Using SDCard and SDIO with the Intel® PXA250 Applications Processor MMC Controller

Application Note

February, 2002

Order Number: 278533-001

Contents

Figures

Tables

Revision History

Date	Revision	Description
October 2001	0.5	Initial release.

1.0 Introduction

The MultiMediaCard (MMC) Controller on the Intel® PXA250 Applications Processor can communicate with either:

- A MultiMediaCard (MMC)
- A Secure Digital (SD) Memory Card
- A Secure Digital I/O (SDIO) card

Since the MMC controller was originally intended to support MMC cards only, this application note describes how to use the SD Memory Card and SDIO card and some of the pitfalls in doing so. This application note gives a detailed description of the following:

- The difference between MMC, SD Memory Cards, and SDIO cards
- How to connect the SD Memory Card or SDIO card to the PXA250 applications processor
- How to reset, initialize, and communicate with the SD Memory Card or SDIO card in MMC and SPI mode
- Describe performance issues of using the SD Memory Card and SDIO card with the MMC Controller

This application note assumes that the reader has a general knowledge of MMC, SD Memory Cards, SDIO cards, and the PXA250 applications processor. If more information is required, refer to the respective specification documents; ***The MultiMediaCard System Specification***, ***SD Memory Card Specifications***, ***SDIO Card Specification*** and the ***Intel® PXA250 and PXA210 Applications Processors Developer's Manual***. In addition, this application note applies to both the PXA250 applications processor and the PXA210 applications processor, however, for brevity, only the PXA250 applications processor is mentioned in the text.

All of the connections and code for the SD Memory Card and SDIO card are based on the respective specifications. The information for the SD Memory Card has been verified on the Lubbock evaluation platform. The information on SDIO is presented from a theoretical perspective based on the specification.

2.0 Differences from MMC

The main differences between the MMC, SD Memory Card, and SDIO are shown in Table 1. This table is taken from the SD Memory Card Physical Layer Specification with additional columns for SDIO and Intel® PXA250 applications processor support.

Table 1. Feature Differences Between MMC, SD Memory Card, and SDIO

Feature	MMC	SD Memory Card	SDIO	PXA250 Applications Processor Supports
Bus width	1 bit only	1 bit or 4 bits	1 bit or 4 bits	1 bit
System bus organization (multiple cards connection)	Bus topology	Star topology	Star Topology	Bus topology
Initialization commands	CMD0 CMD1 CMD2 CMD3	CMD0 CMD55 ACMD41 CMD2 CMD3	CMD0 CMD5 CMD55 ACMD41 CMD2 CMD3	Any combination of commands is supported
Maximum clock rate	20MHz	25MHz	25MHz	20MHz
Copyright protection	not supported	supported	supported	supported
Write protect switch	not supported	supported	supported	support through GPIO[1]
Feature of pin #1	Not connected	Internal pull-up resistor	Internal pull-up resistor	not supported
CSD structure		Different from MMC	Different from MMC and SD Memory Card	Supported as response type 2
CID structure		Different from MMC	Different from MMC and SD Memory Card	Supported as response type 2
SPI R/W multiple block	not supported	supported	supported	supported
Stream R/W mode	supported (optional)	not supported	not supported	supported
I/O mode	supported (optional)	not supported	supported	supported

1. GPIO – General Purpose Input/output

The MMC controller on the PXA250 applications processor has one dedicated data signal: MMDAT; and one dedicated command signal: MMCMD. This limits the controller to supporting only 1-bit devices with a bus topology. The star topology required to use multiple SD Memory Cards and SDIO cards requires a separate data and command signal for each card. Hence, only one SD Memory Card or SDIO card can be used with the controller unless the card is placed into SPI mode. In this case, the controller supports two cards with the chip select signal: MMCCS[1:0].

The MMC controller can send out any of the 64 possible commands supported by the various card types. All cards can be initialized using the controller, but each card requires a different set of commands for the initialization sequence (refer to Section 4.1, “Initializing the SD Memory Card and SDIO Card” on page 11.) Also, the copyright protection in SD Memory Cards and SDIO cards is supported through a special set of commands called application commands.

