;
361     }
362
363     printf("Read buffer complete\n\r");
364 }

```

350 INT\_STATUS BRR 1

358-361 512

EXCSD Card type DataBusWidth CSD Structure EXCsd Version

```

395 printf("Card type: 0x%x \n\r", data[196U]);
396 printf("DataBusWidth: 0x%x\n\r", data[183U]);
397 printf("CSD structure: %x\n\r", data[194U]);
398 printf("Extended CSD revision: %x\n\r", data[192U]);

```

4.4.4 3.4



## 1.3.4.8 5

eMMC 4bit

```
402 /*****
403  *      MMC_SetBusWidth
404  *
405  *
406  *
407  *
408  *      □
409  * -----
410  * 2020/02/28□ V1.0□ LJZ□
411  *****/
412 void MMC_SetBusWidth( void)
413 {
414     □USDHC_Command command;
415     □
416     □command.index = 6;
417     □command.arg = (3<<24) | (183<<16) | (1<<8) | (0<<0);
418     □command.mix_ctrl = 0;
419     □command.xfr_typ = 3 << 16 | 1 << 19;
420     □command.response_type = ResponseTypeR1b;
421     □
422     □USDHC_SendCommand( &command);
423     □
424     □while(1)
425     □{
426         □□command.index = 13;
427         □□command.arg = 1U << 16;
428         □□command.mix_ctrl = 0;
429         □□command.xfr_typ = 2 << 16 | 1 << 19 | 1 << 20;
430         □□command.response_type = ResponseTypeR1;
431
432         □□USDHC_SendCommand( &command);
433         □
434         □□//
435         □□if ( command.response[0] & (1U << 8U) )
```

```
436 {}
437 {}printf("SetBusWidth Complete. \n\r");
438 {}break;
439 {}
440 {}else
441 {}{
442 {}printf("Emmc is Busy. \n\r");
443 {}}
444 {}
445 }
```

CMD6

	[31:26] 0, [25:24] Access, [23:16] Index, [15:8] Value, [7:3] Set to 0, [2:0] Cmd Set	R1b

CMD6      26 31   0 Access                      eMMC5.0Spec\_JESD84-B50                      Access 3    EXCSD



Index   EXCSD                      EXCSD   183                      Index      183



         eMMC5.0Spec\_JESD84-B50      BUS\_WIDTH                      SDR 4bit                      Value      1



CMD6            (3<<24) | (183<<16) | (1<<8) |  
(0<<0)            ResponseTypeR1   CMD\_XFR\_TYP                      CICEN      1   CRC                      1            RSPTYP[1:(

Transfer            CMD13    ORC      ORC    31    eMMC                      424 444

eMMC                      IMX6 USDHC2

```
447 /*****
448 *        USDHC_SetBusWidth
449 *        USDHC
450 *
451 *
452 *
453 *                      {}
```



```
454 * -----
455 * 2020/02/28   V1.0   LJZ
456 *****/
457 void USDHC_SetBusWidth( void)
458 {
459   [usdhc2_reg->PROT_CTRL &= ~(6U << 1);
460   [usdhc2_reg->PROT_CTRL |= 1U<<1;
461 }
```

USDHC2\_PROT\_CTRL DTW[1:0] USDHC2 DTW[1:0] 0x1

)

```
459   USDHC2_PROT_CTRL DTW[1:0] 0
460   USDHC2_PROT_CTRL DTW[1:0] 0x01
```

03\_emmc main.c mmc.c mmc.h

# 1.3.5

Block Block CMD16 CMD16

	[31:26] Block	R1

Block

```
463 *****/
464 *   MMC_SetBlockSize
465 *   MMC Block
466 *
467 *
468 *
469 *   [
470 * -----
471 * 2020/02/28   V1.0   LJZ
472 *****/
473 void MMC_SetBlockSize( void)
474 {
475   [USDHC_Command command;
```

