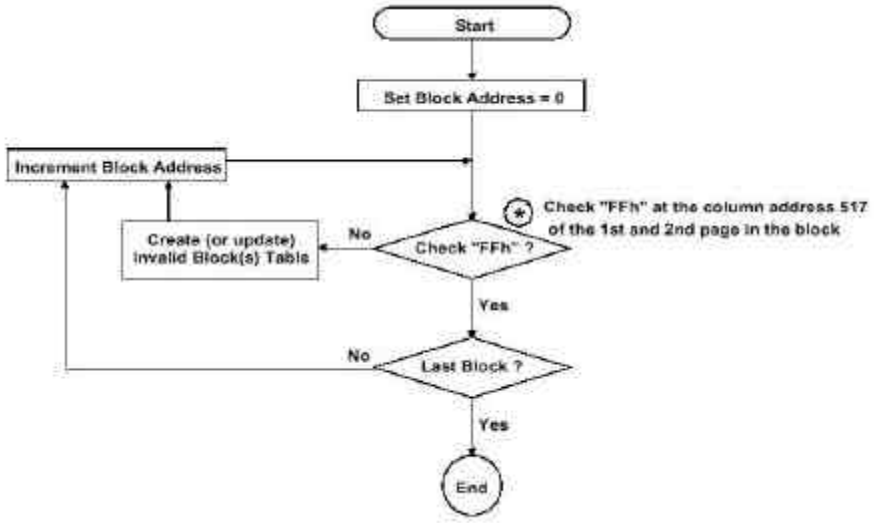
3. NAND FLASH memory / SMC(Smart Media Card) 블럭의 설계

FLASH memory는 휴대형 디지털장치의 저장용 device로 널리 사용되고 있다. 가격도 다른 portable card보다 저렴한 편이며 8bit or 16bit parallel 전송이므로 속도도 빠르다. 이 장에선 이러한 FLASH형태의 저장장치를 control하는 블록을 설계한다. 기본구조는 control register와 data register의 기본구조로 이루어지며 블록단위의 fast read/write는 DMA 블럭에서 담당하도록 구성되어 있다. 설계의 simple한 구조로 control register는 바로 FLASH pin에 물려있으며 read/write strobe 출력 및 FLASH 상태 비트를 받아들인다.

FLASH memory의 컨트롤은 command 와 address를 준 다음 데이터를 read/write하는 단순구조로 되어있다. 그냥 단순히 SRAM이나 DRAM사용하듯이 사용하면 된다. 실제 필드에서는 사용하기 전에 invalid block을 check 한후 사용하는 것이 좋다. 다음은 invalid block을 check하는 flow를 나타내어 보았다.



[그림4-7] FLASH invalid block check diagram

여기서 FLASH control register를 먼저 살펴보면 다음과 같다.

■ FLASH control #0 REGISTER : D9h (Byte access SFR)

7(I/O)	6(I/O)	5(I/O)	4(I/O)	3(I/O)	2(I/O)	1(I/O)	0(I/O)
R/Bb	CEb	CLE	ALE	M1	M0	WEb	REb

RESET VALUE : 11000011

각 bit별 기능은 다음과 같다.

R/Bb : READY/BUSYb - read flag

CEb : FLASH chip selection bit

CLE : FLASH command latch enable

ALE : FLASH address latch enable

M1,M0 : flash control mode selection이다. 기능별로 크게 4가지로 구분.

mode 0, [0,0] : address/command latch write

mode 1 [0,1] : address/command latch read (ex. status reading)

mode 2 [1,0] : sequential write(programming) 528 byte

mode 3 [1,1] : sequential read 528 byte

WEb : write strobe

REb : read strobe

■ FLASH control #1 REGISTER : DAh (Byte access SFR)

7(I/O)	6(I/O)	5(I/O)	4(I/O)	3(I/O)	2(I/O)	1(I/O)	0(I/O)
done528	done528cear						SELECT FLASH

RESET VALUE : 11000011

done528 (output) : 528byte trasfer to DMA done flag이다. 이bit로 인해 DMA address/WRB/REB 신호가 중단됨을 나타내어준다. 이 비트를 보고 그다음 don528clear를 사용하여 다시 DMA

동작을 수행 시킨다.

done528clear(input) : making done528 signal to LOW signal

done528clear bit가 High->LOW일 때 그후 DMA WRB/REB가 동작 하며 DMA_ADDR가 자동 증가된다.

```
예제코드) sampe FLASH 1Page reading start ocde
        mov FLASHCON1,#40h    ; done528clear on => DMA_ADDR resetting
        mov FLASHCON1,#00h    ; done528clear off => DMA_ADDR increase...
        mov FLASHCON,#0ch     ; CEb/        mode3/REb high
        ; wait busy... for 528 reading....
wait_end528:
        mov a,FLASHCON1
        anl a,#80h
        cjne a,#80h,wait_end528
```

