Freescale Semiconductor
Application Note

Document Number: AN2906
Rev. 0, 03/2005

MultiMediaCard/Secure Digital Host Controller Addendum

MC9328MX21

by: Paul Cheung

1 Introduction

This document is written as an addendum to the i.MX21Applications Processor reference manual's MultimediaCard/Secure Digital Host Controller (MMC/SDHC) chapter. This document provides suggestions and guidelines for the following:

- Hardware connectivity and IO pad configuration
- MMC/SD clocking
- Interrupt and DMA Channels for the MMC/SD host controllers
- Relationship of the STATUS and INT_CNTR registers
- Sequence of SDIO/non-SDIO interrupt handling
- Multiple MultiMediaCard support
- Card detection

SPI mode will not be discussed in this document.

Contents

2 Hardware Connectivity

The Hardware connectivity between the MMC/SD host controller and cards is the collaboration of the MMC/SD host controller module, GPIO module, and the System Controller settings.

2.1 The SD Signal and MMC Signal

The MMC/SD host controller pin assignment is provided in Table 1.

Table 1. Pin Assignment of the MMC/SD Host Controllers

SD				MMC	
Pin #	**Name**	**Description**	**Other Function in SD/SDIO**	**Name**	**Description**
1	DAT3	Data Line [bit-3]	Card Detect (SD and SDIO)	Reserved	Reserved for future use
2	CMD	Command/Response	–	CMD	Command/Response
3	Vss1	Supply Voltage Ground	–	Vss1	Supply Voltage Ground
4	Vdd	Supply Voltage	–	Vdd	Supply Voltage
5	CLK	Clock	–	CLK	Clock
6	Vss2	Supply Voltage Ground	–	Vss2	Supply Voltage Ground
7	DAT0	Data Line [bit-0]	–	DAT	–
8	DAT1	Data Line [bit-1]	Interrupt (SDIO only)	N/A	–
9	DAT2	Data Line [bit-2]	Read Wait (SDIO only)	N/A	–

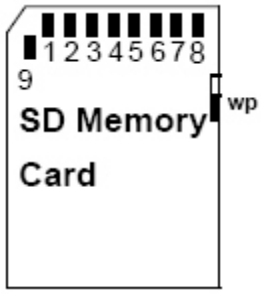

Figure 1. Physical Structure of an SD Card

2.2 The Physical Connection with i.MX21

Table 2 identifies the pin names of MMC/SDHC-1 and MMC/SDHC-2 and the associated GPIO pins.

Table 2. Pin Names of SDHC and the According GPIO Pins

	MMC/SDHC1			MMC/SDHC2		
Signal Name in SD Spec	**Signal Name in i.MX21**	**Associated GPIO**	**Pull-Up/ Pull-Down**	**Signal Name in i.MX21**	**Associated GPIO**	**Pull-Up/ Pull-Down**
CLK	SD1_CLK	PE23	N/A	SD2_CLK	PB9	N/A
CMD	SD1_CMD	PE22	Internal Pull-Up	SD2_CMD	PB8	Internal Pull-Up
DAT3	SD1_D3	PE21	Internal Pull-Up[1]	SD2_D3	PB7	Internal Pull-Up[1]
DAT2	SD1_D2	PE20	Internal Pull-Up	SD2_D2	PB6	Internal Pull-Up
DAT1	SD1_D1	PE19	Internal Pull-Up	SD2_D1	PB5	Internal Pull-Up
DAT0	SD1_D0	PE18	Internal Pull-Up	SD2_D0	PB4	Internal Pull-Up

[1] For the card detection function, the user may occasionally use an external pull-low resistor. For more details about card detection, please read Table 3 Footnote in Section 2.3, “Setting of the GPIO Pins” and Section 8, “Card Detection”.

2.3 Setting of the GPIO Pins

Every GPIO in the i.MX21 processor is multiplexed with different functional modules in i.MX21. To use the MMC/SDHC function, the user must configure the GPIO pad to be used as the primary function. To configure the function of a GPIO pad, use the registers in the GPIO module identified in Table 3.

Table 3. GPIO Register Settings for MMC/SDHC Function

Set 1 / 0	SD1_CLK	SD1_CMD	SD1_D3	SD1_D2	SD1_D1	SD1_D0
DDIR	not necessary to be configured					
OCR1 / OCR2	not necessary to be configured					
ICONFA1 / ICONFA2	not necessary to be configured					
ICONFB1 / ICONFB2	not necessary to be configured					
DR	not necessary to be configured					
GIUS	0	0	0	0	0	0
SSR	not necessary to be configured					
ICR1 / ICR2	not necessary to be configured					
IMR	not necessary to be configured					
ISR	not necessary to be configured					
GPR	0	0	0	0	0	0
PUEN	0	1	1[1]	1	1	1

[1] According to reference [2], an internal pull-up resistor in the SD card for card detection is optional. The MMC/SDHC in i.MX21 supports this feature. When using this card detection feature, the interrupt pull-up resistor of SD_D3 must not be used (set to 0) at the card identification stage. Otherwise, continue to use the interrupt pull-up resistor (set to 1) in i.MX21. More details about card detection is discussed in Section 8, “Card Detection”.