To read the various responses from each of the cards using the MMC controller, program the correct response size into the MMC_CMDAT[RESPONSE_FORMAT] bits. This enables software to read any of the card registers, including the CID and CSD.

There are several circuit design issues that must be addressed when using the MMC controller with SD Memory Cards and SDIO cards. Some of these include pull-up resistor values, pin 1 features, the interrupt signal, the write protect signal, and the card detect signal. All design issues are described in Section 3.0, “Circuit Design Issues” .

3.0 Circuit Design Issues

To create a design with the PXA250 applications processor that uses an MMC, SD Memory Card, or SDIO card various issues must be addressed. You must configure the correct connections from the processor to the card including interrupt signals and pull-up resistors for the appropriate operation. Also, use either an MMC socket or an SD Memory Card socket. These issues are discussed in the following paragraphs.

3.1 Signal Descriptions

The main consideration when connecting a card to the MMC controller is the available signals. As a starting point, consider the pins specified for the MMC card as shown in Table 2. While in MMC mode, there are three relevant signals: clock, command, and data. If you are using the SPI mode, a chip select is also required. This interface is completely supported by the MMC Controller.

Table 2. MultiMediaCard Pin Description

Pin No.	MultiMediaCard Mode		SPI Mode		MMC Controller Signals
	Name	Description	Name	Description	Name
1	RSV	Reserved	CS	Chip select	MMCCS
2	CMD	Command/Response	DI	Data in	MMCMD
3	VSS	Ground	VSS	Ground	
4	VDD	Supply Voltage	VDD	Supply voltage	
5	CLK	Clock	SCLK	Clock	MMCLK
6	VSS2	Ground	VSS2	Ground	
7	DAT	Data	DO	Data out	MMDAT

The signal differences between an MMC card and either an SD Memory Card or SDIO card is three more data signals. An SDIO card also has an interrupt signal that can be used in the narrow or SPI modes. The pin descriptions for the two types of cards are shown in Table 3 and Table 4.

Table 3. SD Memory Card Pin Description (Sheet 1 of 2)

Pin No.	SD Wide (4-bit) Mode		SD Narrow (1-bit) Mode		SPI Mode	
	Name	Description	Name	Description	Name	Description
1	CD/DAT3	Card detect/Data[3]	N/C	Not used	CS	Chip select
2	CMD	Command/Response	CMD	Command/Response	DI	Data in
3	VSS	Ground	VSS	Ground	VSS	Ground
4	VDD	Supply Voltage	VDD	Supply Voltage	VDD	Supply voltage
5	CLK	Clock	CLK	Clock	SCLK	Clock
6	VSS2	Ground	VSS2	Ground	VSS2	Ground

Table 3. SD Memory Card Pin Description (Sheet 2 of 2)

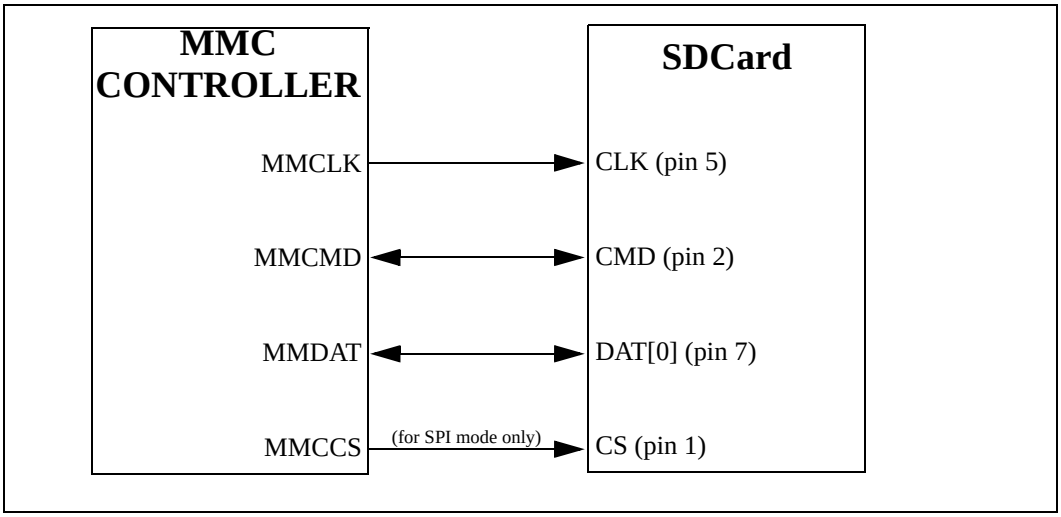
Pin No.	SD Wide (4-bit) Mode		SD Narrow (1-bit) Mode		SPI Mode	
	Name	Description	Name	Description	Name	Description
7	DAT[0]	Data[0]	DATA	Data	DO	Data out
8	DAT[1]	Data[1]	N/C	Not used	RSV	Reserved
9	DAT[2]	Data[2]	N/C	Not used	RSV	Reserved