SlectFLASH : reset='0' => on board flash, '1' => SMC select

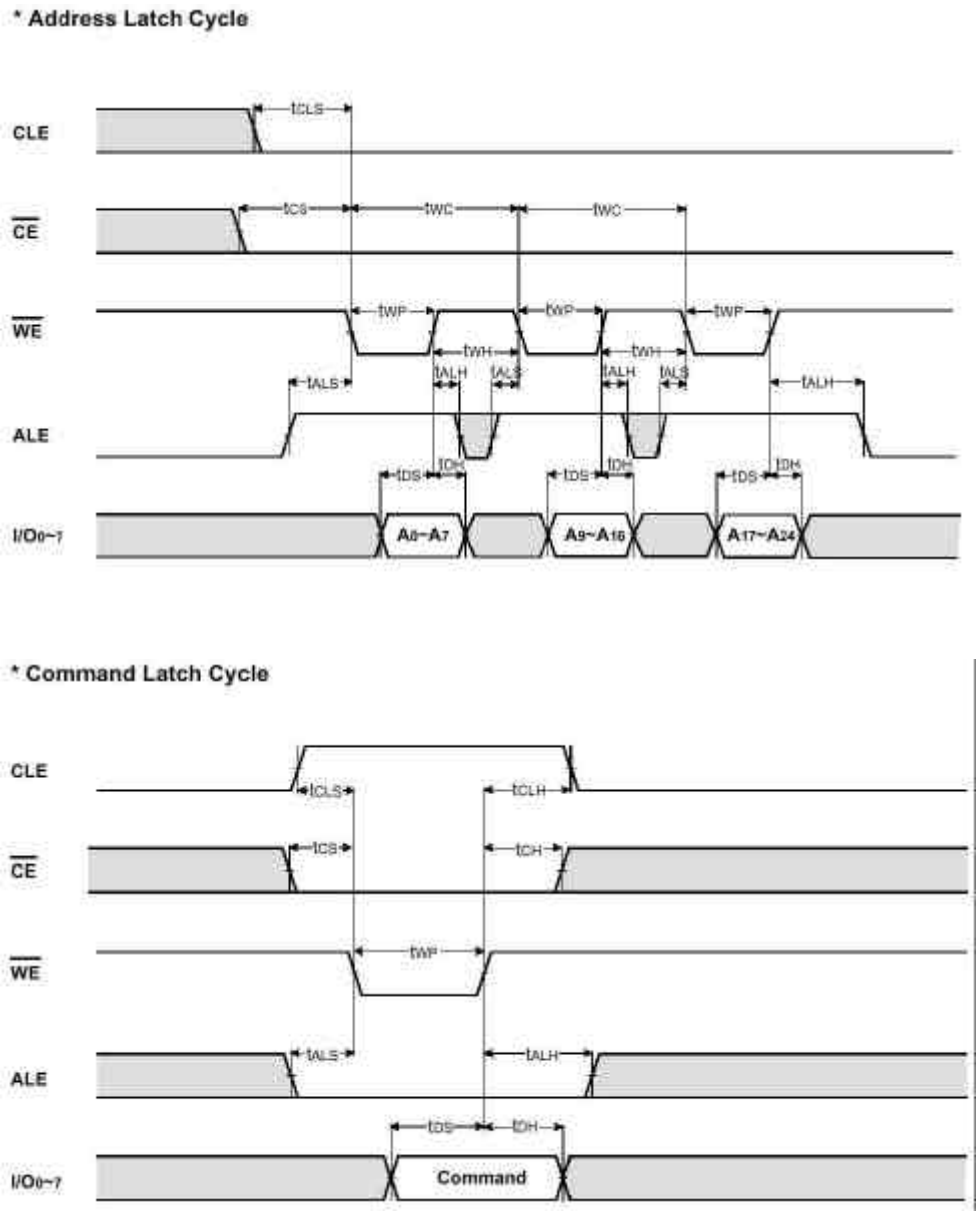
이 값을 가지고 두형태의 FLASH를 ALE/CLE등 control신호를 muxing한다...

■ FLASH command/address REGISTER : DBh (Byte access SFR)