The GIUS register is configured to use multiplexed functions.

The CPR register is configured to use the multiplexed primary function. There may be two functions multiplexed which are primary and secondary functions.

The PUEN register is configured (set to 1) to use internal pull-up resistor.

Please read sample code `SDHC_Port_Init()` in Appendix A on page 14 for reference.

Details of GPIO setting can be found in General-Purpose I/O chapter in [1].

2.4 Driving Strength

In the i.MX21 processor, the user can set the driving strength for different groups of IO pads. Depending on the PCB design and the use of the GPIO, the user can meet their requirements for different connected devices. For example, the user can set the DSCR1 register value to increase or decrease the driving strength for the entire group of IO pads for MMC/SDHC-1. Once set, the driving strength of all six IO pins is changed within the group for MMC/SDHC-1 (SD1_CLK, SD1_CMD, SD1_DAT3, SD1_DAT2, SD1_DAT1 and SD1_DAT0).

Table 4. Setting Driving Strength

Location	MMC/SDHC-1		MMC/SDHC-2	
Register	DSCR1			
Field	DS_SLOW1	setting: 000 = 2mA 001 = 4mA 011 = 8mA 111 = 12mA	DS_SLOW7	setting: 000 = 2mA 001 = 4mA 011 = 8mA 111 = 12mA
Bit Number	[2:0]	–	[21:19]	–

Please read Example 1 `SDHC_Port_Init()` in Appendix A on page 14 for reference.

For more details about driving strength settings, please read System Control chapter in [1].

2.5 Operation Logic Voltage

Generic MMC and SD memory cards support an operating voltage of 2.0V to 3.6V. The MMC/SD host controllers in i.MX21 processors have individual operation voltage inputs; NVdd2 for MMC/SDHC-1 and NVdd5 for MMC/SDHC-2. Users can easily adjust the operating voltage inputs to meet their design requirements. For example, if NVdd2 is changed from 3.3V to 3.6V, then the logic voltage of all six IO pins for MMC/SDHC-1 are changed to 3.6V within the group (SD1_CLK, SD1_CMD, SD1_DAT3, SD1_DAT2, SD1_DAT1 and SD1_DAT0). Please note that the same voltage supply must also be provided to the Vdd (Pin 4) of the SD card.

You can change the supply voltages to NVdd2 and NVdd5 at anytime, as necessary, to change the operating voltages of the SD cards. There are no register setting changes required to change the operating voltage inputs for NVdd2 or NVdd5.

For more information read the Signal Descriptions and Pin Assignments chapter of reference [1].

3 Clocking of MMC/SD Host Controller

The clock source of the MMC/SD host controllers in the i.MX21 processor is Peripheral Clock 2. To obtain the appropriate speed of MMCCLK, the user must set the clock tree using the following two steps. See Figure 2

1. Set the System Clock to Peripheral Clock 2.
2. Set the Peripheral Clock 2 to MMCCLK.

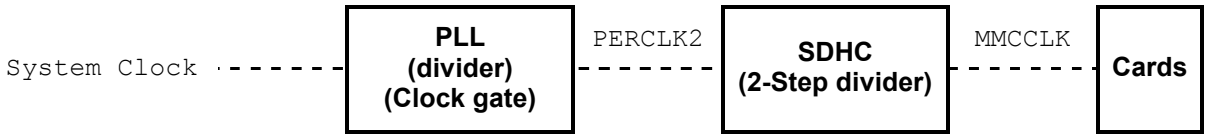

Figure 2. MMC/SDHC Clocking

Furthermore, the user can gate the Peripheral Clock 2 path to save power. The clocking of the MMC/SD host controller is the collaboration between the MMC/SD host controller and the PLL module.

3.1 Peripheral Clock 2

The first step is to set the Peripheral Clock 2 (PERCLK2) of the PLL, which is the primary clock source of the entire MMC/SD host controller. However, this first step is out of the scope of this document. For information on setting the PERCLK2, see the Phase Locked Loop (PLL), Clock and Reset Controller chapter of reference [1].

Please note that the Peripheral Clock2 from the PLL module is shared by MMC/SDHC-1, MMC/SDHC-2, CSPI-1, CSPI-2, and CSPI-3. Thus, while setting the PERCLK2, users must consider all devices that are connected to those modules.