Table 4. SDIO Pin Description

Pin No.	SD Wide (4-bit) Mode		SD Narrow (1-bit) Mode		SPI Mode	
	Name	Description	Name	Description	Name	Description
1	CD/DAT3	Card detect/Data[3]	N/C	Not used	CS	Chip select
2	CMD	Command/Response	CMD	Command/Response	DI	Data in
3	VSS	Ground	VSS	Ground	VSS	Ground
4	VDD	Supply Voltage	VDD	Supply Voltage	VDD	Supply voltage
5	CLK	Clock	CLK	Clock	SCLK	Clock
6	VSS2	Ground	VSS2	Ground	VSS2	Ground
7	DAT[0]	Data[0]	DATA	Data	DO	Data out
8	DAT[1]	Data[1]	IRQ	Interrupt	IRQ	Interrupt
9	DAT[2]	Data[2]	RW	Read wait (optional)	RSV	Reserved

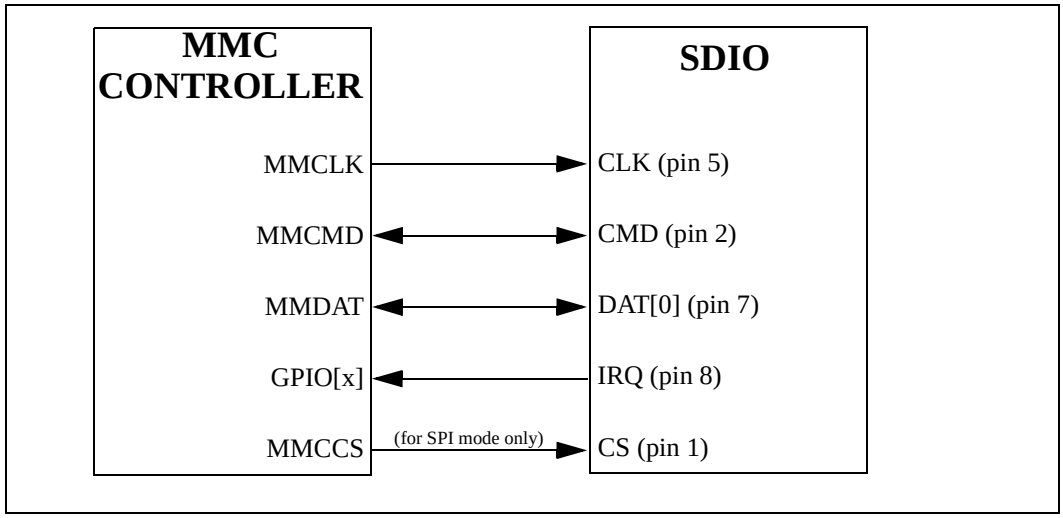
The connections from these cards to the MMC controller is shown Figure 1 and Figure 2. Since the MMC controller has only one data signal, use DAT[0] on the SD Memory Card and SDIO card. The other data signals on the card must be pulled up according to the appropriate specification.