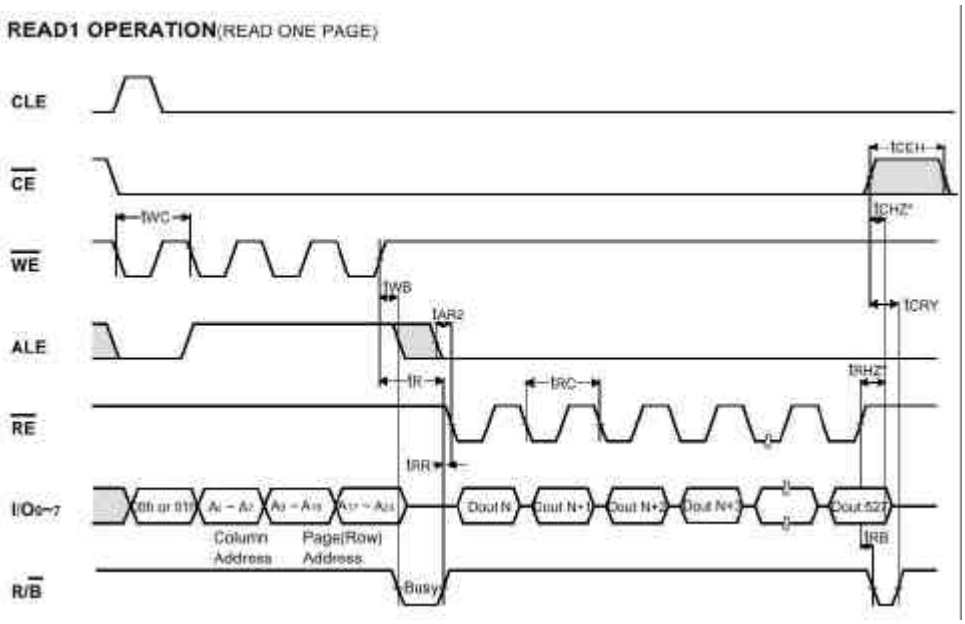
7(I/O)	6(I/O)	5(I/O)	4(I/O)	3(I/O)	2(I/O)	1(I/O)	0(I/O)
CA7	CA6	CA5	CA4	CA3	CA2	CA1	CA0

플래쉬 동작에 필요한 mode 0,1에 필요한 command 및 address를 저장시키는 register이다. 필요한 동작에 따라 command 및 address를 써 넣어주면 된다. 일반적으로 read는 0x00, erase는 0x60, program은 0x80, status reading은 0x70이며 자세한 command 및 address형태는 FLASH나 SMC의 datasheet를 참고 하면 된다.

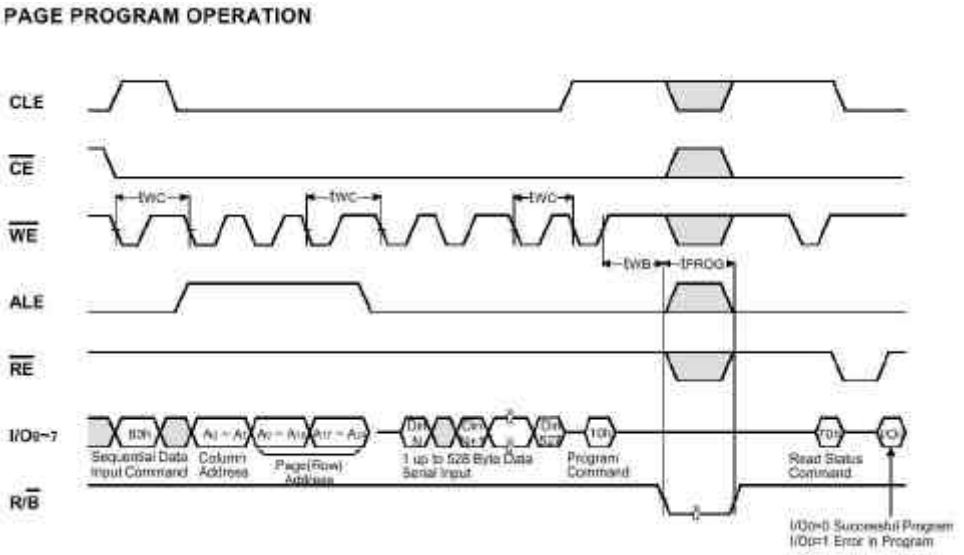
다음은 flash memory의 기본 동작 timing을 나타내었다.



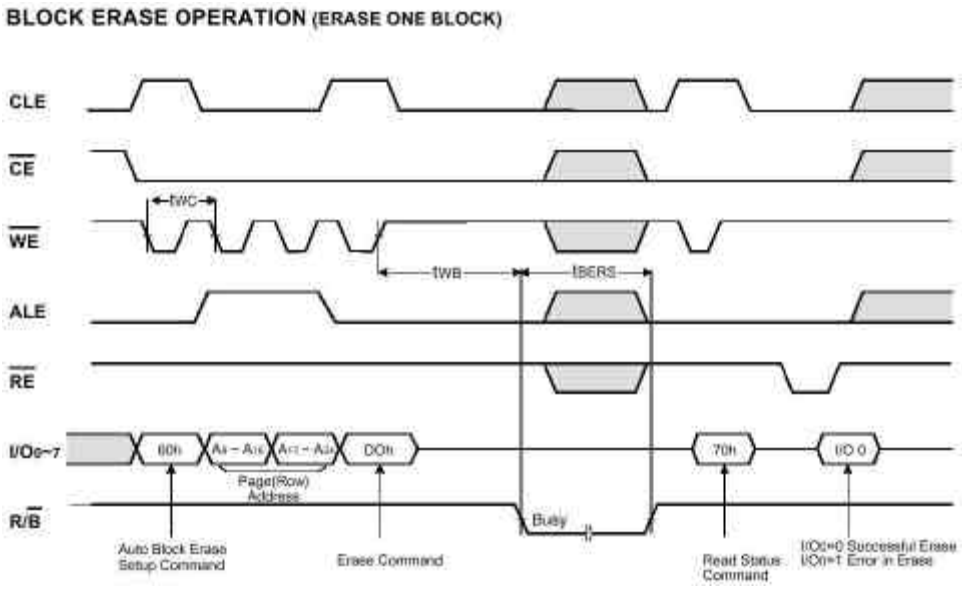
[그림4-8] FLASH에 address/command를 입력시키는 timing