```

476
477 command.index = 16;
478 command.arg = 512;
479 command.mix_ctrl = 0;
480 command.xfr_typ = 2 << 16 | 1 << 19;
481 command.response_type = ResponseTypeR1;
482
483 USDHC_SendCommand( &command);
484
485 //COM_CRC_ERROR 1U << 23U
486 //ILLEGAL_COMMAND 1U << 22U
487 if (!(command.response[0] & ((1U << 23U) | (1U << 22U))))
488 {
489     printf("Emmc Set Blocksize 0k\n\r");
490 }
491 else
492 {
493     printf("Emmc Set Blocksize Error\n\r");
494 }
495 }

```

```

477-481      CMD16      512 CMD_XFR_TYP      CRC      CCCEN      1      RSPTYP[1:0]      2      ;
487      R1      CRC      COM_CRC_ERROR      ILLEGAL_COMMAND

EXCSD

```

```

497 /*****
498 *      MMC_WriteBlock
499 *      BLOCK      EMMC
500 *      unsigned int address
501 *      const unsigned int* data
502 *
503 *
504 *
505 * -----
506 * 2020/02/28      V1.0      LJZ
507 *****/
508 void MMC_WriteBlock(unsigned int address, const unsigned int* data)
509 {
510     USDHC_Command command;

```

```

511 unsigned int i;
512
513 command.index = 24;
514 command.arg = address;
515 command.mix_ctrl = 0;
516 command.xfr_typ = 2 << 16 | 1 << 19 | 1 << 20 | 1 << 21;
517 command.response_type = ResponseTypeR1;
518
519 usdhc2_reg->BLK_ATT = 512 | 1 << 16;
520
521 USDHC_SendCommand( &command);
522
523 //COM_CRC_ERROR 1U << 23U
524 //ILLEGAL_COMMAND 1U << 22U
525 if (!(command.response[0] & ((1U << 23U) | (1U << 22U))))
526 {
527     printf("CMD24 Success\n\r");
528 }
529 for(i=0; i<512/4; i++)
530 {
531     usdhc2_reg->DATA_BUFF_ACC_PORT = data[i];
532 }
533
534 else
535 {
536     printf("CMD24 Error\n\r");
537 }
538 }

```

513-517      CMD24                      ResponseTypeR1   CMD\_XFR\_TYP                      CICEN    1   CRC  
<< 16 | 1<<19 | 1<<20 | 1<<21

## Block

```

540 /*****
541  *      MMC_ReadBlock
542  *      BLOCK
543  *      unsigned int address
544  *      unsigned int* data
545  *
546  *

```

```

547 *
548 * -----
549 * 2020/02/28 V1.0 LJZ
550 *****/
551
552 void MMC_ReadBlock(unsigned int address, unsigned int* data)
553 {
554     USDHC_Command command;
555     unsigned int i;
556
557     command.index = 17;
558     command.arg = address;
559
560
561     command.mix_ctrl = 0x1 << 4 | 0x1 << 1;
562     command.xfr_typ = 2 << 16 | 1 << 19 | 1 << 20 | 1 << 21;
563     command.response_type = ResponseTypeR1;
564
565     usdhc2_reg->BLK_ATT = 512 | 1 << 16;
566
567     USDHC_SendCommand( &command);
568
569     // COM_CRC_ERROR 1U << 23U
570     //ILLEGAL_COMMAND 1U << 22U
571     if (!(command.response[0] & ((1U << 23U) | (1U << 22U))))
572     {
573         printf("CMD17 Success\n\r");
574     }
575     else
576     {
577         printf("CMD17 Error\n\r");
578     }
579
580     USDHC_ReadWordData( data);
581 }

```

561-563	CMD17	ResponseTypeR1	CMD_XFR_TYP	CICEN	1	CRC
<< 16   1<<19   1<<20   1<<21	MIX_CTRL	DTDSEL	1			

```

152 // EMMC Block
153 MMC_SetBlockSize();
154
155 for(i=0; i<512/4; i++)
156 {
157     WriteData[i] = i;
158 }
159 // Block
160 MMC_WriteBlock(0, WriteData);
161
162 MMC_ReadBlock(0, ReadData);
163
164 for(i=0; i<512/4; i++)
165 {
166     printf("Data[ %d]: %d\n\r", i, ReadData[i]);
167 }

```

## 1.3.6 4-1.4

cd 04\_emmc

## 1.3.7 4-1.4

image not found or type unknown



image not found or type unknown



04\_emmc    main.c mmc.c mmc.h