3.2 MMCCLK

After setting the Peripheral Clock 2, the second step is to set the value of the CLK_RATE register which divides the MMC/SDHC clock source (Peripheral Clock 2) into the suitable MMCCLK speed provided to the memory card. The CLK_RATE register is actually a 2-step divider. The clock can be divided by:

- CLK_PRESCALER field, bit[15:4]
- CLK_DIVIDER field, bit[3:0]

For the detail settings of these 2 fields, please read the MMC/SD host controller chapter in [1].

Please read Example 2 code `SDHC_Set_Clk_Rate()` in Appendix A for reference.

3.3 Clock Gating

This section discusses manually starting and stopping the MMC/SDHC and gating using PERCLK2.

3.3.1 Start/Stop MMCCLK

To start and stop the MMCCLK manually, there is a START_CLK bit and a STOP_CLK bit in the STR_STP_CLK register. During data transfer, the MMC/SD host controller occasionally will stop and re-start the MMCCLK automatically as necessary.

Please note that even when the MMCCLK is not running, an SDIO interrupt can still be detected.

3.3.2 PERCLK2 Gating

For power saving, the clock to each module in the i.MX21 processor can be gated. Without the clock going into the MMC/SD host controllers, any read or write to any register in the MMC/SDHC is invalid and no MMCCLK will be generated. Thus, this clocking could also be seen as a module enable bit. Please note that this enable bit is not in the MMC/SDHC but rather in the PLL module.

- MMC/SDHC1—SDHC1_EN bit (bit-9) of PCCR0 register
- MMC/SDHC2—SDHC2_EN bit (bit-10) of PCCR0 register

Please read Example 3 code `Module_Enable_0()` in Appendix A for reference. For the detail settings of these two bits, please read the Phase Locked Loop (PLL), Clock and Reset Controller chapter in [1].

Please note that even the PERCLK2 to MMC/SDHC is gated (SDHC1_EN or SDHC_EN bit is 0), the SDIO interrupt can be detected whether the SDIO_WAKEUP_EN bit is enabled.

3.4 Valid Clock Speed of MMCCLK

In Clock Frequency Identification Mode, according to SD and MMC protocol ([2] and [4]), the MMCCLK speed must be under 400 kHz.

In Data Transfer Mode:

- For SD memory cards or SDIO cards, the maximum MMCCLK speed is 25 MHz.
- For MMC:
  - For less than or equal to 10 cards, the maximum MMCCLK speed is 20 MHz.
  - For less than or equal to 30 cards, the maximum MMCCLK speed is 5 MHz.

4 The Interrupt Source and DMA Channels for MMC/SDHC

This section discusses the interrupt source and DMA channels for the MMC/SD host controllers.

4.1 The Interrupt Source Channels

The following interrupt source channels are reserved for the MMC/SD host controllers:

- MMC/SDHC1—Interrupt Source 11
- MMC/SDHC2—Interrupt Source 12

For the detail settings of these 2 bits, please read the ARM926EJ-S™ Interrupt Controller (AITC) chapter in [1].

4.2 DMA Channels

No DMA channel is dedicated for the MMC/SD host controllers. Any DMA channel for MMC/SDHC1 and MMC/SDHC2 can be used for data transfer. The setting of DMA channels is out of the scope of this document. For more information see the Direct Memory Access Controller (DMAC) chapter of reference [1].

5 Relationship between STATUS and INT_CNTR Registers

It is recommended that the user thoroughly read this section before SDIO driver development. As discussed in the MMC/SD host controller chapter in [1], there are relationships between certain functions of the bits in the STATUS and INT_CNTR registers. However, it is important to understand the use of these bits. In this section, specific situations are discussed to clarify these relationships.

A. **Write-1-to-clear** any bit in the STATUS register does not clear the corresponding interrupt.

   This write-1-to-clear feature added for mask 2L45X is used so that software restricts a global variable when an MMC/SDHC interrupt has been served.

B. Writing to any bit in the INT_CNTR register clears all interrupt requests from MMC/SDHC to AITC (ARM® Interrupt Controller).

   — regardless if the current value of the bit is 0 or 1

   — regardless if writing 0 or 1 to un-mask/mask any interrupt

   For Example:

- When INT_CNTR = 0x00000036:

  BUF_READY and READ_OP_DONE interrupts are un-masked and trigger at the same time.

- In the Interrupt Service Routine (ISR), when INT_CNTR = 0x0000003e to mask BUF_READY interrupt:

  This action is equal to writing 1 to BUF_READY bit and writing 0 to READ_OP_DONE.