[그림4-9] FLASH 512byte Block read timing



[그림4-10] FLASH 512byte Programming timing



[그림4-11] FLASH 512byte Erase timing

위의 timing diagram과 설계된 control register를 사용하는 다음 firmware code를 참고하면 된다. 전체 firmware는 지면상 다 공개를 하지 못하는 것을 밝혀둔다

```
void flash_block_read(Word page) {
//////////////////////////////////////////// for SMC
    FLASHCON1=0x01;  // SMC select
//////////////////////////////////////////// for SMC
    DMACON = 0x40;  // FLASH DMA access
    FLASHCON0 = 0x03;   // CEb/Mode0
    FLASHCA  = 0x00;   // flash command
    FLASHCON0 = 0x21;  // CEb/CLE  mode0/WEb
    FLASHCON0 = 0x23;  // CEb      mode0
    FLASHCON0 = 0x03;  // CEb      mode0
    FLASHCON0 = 0x11;  // CEb/ALE  mode0/WEb
    FLASHCA  = 0x00;   // flash #1 addr
    FLASHCON0 = 0x13;  // CEb      mode0
    FLASHCA  = page;   // flash #2 addr
    FLASHCON0 = 0x11;  // CEb/ALE  mode0/WEb
    FLASHCON0 = 0x13;  // CEb      mode0
    FLASHCA  = page>>8;   // flash #3 addr
    FLASHCON0 = 0x11;  // CEb/ALE  mode0/WEb
    FLASHCON0 = 0x13;  // CEb      mode0
//////////////////////////////////////////////// for SMC 64Mbyte
//    FLASHCA  = page>>16;   // flash #3 addr
//    FLASHCA  = 0x00;   // flash #4 addr
//    FLASHCON0 = 0x11;  // CEb/ALE  mode0/WEb
//    FLASHCON0 = 0x13;  // CEb      mode0
//////////////////////////////////////////////// for SMC 64Mbyte
    FLASHCON0 = 0x03;  // CEb      mode0    ALE OFF
    while(!(FLASHCON0 & 0x80));  // READYB HIGH
//////////////////////////////////////////// for SMC
    FLASHCON1=0x41;  // dma address resetting
    FLASHCON1=0x01;  // (= done528 toggle)
//////////////////////////////////////////// for SMC
    FLASHCON0 = 0x0c;  // CEb   Mode3/REb
    while(!(FLASHCON1 & 0x80));  // waiting end..
    while(!(FLASHCON1 & 0x80));  // waiting end..
    FLASHCON0 = 0x43;   // CEb HIGH
    delay(5);
}
```

4. Multi-Media Card 블럭의 설계

MMC카드는 크기가 아주 작은 관계로 MP3나 디지털카메라 같은 소형 디지털제품에 많이 쓰인다. 사용의 편리성으로 인해 그 사용이 증가되는 추세이나 가격이 아직 비싸고 그 용량이 크지 않다는 점에서 제품 cycle이 그렇게 길게 가지는 않을 것으로 판단된다. 그러나 일반적으로 많이 사용하는 serial FLASH card인 MMC를 control해 보는 것도 좋을 것이라 판단하여 interface하게 되었다.

위 2장에서도 설명하였지만 MMC는 2가지의 protocol모드로 데이터를 read/write한다. 2가지 모드의 차이점은 다음과 같이 다시 요약할수 있다.