Figure 1. SD Memory Card Connections



One difference to note between the SD Memory Card and the SDIO card is the IRQ signal, which is available on the SDIO card. The IRQ signal is DAT[1] on the SDIO card and only functions as an interrupt in the narrow and SPI modes of operation. Connect the IRQ signal to one of the general purpose input/output (GPIO) signals on the PXA250 applications processor and configure the GPIO to be an interrupt.

Figure 2. SDIO Connections



3.2 Pull-up resistor values

The pull-up resistors required by the SD Memory Card and SDIO card are the same and are shown in Table 5. The cards also have an internal pull-up resistor on pin 1, DAT[3], that varies from 10kΩ to 90 kΩ. Refer to the manufacturer's card specifications for the precise value.

Table 5. Required Pull-up Resistors

Name	Value	Description
RDAT	10 kΩ - 100 kΩ	Pull-up for the data signals, DAT[3:0]. All pull-ups are required even though only a single data signal will be used.
RCMD	10 kΩ - 100 kΩ	Pull-up for the command signal, CMD.

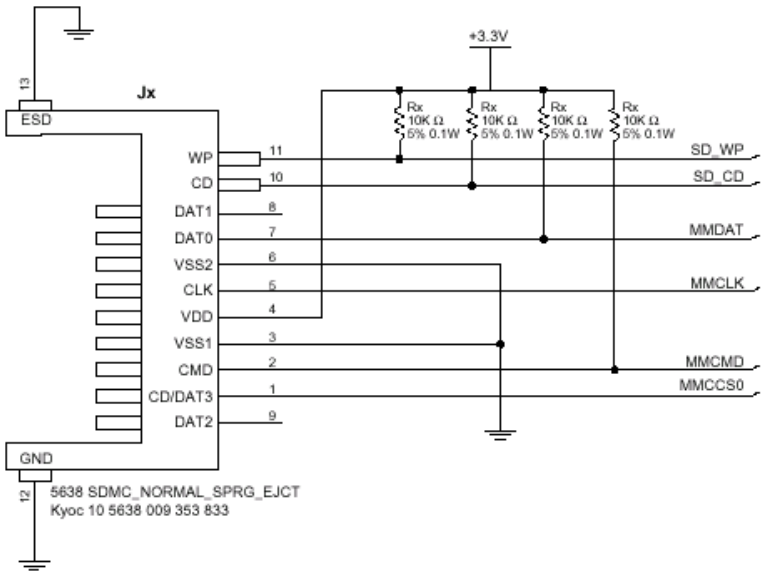
When designing a system that use both an MMC card and an SD Memory Card or an SDIO card, the pull-up on the data signal should be 50 kΩ to 100 kΩ to match the specification for the MMC card.

3.3 Sockets

During the design phase, the decision between an MMC or SD Memory Card socket must be made. An MMC socket only supports an MMC card, while an SD Memory Card socket supports an SD Memory Card, an SDIO card, or an MMC card. The only difference between the two socket types is the number of pins. The MMC socket has 7 pins and the SD Memory Card socket has 9 pins to accommodate the extra data signals. The SD Memory Card socket also has two more pins, usually on the side or back of the socket. These pins are the card detect (CD) and write protect (WP) signals. These signals are mechanical switches inside the socket. One signal indicates the presence

of a card (CD) and the other write protects the card (WP). Figure 3 shows an SD Memory Card socket in a typical system. The CD and WP signals can be connected to a complex programmable logic device (CPLD) or connected to spare GPIOs on the PXA250 applications processor to generate interrupts.

Figure 3. Typical SD Memory Card Socket



3.4 Using the PXA250 Applications Processor's GPIOs

The CD and WP pins on the socket and the IRQ signal on an SDIO card can be connected to spare GPIO signals on the PXA250 applications processor. Configure the CD and IRQ GPIOs so a rising edge causes an interrupt to the applications processor. It is not necessary for the WP signal to cause an interrupt, so software only needs to read the level of the GPIO to determine whether the card is write protected or not. Figure 4 shows the code necessary to configure a GPIO as an interrupt.