  BUF_READY and READ_OP_DONE interrupts are cleared at the same time.

  After the program exits the ISR (Interrupt Service Routine), it will not jump into the ISR immediately although the READ_OP_DONE interrupt is not served. It will only jump into the ISR again if there is another MMC/SDHC interrupt occurring.

C. SDIO interrupt is an exception to point B. This interrupt differs from non-SDIO interrupts even though the interrupt request is cleared, when:

   a) the SDIO interrupt is not masked by writing 1 to SDIO-bit in INT_CNTR register,

   and

   b) the interrupt source from the SDIO card is not cleared after the program exits the ISR.

The SDIO interrupt is triggered again and the system will immediately go into ISR again because the SDIO interrupt is a level-trigger interrupt from the card. This does not occur for non-SDIO interrupts.

D. When the FIFO is empty, the FIFO_READY interrupt can occur simultaneously with the READ_OP_DONE interrupt.

E. Writing 1 to mask the interrupt will not clear the corresponding bit in STATUS register.

   Accessing the mask register will not change the STATUS register bits. The bits in STATUS register may be cleared by writing 1 directly to the STATUS register (corresponding bits) or stopping the MMCCLK.

F. For a multi-block write, when a **Write-CRC-Error** occurs immediately, the current write process is stopped by the MMC/SDHC and the following data will not be sent to the card.

G. For a multi-block read, when a **Read-CRC-Error** occurs immediately, although the card will keep sending the data to MMC/SDHC, the MMC/SDHC will stop the current read process. No data will be filled into FIFO.

6 Proposed Handling Sequence for SDIO and Non-SDIO Interrupts

Because the interrupt handling of the MMC/SD host controller is different from other modules in the i.MX21 processor and the behavior of an SDIO interrupt is unique, it is necessary to handle the SDIO interrupt with other non-SDIO interrupts from a single interrupt source of one MMC/SDHC. This section provides a suggested sequence for the Interrupt Service Routine (ISR) or Interrupt Service Thread (IST). Figure 3 illustrates the recommended sequence.