MultiMediaCard	SPI
Three-wire serial data bus (Clock, command, data)	Three-wire serial data bus (Clock, dataIn, dataOut) + card specific CS signal.
Up to 64k cards addressable by the bus protocol	Card selection via a hardware CS signal
Easy card identification	Not available
Error-protected data transfer	Optional. A non protected data transfer mode is available.
Sequential and single/multiple block oriented data transfer	*Single/Multiple block read/write

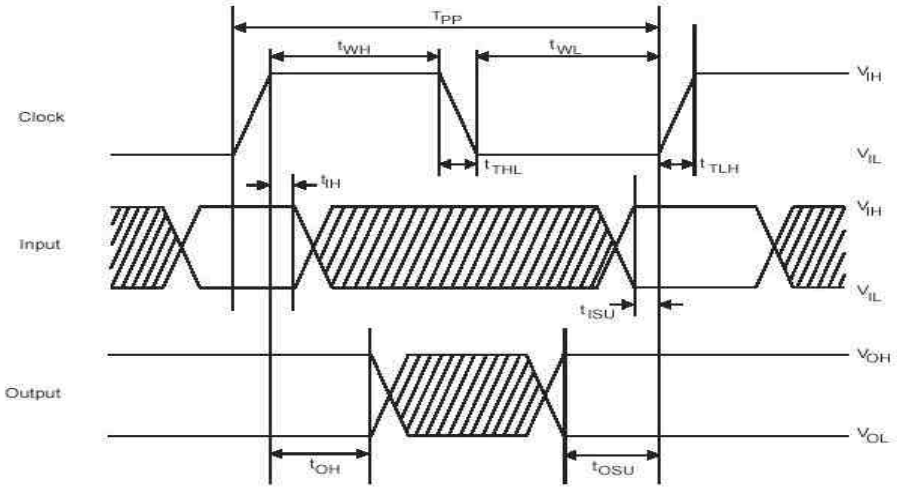
[표4-1] MultiMedia mode와 SPI mode의 비교

본 논문에서는 구현의 편이성을 위해 SPI mode만 고려하여 firmware작업을 하였으나 register나 기타 h/w적은 interface는 두가지 모드 동일하므로 나중에 추후에 multimedia mode로 firmware작업을 하면 될 것이라 생각된다.

요즘은 MMCA standard rev3.1에서는 SPI mode에서도 multiple block read가 가능하지만 일단 single block read로 인터페이스를 수행한다.

다음은 [그림4-11]은 MMC clock과 DATA의 timing diagram을 나타낸다

.



[그림4-12] MMC의 clock과 data inout의 timing diagram

위 [그림4-12]에서 알 수 있듯이 MMC는 data의 in-out이 clock의 rising edge에서 valid함을 뜻하며 설계하는 interface block에서 clock을 주는 동시에 CMD와 DATA를 가져오거나 주면 된다. 다음 [그림4-13]에는 MMC card의 state diagram을 나타내보았다. 각 모드별 command는 MMC의 datasheet를 참고하면 된다. 여기서 SPI mode에서는 Power ON후 CMD0후 IDLE-STATE에서 CMD1을 통해 MMC의 READY state까지만 인식되면 SPI mode를 사용할 수 있다. 참고로 여기서 말하는 CMD는 48bit의 MMC command bit로 포맷은 다음과 같이 구성된다.

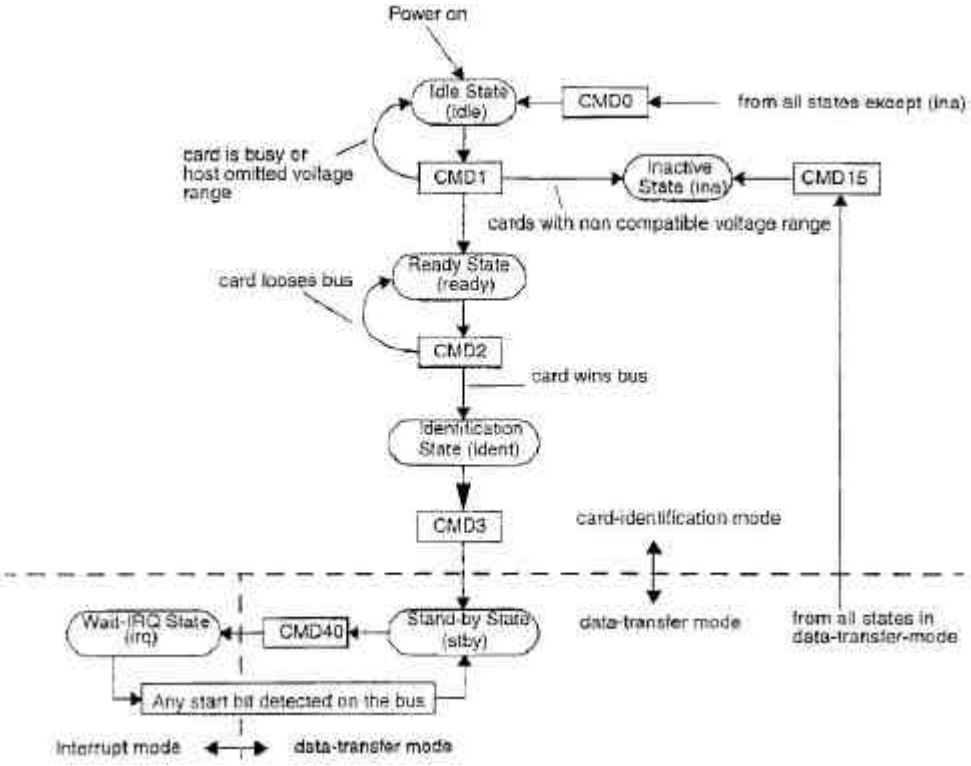
(Command length 48 bits, 2.4 μs @ 20 MHz)