Figure 4. Code to Configure a GPIO as an Interrupt

```
//Set up GPIO[22] to interrupt on rising edge
*ICMR = *ICMR & 0xFFFFFBFF;//disable interrupts on GPIO[80:2]
*ICLR = *ICLR & 0xFFFFFBFF;//set GPIO interrupts to be IRQ
*GRERx = *GRERx | 0x00400000;// enable rising edge detect on GPIO[22]
*GFERx = *GFERx & 0xFFBFFFFF;// disable falling edge detect on GPIO[22]
*GPDRx = *GPDRx | 0x00000140;// Set GPIO[22] to be input
*ICMR = *ICMR | 0x00000400;//enable interrupts on GPIO[80:2]
```

4.0 Software Issues

There are two main software issues when using an SD Memory Card or an SDIO card with the MMC controller:

- Initializing the card
- Reading the responses

Depending on the type of card being used, the command initialization sequence will vary. This is covered in Section 4.1, “Initializing the SD Memory Card and SDIO Card” . The responses also vary across card types. The MMC controller can read all of the response types, but some issues with response sizes are addressed in Section 4.2, “Response Formats” .

4.1 Initializing the SD Memory Card and SDIO Card

The first steps to initialize the SD Memory Card and SDIO card are the same as the first steps to initialize the MMC card. The software must configure the GPIOs multiplexed with the MMCLK and MMCCS0 (if using SPI mode) to their alternate functions. The GPIOs configuration code is shown in Figure 5. The host must then send a contiguous stream of 1’s and wait for the power supply to ramp up then wait 1 ms followed by 74 clocks. To accomplish this with the MMC controller, wait 1 ms using one of the PXA250 applications processor’s timers, then set the MMC_CMDAT[INIT] bit before sending out the first command. Setting the MMC_CMDAT[INIT] bit causes the MMC controller to hold the MMCMD signal high while toggling the clock 80 times before sending out the command in the MMC_CMD register.

Figure 5. Code to Use GPIO Alternate Functions for MMC Controller

```
//Set up alternate function GPIO
*GRERx = *GRERx & 0xFFFFFEBF;// disable rising edge detect on MMCLK and MMCCS0
*GFERx = *GFERx & 0xFFFFFEBF;// disable falling edge detect on MMCLK and
                             // MMCCS0
*GPSRx = *GPSRx | 0x00000140;// Set MMCLK and MMCCS0 high before configuring
                             // output
*GPDRx = *GPDRx | 0x00000140;// Set MMCLK and MMCCS0 to outputs on
                             // GPIO 6 and 8
*GAFR0x = (*GAFR0x & 0xFFFDDFFF) | 0x00011000;//Set GPIO6 and GPIO8 to
                                              // alternate function 1
```

After configuring the GPIOs for use with the MMC controller and initializing the card through the power up sequence, certain commands need to be sent to the card to finish the initialization procedure. Table 6 shows a list of the commands sent to each card type that completes the power up initialization. The code for the entire setup sequence is shown in Appendix A, “SD Memory Card Initialization Code”.

Table 6. Initialization Commands

MMC	SD Memory Card	SDIO
CMD0 CMD1 CMD2 CMD3	CMD0 CMD55 ACMD41 CMD2 CMD3	CMD0 CMD5 CMD55 ACMD41 CMD2 CMD3

4.2 Response Formats

The MMC controller handles responses based on the size of the response. The MMC has responses R1, R1b, R2, R3, R4, and R5. The SD Memory Card and the SDIO card have responses R1, R1b, R2, R3, and R6. The R1, R1b, R2, and R3 response sizes are the same regardless of the card type being used. Hence, the settings for the MMC_CMD[RESPONSE_FORMAT] bits are the same as described in the ***Intel® PXA250 and PXA210 Applications Processors Developer's Manual***. Since response R6 is 48 bits, the MMC_CMDAT[RESPONSE_FORMAT] bits can be set to 0b01 to read the proper response from the card. Table 7 shows all responses, their sizes, associated card types, and MMC_CMDAT[RESPONSE_FORMAT] bit values required.