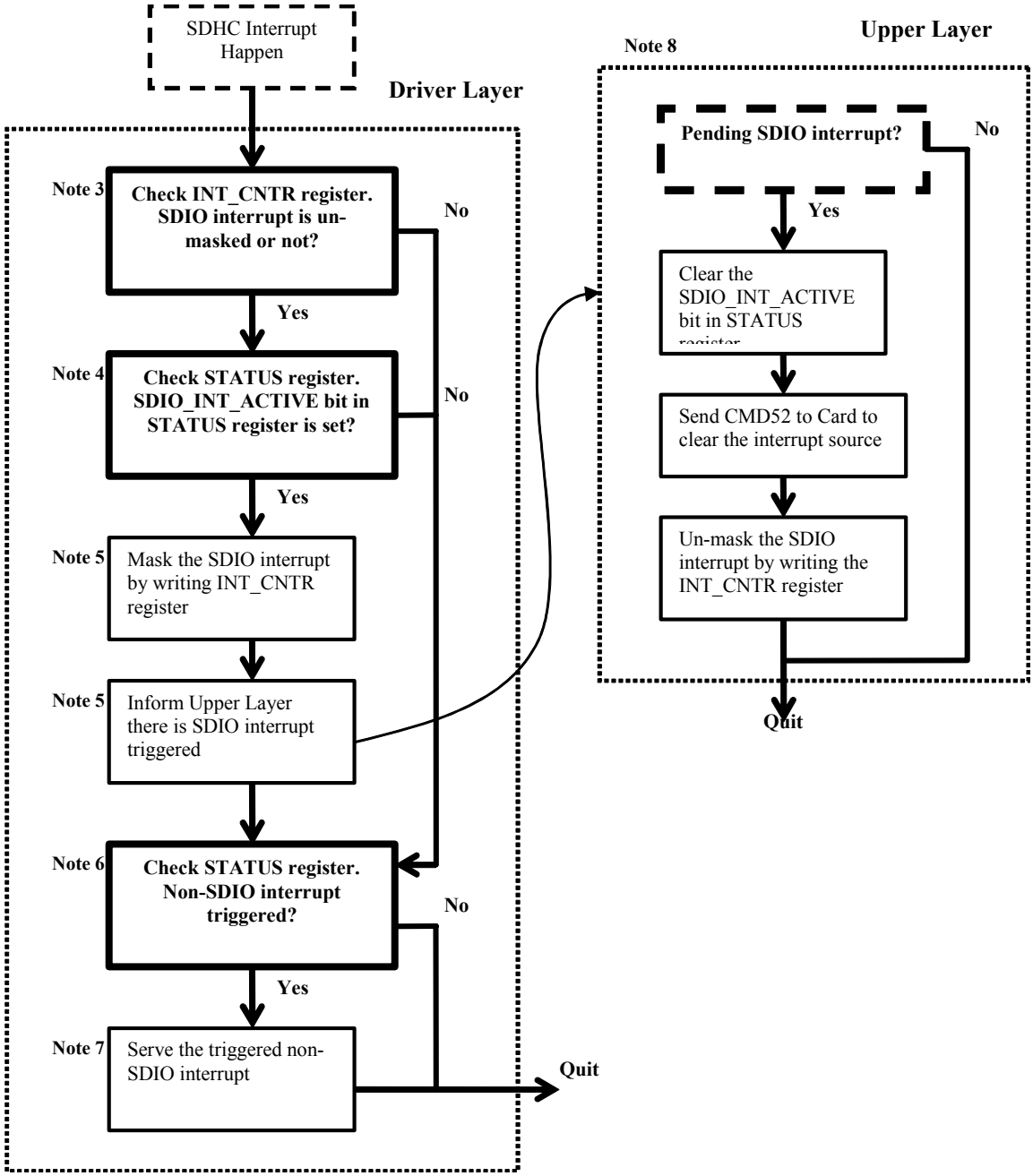

Figure 3. MMC/SDHC ISR/IST Flow

6.1 Driver Layer

1. When any MMC/SDHC interrupt occurs, the system goes into ISR or IST. The SDIO interrupt must be served first.
2. (**Note 3**)—Check whether the SDIO interrupt is masked or un-masked—that is, check the SDIO bit in INT_CNTR register. Note that this step must be performed before checking the STATUS register because the current SDIO_INT_ACTIVE bit might be a “fault” SDIO interrupt.

Reason: SDIO_INT_ACTIVE bit always reflects whether there is an SDIO interrupt request from the card regardless if the SDIO interrupt is masked or un-masked. Thus, an active SDIO_INT_ACTIVE bit in the STATUS register might previously have been reported to the upper layer (OS layer or Application layer). Therefore, the SDIO bit in INT_CNTR register must be checked first.

a) If YES, go to step 3.

b) If NO, go to step 4.

3. (**Note 4**)—Check whether SDIO_INT_ACTIVE bit in STATUS register is set or not.
   a) If YES (**Note 5**),
   — Mask the SDIO interrupt by writing INT_CNTR register, then
   — Inform upper layer there is a new SDIO interrupt,
   — Then, go to step 4
   b) If NO, go directly to step 4.
4. (**Note 6**)—Check the STATUS register when a non-SDIO interrupts is incoming. See Note A and Note B below.
   a) If YES, go to step 5.
   b) If NO, go to step 6.
5. (**Note 7**)—Serve the triggered non-SDIO interrupt.
6. Quit the ISR or IST.

Note A: Only one of END_CMD_RES, WRITE_OP_DONE, and READ_OP_DONE interrupts will occur simultaneously.

Note B: It is strongly recommend to use DMA to transfer data.
Reason: BUF_READY interrupt will occur with READ_OP_DONE interrupt at the same time, and BUF_READY interrupt will occur very near to WRITE_OP_DONE. This causes more concern with the BUF_READY interrupt.

6.2 Upper Layer

(**Note 8**) Upper layer means OS layer or Application layer.

If there is a pending SDIO interrupt to the upper layer and after the upper layer has processed to clear the interrupt source from the card. Most likely CMD52's will be sent to the card to clear the interrupt request from the card. Then:

a) Upper layer should clear the SDIO_INT_ACTIVE bit in STATUS register,

and

b) Upper layer should un-mask the SDIO interrupt by writing to the SDIO bit in INT_CNTR register.

7 MMC: Multiple MMC and Stream Support

7.1 Multiple MMC Support

For multiple MultiMediaCard support, during the Card Identification state, primarily for CMD02 and CMD03, the cards will be “wired-or” such that the cap-load of the bus will be large and the rising/fall time is not optimal. To solve this problem:

1. Bring MMCCLK low during the Card Identification stage. Reference value is 100 kHz for two MMC cards.

   Or,

2. Disable the i.MX21 interrupt pull-up resistor for the CMD line and use instead an external pull-up resistor. Reference value is 47 kohm for two MMC.

The “wire-or” case must not be in any state other than Card Identification. See reference [4] for more information.

NOTE

According to reference [4], for number of cards less than ten, the maximum rising/falling time is 10ns.

7.2 Stream Support

Stream Read and Stream Write in the MMC specification is not supported in the i.MX21 processor. Multi-Block Read and Multi-Block Write are recommended instead.

8 Card Detection

There are three approaches for the SD card detection as described in Table 5.