0	1	bit 5...bit 0	bit 31...bit 0	bit 6...bit 0	1
start bit	host	command	argument	CRC7	end bit

Commands and arguments are listed in Table 5-3 through Table 5-9.

7-bit CRC Calculation: $G(x) = x^7 + x^3 + 1$
$M(x) = (\text{start bit})*x^{39} + (\text{host bit})*x^{38} + ... + (\text{last bit before CRC})*x^0$
$CRC[6...0] = \text{Remainder}[(M(x)*x^7)/G(x)]$

각 CMD는 분량상 논문에 기재하지 못하며 MMC의 datasheet를 참고하면 된다. 간단하게 기본 상위 8bit command만 [표4-2]에 나타내어 보았다.

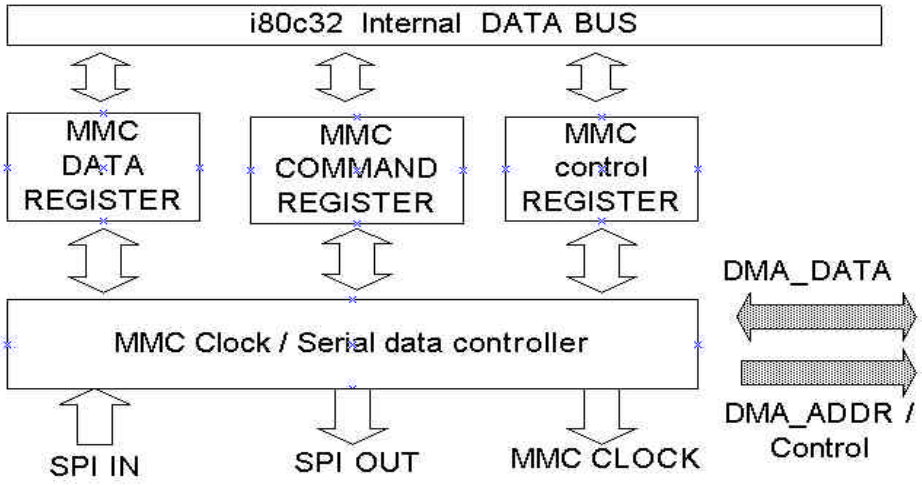


[그림4-13] MMC의 state diagram

CMD_START_BIT	0x40	SEND_STATUS	+13
GO_IDLE_STATE	+0	SET_BUS_WIDTH_REGISTER	+14
SEND_OP_COND	+1	GO_INACTIVE_STATE	+15
ALL_SEND_CID	+2	SET_BLOCKLEN	+16
SET_RELATIVE_ADDR	+3	READ_BLOCK	+17
SET_DSR	+4	READ_MULTIPLE_BLOCK	+18
SELECT_DESELECT_CARD	+7	WRITE_DAT_UNTIL_STOP	+20
SEND_CSD	+9	WRITE_BLOCK	+24
SEND_CID	+10	WRITE_MULTIPLE_BLOCK	+25
READ_DAT_UNTIL_STOP	+!1	PROGRAM_CID	+26
STOP_TRANSMISSION	+12	PROGRAM_CSD	+27
SET_WRITE_PROT	+28	TAG_ERASE_GROUP_START	+35
CLR_WRITE_PROT	+29	TAG_ERASE_GROUP_END	+36
SEND_WRITE_PROT	+30	UNTAG_ERASE_GROUP	+37
TAG_SECTOR_START	+32	ERASE	+38
TAG_SECTOR_END	+33	CRC_ON_OFF	+39
UNTAG_SECTOR	+34		

[표4-2] MMC의 기본 상위 8bit command

다음은 VHDL로 구현한 MMC 제어블럭에 대한 diagram을 나타내어 보았다.



[그림4-14] 설계된 MMC control의 block diagram

위의 block diagram에서 사용한 3개의 register에 대한 설명은 다음과 같다.