Table 7. Response Types and Sizes

Response	Size	Card	RESPONSE_FORMAT Bits
R1	48 bit	MMC, SD, SDIO	0b01
R1b	48 bit	MMC, SD, SDIO	0b01
R2	136 bit	MMC, SD, SDIO	0b10
R3	48 bit	MMC, SD, SDIO	0b11
R4	48 bit	MMC	0b01
R5	48 bit	MMC	0b01
R6	48 bit	SD, SDIO	0b01

Appendix A SD Memory Card Initialization Code

```
// Copyright (C) 2000.  Intel Corporation.
// File:     SDCard.c

#include <stdio.h>
#include <math.h>

extern char pstring[100];

extern void printlcd(const char *cntrl_string);
extern void Delay(int);
extern lcd_row, lcd_col;

int SDCard_test(void)
{
    int SDPassed, response[16], rxdata[512], x, y, z;
    int SD_Address;
    int READ_BL_LEN;
    int WRITE_BL_LEN_U, WRITE_BL_LEN_L, WRITE_BL_LEN;
    int C_SIZE_U, C_SIZE_L, C_SIZE;
    int C_SIZE_MULT;
    int memory_capacity;

    //Set up alternate function GPIO
    *GRERx = *GRERx & 0xFFFFFEBF;// disable rising edge detect on MMCLK and MMCCS0
    *GFERx = *GFERx & 0xFFFFFEBF;// disable falling edge detect on MMCLK and MMCCS0
    *GPSRx = *GPSRx | 0x00000140;// Set MMCLK and MMCCS0 high before configuring
                                 //output
    *GPDRx = *GPDRx | 0x00000140;// Set MMCLK and MMCCS0 to outputs on GPIO 6 and 8
    *GAFR0x = (*GAFR0x & 0xFFFDDFFF) | 0x00011000;//Set GPIO6 and GPIO8 to
                                                  //alternate function 1

    //initialize card
    Delay(1000);//wait 1ms per MMC specification

    //CMD00 GO_IDLE_STATE

    *MMCSTRPCL = 0x00000001;//stop clock
    while((*MMCSTAT & 0x00000100) == 0x00000100); //wait for clock to stop
    *MMCRESTO  = 0x0000007f;//set response timeout to max value (64 clocks)
    *MMCCLKRT  = 0x00000006;//set MMCLK rate to 312.5kHz
    *MMCCMD    = 0x00000000;//CMD02 index ALL_SEND_CID
    *MMCARGH   = 0x00000000;//stuff bits for argument
    *MMCARGL   = 0x00000000;//stuff bits for argument
    *MMCCMDAT  = 0x00000000;//expect no response
    *MMCSTRPCL = 0x00000002;//start clock

    while((*MMCSTAT & 0x00002000) == 0x00000000);//wait for end_cmd_res


    //CMD55 APP_CMD
    //This command must be sent every time before an application command
    *MMCSTRPCL = 0x00000001;//stop clock
    while((*MMCSTAT & 0x00000100) == 0x00000100); //wait for clock to stop
    *MMCCMD    = 0x00000037;//CMD55 index APP_CMD
    *MMCARGH   = 0x00000000;//relative card address 0x0
    *MMCARGL   = 0x00000000;//stuff bits
    *MMCCMDAT  = 0x00000001;//expect response 1
    *MMCSTRPCL = 0x00000002;//start clock

    while((*MMCSTAT & 0x00002000) == 0x00000000);//wait for end_cmd_res
    //read response FIFO
    for(x=0;x<3;x++)
    {
```

```
    response[x] = *MMCRES & 0x0000ffff;

}

//ACMD41
*MMCSTRPCL = 0x00000001;//stop clock
while((*MMCSTAT & 0x00000100) == 0x00000100); //wait for clock to stop

*MMCCLKRT  = 0x00000006;//set MMCLK rate to 312.5kHz
*MMCCMD    = 0x00000029;//ACMD41 index SD_APP_OP_COND