Table 5. Card Detection Approaches

Method	Approach	Brief Description	SD/SDIO Card Support	MMC Support
1	Mechanical	Insertion can be sensed by mechanical detecting the WP switch	Yes	Yes
2	Electrical	Insertion can be sensed using the pull-up resistor on DAT3	Yes	No
3	Software	Periodical attempts to initialize the card	Yes	Yes

The Mechanical approach is the most common approach and is independent of the MMC/SDHC. Only a card socket is used with a mechanical design and is connected to a GPIO pin of the i.MX21 processor, the card detection is made by an interrupt trigger from the GPIO pin.

The Software approach is a less common approach because of increased power consumption. This approach will not be discussed in this document.

The Electrical approach is MMC/SDHC dependent and is supported by the MMC/SD host controllers in the i.MX21 processor. When using this feature, the user must use care with the pull-up/pull-down resistor

connected to DAT3 line. Moreover, the following software implementations must occur: (See also Figure 4.)

a) The internal pull-up resistor must not be enabled when there is no card inserted.

b) There must be an external pull-down resistor connected to DAT3 and this pull-down resistor can be disconnected by software. The suggested value is 750 kohm.

c) For an SD memory card, after the card detection phase, ACMD42 must be issued to disconnect the pull-up resistor inside the card, which is connected to DAT3.

d) For an SDIO card, after the card detection phase, ACMD52 must be issued to clear the CD Disable bit in Bus Interface Control register (address: 0x08) in the CCCR to disconnect the pull-up resistor inside the card, which is connected to DAT3.

e) For a combination of cards, both items *c* and *d* must be implemented. Otherwise the card-internal pull-down resistor will not be disconnected.

f) After disconnect the card-internal resistor, then:

— Disconnect the external pull-down resistor to DAT3 line (Note 8 in Figure 4).

— Enable the i.MX21 internal pull-up resistor for DAT3 pin (Note 9 in Figure 4).

g) The card identification can be started.

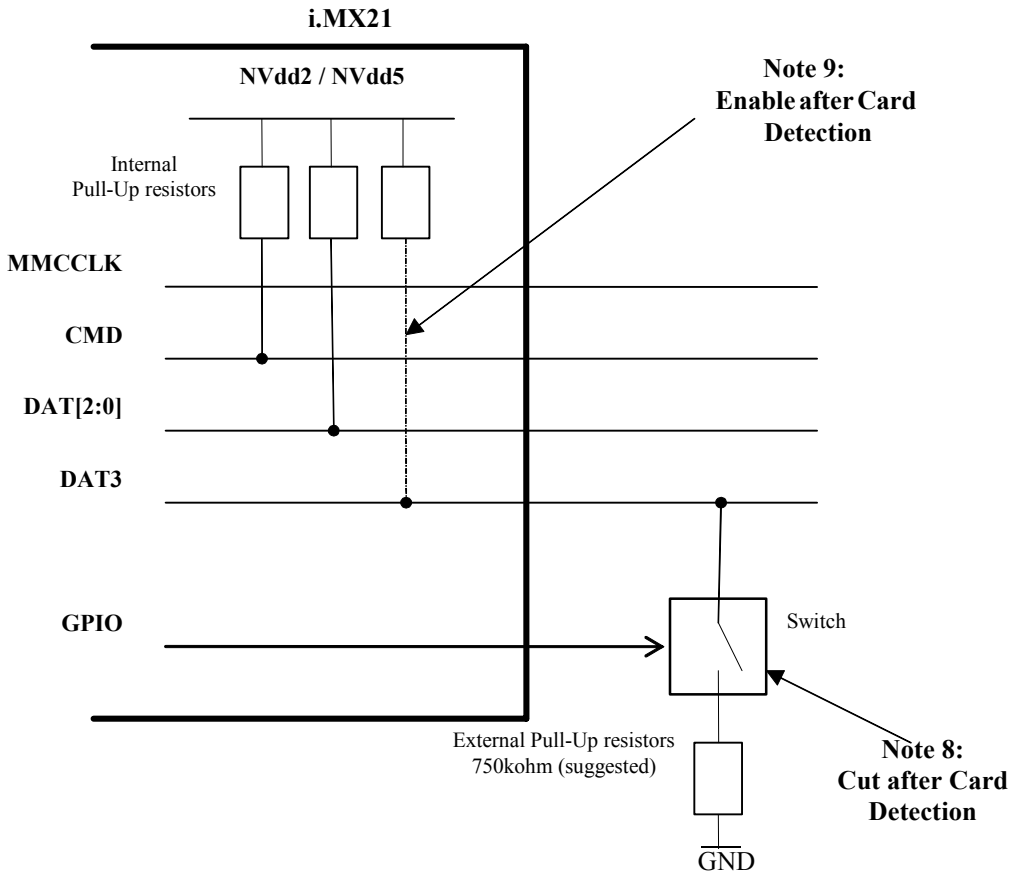

Figure 4. Pull-up / Pull-down Resistor Setting for Card Detection

9 Reference Documents

The following documents are helpful when used in conjunction with this application note.