■ MMC Command REGISTER : FDh (Byte access SFR)

7(I/O)	6(I/O)	5(I/O)	4(I/O)	3(I/O)	2(I/O)	1(I/O)	0(I/O)
MC7	MC6	MC5	MC4	MC3	MC2	MC1	MC0

MMC SPI mode에서의 기본 6byte의 첫 번째에서 여섯 번째 command 값을 여기 레지스터에 넣어서 차례대로 보내줄 수 있다. command transmission은 control register의 setting으로 이루어진다. Multimedia mode에서도 마찬가지로 command를 저장하는 register로 사용할 수 있다.

```
ex)  MMCCMD=GO_IDLE_STATE;    // 0x40 commmand
     MMCCON=0x82;             // CMD TX start
     while(!(MMCCON & 0x01));    // CMD_TX_DONE flag check....
```

■ MMC DATA REGISTER : FEh (Byte access SFR)

7(I/O)	6(I/O)	5(I/O)	4(I/O)	3(I/O)	2(I/O)	1(I/O)	0(I/O)
MD7	MD6	MD5	MD4	MD3	MD2	MD1	MD0

MMC SPI mode에서의 MMC CLOCK에 따라서 MMC card에서 in/output되는 Data를 저장한다. 여기서 Mutimedia mode로 사용할 때 register자체는 inout이 되지만 전체 system에서 pin을 inout처리(open drain 및 push pull)를 고려해줘야 한다. 참고로 data를 받아들이기 위해선 clock을 출력시키며 SPI_out에 0XFF를 출력시켜주면 SPI_in에 data를 받아들일 수 있다. 따라서 write/read function으로 데이터를 처리한다.

```
ex) Byte MMC_WRITE_READ(Byte dt) {
        MMCCMD=dt;
        MMCCON=0x82;              // CMD TX start
        while(!(MMCCON & 0x01));    // CMD_TX_DONE flag check....
        return (MMCDATA);}
```

■ MMC Control REGISTER : FFh (Byte access SFR) reset=> 80h

7(I/O)	6(I/O	5(I/O)	4(I/O)	3(I/O)	2(I/O)	1(I)	0(I/O)
MMCEN	X	X	RS/CSb	BLK_READ_END	BLK_READ_EN	CMD_TX	CMD_TXD_DONE

7bit : MMCEN - chip output pin인 MMC_VCCEN과 맞물려있다. TR로 MMC power 를 on/off해서 절전모드로 사용할 경우 사용된다. default는 high이다.

4bit : RS/CSb - Multimedia mode에서 사용하지 않으나 MMC SPI mode에서 MMC enable/disable할 때 사용된다. LOW 일 때 MMC가 enable된다

3bit : BLOCK_READ_END - 513 byte를 DMA로 전송한후 high가 된다. software 처리로 high가 된걸 확인후 low로 만들어 주면 다음 block reading에서512전송 end_flag로 사용할 수 있다.

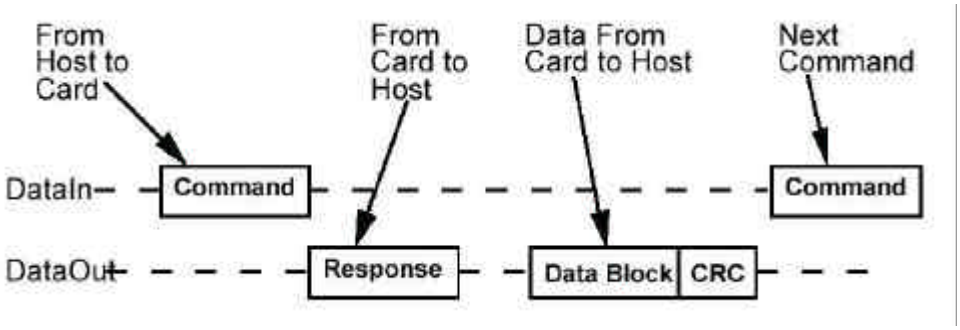
(주의 : 반드시 HIGH 값을 확인->S/W처리로 LOW로 만들어줘야 한다.)

2bit : BLOCK_READ_EN - 이 flag가 HIGH가 된후 513byte(512byte + CRC)를 한꺼번에 받아서 DMA에 저장한다. 513byte가 DMA로 writing된후 hardware처리로 low로 된다. 이 값이 high가 되기전에 MMC가 DMA를 access할 수 있도록 DMA_CON register를 setting해야 한다.

1bit: CMD_TX : 저장되어 있는 command를 이 flag값이 HIGH가 되면 MMC_CLK이 출력되면서 serial로 command가 출력된다. 8bit가 전송되고 나서는 자동으로 low로 바뀐다.

0bit : CMD_TX_DONE : 저장되어 있는 command를 모두 serial 전송시킨후 high가 된다. CMD_TX값에 의해서 clear된다.

위 3개의 register로 다음과 같은 read sequence에 의해서 MMC를 control하면 된다. write operation 및 기타 operation도 마찬가지이며 [그림4-15]의 control 및 data in/out을 수행한후 response를 check후 진행하면 된다.