*MMCARGH   = 0x00000020;//set voltage limit of system in command argument
*MMCARGL   = 0x00000000;
*MMCCMDAT  = 0x00000043;//set init bit for initial 80 clocks, expect response 3
*MMCSTRPCL = 0x00000002;//start clock

while((*MMCSTAT & 0x00002000) == 0x00000000);//wait for end_cmd_res
//read response FIFO
for(x=0;x<3;x++)
{
    response[x] = *MMCRES & 0x0000ffff;
}

y=1;

while(response[0] != 0x3f80)//continue doing ACMD1 until busy bit in response
                            //is set
{
    //CMD55 APP_CMD
    *MMCSTRPCL = 0x00000001;//stop clock
    while((*MMCSTAT & 0x00000100) == 0x00000100); //wait for clock to stop
    *MMCCMD    = 0x00000037;//CMD55 index APP_CMD
    *MMCARGH   = 0x00000000;//relative card address 0x0
    *MMCARGL   = 0x00000000;//stuff bits
    *MMCCMDAT  = 0x00000001;//expect response 1
    *MMCSTRPCL = 0x00000002;//start clock

    while((*MMCSTAT & 0x00002000) == 0x00000000);//wait for end_cmd_res
    //read response FIFO
    for(x=0;x<3;x++)
    {
        response[x] = *MMCRES & 0x0000ffff;
    }

    //ACMD41
    *MMCSTRPCL = 0x00000001;//stop clock
    while((*MMCSTAT & 0x00000100) == 0x00000100); //wait for clock to stop
    *MMCCMD    = 0x00000029;//ACMD41 index SD_APP_SEND_OP_COND
    *MMCARGH   = 0x00000020;//set voltage limit of system in command argument
    *MMCARGL   = 0x00000000;
    *MMCCMDAT  = 0x00000003;//expect response 3
    *MMCSTRPCL = 0x00000002;//start clock

    while((*MMCSTAT & 0x00002000) == 0x00000000);//wait for end_cmd_res
    //read response FIFO
    for(x=0;x<3;x++)
    {
        response[x] = *MMCRES & 0x0000ffff;
    }
}

//CMD02 ALL_SEND_CID

*MMCSTRPCL = 0x00000001;//stop clock
while((*MMCSTAT & 0x00000100) == 0x00000100); //wait for clock to stop
*MMCCMD    = 0x00000002;//CMD02 index ALL_SEND_CID
*MMCARGH   = 0x00000000;//stuff bits for argument
*MMCARGL   = 0x00000000;//stuff bits for argument
```

```
*MMCCMDAT  = 0x00000002;//expect response 2
*MMCSTRPCL = 0x00000002;//start clock

while((*MMCSTAT & 0x00002000) == 0x00000000);//wait for end_cmd_res
//read response FIFO
for(x=0;x<8;x++)
{
    response[x] = *MMCRES & 0x0000ffff;
}

//CMD03 SET_RELATIVE_ADDR
*MMCSTRPCL = 0x00000001;//stop clock
while((*MMCSTAT & 0x00000100) == 0x00000100); //wait for clock to stop
*MMCCMD    = 0x00000003;//CMD03 index SET_RELATIVE_ADDR
*MMCARGH   = 0x00000000;//stuff bits
*MMCARGL   = 0x00000000;
*MMCCMDAT  = 0x00000001;//expect response 1
*MMCSTRPCL = 0x00000002;//start clock

while((*MMCSTAT & 0x00002000) == 0x00000000);//wait for end_cmd_res
//read response FIFO
for(x=0;x<3;x++)
{
    response[x] = *MMCRES & 0x0000ffff;
}

//The address of an SDCard is automatically assigned and returned in the
//response of CMD03
//The following gets the address from the response and puts it into a variable
SD_Address = ((response[0]&0x000000ff)<<8)|((response[1]&0x0000ff00)>>8);
```


ARM
POWERED