[1]: *MC9328MX21 Reference Manual* (order number MC9328MX21RM/D)

[2]: *SD Memory Card Specifications*, Part 1 Physical Layer Specification Version 1.0

[3]: *SD Memory Card Specifications*, Part E1 Secure Digital Input/Output (SDIO) Card Specification, Version 1.00

[4]: *The MultiMediaCard System Specification,* Version 2.1

The Freescale manual is available on the Freescale Semiconductors Web site at http://www.freescale.com/imx. This documents may be downloaded directly from the Freescale Web site, or printed versions may be ordered.

Appendix A Reference Code)

This appendix provides the example code discussed within this application note.

Example 1. SDHC_Port_Init()

```
//==========================================================================
//       Function:         SDHC_Port_Init(uint8_t sdhc_no)
//==========================================================================
//       Description:      initialize the gpio port as the function of sdhc
//       Parameters:       sdhc_no, the number of SHDC, 1 or 2
//       Returns: success or not
//                         0: failed, 1: success
//       Remarks:
//==========================================================================
bool_t SDHC_Port_Init(uint8_t sdhc_no)
{
        //enable the GPIO module
        Module_Enable_0(GPIO_EN, ON);

        if (sdhc_no==1)
        {
                //set the function of gpio
                //SD1_CLK:        PE23     Pin 116
                //SD1_CMD:        PE22     Pin 117  (internal pull-up)
                //SD1_D3:         PE21     Pin 118  (internal pull-up/external pull-low)
                //SD1_D2:         PE20     Pin 119  (internal pull-up)
                //SD1_D1:         PE19     Pin 120  (internal pull-up)
                //SD1_D0:         PE18     Pin 121  (internal pull-up)

                //set the port using multiplexed function (set 0)
                *(p_uint32_t) GPIOE_GIUS   &= ~((uint32_t)(PIN23 | PIN22 | PIN21 | PIN20 | PIN19 | PIN18));
                //set the port using primary function (set 0)
                *(p_uint32_t) GPIOE_GPR            &= ~((uint32_t)(PIN23 | PIN22 | PIN21 | PIN20 | PIN19 | PIN18));
                //enable port pull-high resistors
                //CMD, DAT2, DAT1, DAT0 need to be internal pull-up
                //MMCCLK does not need
                //according to SD/MMC spec., for card detection, DAT3 need pull-low.
                *(p_uint32_t) GPIOE_PUEN   |= (PIN22 & PIN20 & PIN19 & PIN18);//use the interrnal pull-up but use the
external pull-up
```

```
                *(p_uint32_t) GPIOE_PUEN   &= ~(PIN23 | PIN21);

                //set the I/O driving strength of the GPIO pad
                //set the DS_SLOW1 field (bit 2:0) in DSCR1 register.  this field is exactly for Pin 116 to Pin 121
                *(p_uint32_t) SYS_DSCR1            = (*(p_uint32_t) SYS_DSCR1 & ~(MASK_3_BIT << 0)) | (IO_STR_2MA <<
0);     //2mA
//              *(p_uint32_t) SYS_DSCR1            = (*(p_uint32_t) SYS_DSCR1 & ~(MASK_3_BIT << 0)) | (IO_STR_4MA <<
0);     //4mA
//              *(p_uint32_t) SYS_DSCR1            = (*(p_uint32_t) SYS_DSCR1 & ~(MASK_3_BIT << 0)) | (IO_STR_8MA <<
0);     //8mA
//              *(p_uint32_t) SYS_DSCR1            = (*(p_uint32_t) SYS_DSCR1 & ~(MASK_3_BIT << 0)) | (IO_STR_12MA <<
0);     //12mA
        }
        else    //sdhc_no==2
        {
                //set the function of gpio
                //SD2_CLK:        PB9     PIN 254
                //SD2_CMD:        PB8     PIN 255  (internal pull-up)
                //SD2_D3:         PB7     PIN 256  (external pull-low
                //SD2_D2:         PB6     PIN 257  (internal pull-up)
                //SD2_D1:         PB5     PIN 258  (internal pull-up)
                //SD2_D0:         PB4     PIN 259  (internal pull-up)

                //set the port using multiplexed function (set 0)
                *(p_uint32_t) GPIOB_GIUS   &= ~((uint32_t)(PIN09 | PIN08 | PIN07 | PIN06 | PIN05 | PIN04));
                //set the port using primary function (set 0)
                *(p_uint32_t) GPIOB_GPR            &= ~((uint32_t)(PIN09 | PIN08 | PIN07 | PIN06 | PIN05 | PIN04));
                //enable port pull-high resistors
                //CMD, DAT2, DAT1, DAT0 need to be internal pull-up
                //MMCCLK does not need
                //according to SD/MMC spec., for card detection, DAT3 need pull-low.
                *(p_uint32_t) GPIOB_PUEN   |= (PIN08 & PIN06 & PIN05 & PIN04);//use the interrnal pull-up but use the
external pull-up
                *(p_uint32_t) GPIOB_PUEN   &= ~(PIN09 | PIN07);

                //set the I/O driving strength of the GPIO pad
                //set the DS_SLOW7 field (bit 21:19) in DSCR1 register.  this field is exactly for Pin 254 to Pin 259
//              *(p_uint32_t) SYS_DSCR1            = (*(p_uint32_t) SYS_DSCR1 & ~(MASK_3_BIT << 19)) | (IO_STR_2MA <<
19);     //2mA
```

```
//                *(p_uint32_t) SYS_DSCR1             = (*(p_uint32_t) SYS_DSCR1 & ~(MASK_3_BIT << 19)) | (IO_STR_4MA <<
19);     //4mA
//                *(p_uint32_t) SYS_DSCR1             = (*(p_uint32_t) SYS_DSCR1 & ~(MASK_3_BIT << 19)) | (IO_STR_8MA <<
19);     //8mA
                  *(p_uint32_t) SYS_DSCR1             = (*(p_uint32_t) SYS_DSCR1 & ~(MASK_3_BIT << 19)) | (IO_STR_12MA
<< 19);  //12mA
         }

         return 1;
}
```

Example 2. SDHC_Set_Clk_Rate()

```
//=============================================================================
//Function:SDHC_Set_Clk_Rate(uint8_t sdhc_no, uint16_t prescaler, uint8_t divider)
//=============================================================================
//Description:             set the CLK_RATE register value for different value of
//                                   CLK_PRESCALER field and CLK_DIVIDER field
//Parameters:                              sdhc_no, the number of SHDC, 1 or 2
//                                  prescaler, the value of CLK_PRESCALER field
//                                   prescaler, the value of CLK_DIVIDER field
//Returns:success or not
//                                                        0: failed, 1: success
//Remarks:
//=============================================================================
bool_t SDHC_Set_Clk_Rate(uint8_t sdhc_no, uint16_t prescaler, uint8_t divider)
{
uint16_t i;
uint8_t                                                  prescaler_bit= 0;

if (divider==0)
{
         printf("divider (0x%x) is invalid!!\n", divider);
         return 0;
}
```

```
        if (prescaler!=0)
        {
                for (i=0; i<12; i++)
                {
                                                        if (prescaler & (0x0001<<i))
                                                                                {
                                                                        prescaler_bit++;
                                                                                }
                }

                if (prescaler_bit!=1)
                {
                                printf("prescaler (0x%x) is invalid!!\n", prescaler);
                                                                        return 0;
                }
        }

        debugprintf("\tCLK_PRESCALER is 0x%x, CLK_20M = CLK_DIV / %d\n", prescaler, prescaler*2);
        debugprintf("\tCLK_DIVIDER is 0x%x, CLK_DIV = PERCLK2 / %d\n", divider, divider+1);

        *(p_uint32_t)SDHC_reg[sdhc_no-1][CLK_RATE]= (uint32_t)(prescaler<<4) | (uint32_t)divider;

        debugprintf("setting CLK_RATE register value for SDHC-%d complete.\n", sdhc_no);

        return 1;
}
```

Example 3. Module_Enable_0()

```
//=============================================================================
//      Function:         void Module_Enable_0(uint32_t module_bit, bool_t on_off)
//=============================================================================
//      Description:      do the module enable according to the PCCR0 register
//      Parameters:       module_bit, the module-bit defined the the PCCR0 register
//                        on_off, 0 for off and 1 for on
```

```
//    Returns: success or not
//                0: failed, 1: success
//    Remarks:
//=============================================================================
void Module_Enable_0(uint32_t module_bit, bool_t on_off)
{
    if (on_off)      //if on
    {
        *(p_int32_t)CRM_PCCR0    |=    module_bit;
    }
    else             //if off
    {
        *(p_int32_t)CRM_PCCR0    &=    ~module_bit;
    }
}
```

NOTES

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

***For Literature Requests Only*:**
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-521-6274 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN2906
Rev. 0
03/2005

freescale™
semiconductor