[그림4-15] MMC의 SPI mode에서의 data read operation

위 READ operation을 수행하는 firmware core는 다음과 같다

```
void MMC_read_sector(Dword PhySector)
{

    Byte data temp=0;
    Word wi=0;
    Dword MMC_addr;
    Byte data ex_cmd1,ex_cmd2,ex_cmd3,
         ex_cmd4,ex_cmd5;

    MMC_WRITE_READ(0xff);        // read 2 byte
                                    CRC high
    MMC_WRITE_READ(0xff);
    MMC_WRITE_READ(0xff);        // flush before
                                    CS high

    MMC_disable();
    MMC_WRITE_READ(0xff);
    MMC_WRITE_READ(0xff);
    MMC_enable();

    temp=MMC_WRITE_READ(0xff);
```

```
while(temp!=0xff) {
  temp=MMC_WRITE_READ(0xff);
 }

  MMC_addr = PhySector << 9;
  ex_cmd1 = (Byte) ( MMC_addr >> 24 );
  ex_cmd2 = (Byte) ( MMC_addr >> 16 );
  ex_cmd3 = (Byte) ( MMC_addr >> 8  );
  ex_cmd4 = (Byte) ( MMC_addr ) ;
  ex_cmd5 = 0xff;

  MMC_WRITE_READ(0xff);
  MMC_WRITE_READ(0xff);
  MMC_WRITE_READ(0xff);
  MMC_WRITE_READ(0xff);
```

```
 MMCCMD=READ_BLOCK;
 MMCCON=0x82;                 // CMD TX start
 while(!(MMCCON & 0x01));  // CMD_TX_DONE
                                 flag check....
 MMCCMD=ex_cmd1;
 MMCCON=0x82;                 // CMD TX start
 while(!(MMCCON & 0x01));  // CMD_TX_DONE
                                 flag check....
 MMCCMD=ex_cmd2;
 MMCCON=0x82;                 // CMD TX start
 while(!(MMCCON & 0x01));  // CMD_TX_DONE
                                 flag check....
 MMCCMD=ex_cmd3;
 MMCCON=0x82;                 // CMD TX start
 while(!(MMCCON & 0x01));  // CMD_TX_DONE
                                 flag check....
 MMCCMD=ex_cmd4;
 MMCCON=0x82;                 // CMD TX start
 while(!(MMCCON & 0x01));  // CMD_TX_DONE
                                 flag check....

 temp=MMC_WRITE_READ(ex_cmd5);

 MMC_errorcheck(temp);

    DMACON=0x10;   // MMC DMA access (write)
    MMCCON=0x84; //DMA Block read command

wait_blk_rd:
        temp = MMCCON & 0x08;
        if (temp!=0x08) goto wait_blk_rd;

       MMCCON=0x80;   //clear block read end flag
}
```

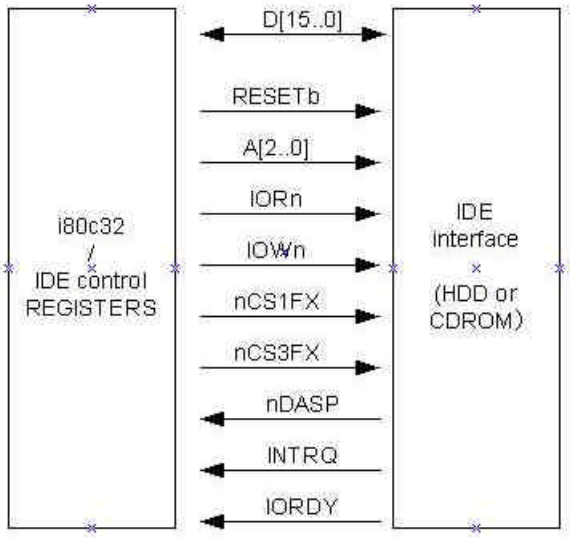
```
Byte MMC_errorcheck(Byte dt)
{
   Word LL=0;
   Byte temp,dt0;
   dt0=dt;
    for (LL=0;LL<524287;LL++)
    {
       if(dt0 != 0xff) {
            if (dt0==MMC_NO_ERROR) {
                temp = 0x00;}
            else if(dt0==START_BYTE) {
                return(0); }

            else {
                    while(1);
                }
       }
        dt0=MMC_WRITE_READ(0xff);
    }
        while(1);
}
```

MMC에 대한 추가적인 firmware code는 Apendix #2의 firmware code를 참고하여 다른 operation도 수행할수 있다.

4. IDE(integrated Drive Electronics) interface 블럭의 설계

IDE는 ATA(At-attachment)로 불려지며 여기서는 ATA-1 표준 interface를 따랐음을 밝혀둔다. IDE controller는 따로 별개의 control block으로 만들어지지 않고 pin과 register를 mapping하여 register를 control함으로서 pin으로 입출력되는 control signal을 처리하였다. 해당 register를 firmware로 control하여 입출력데이터를 DMA로 80c32 core에서 전송시켜 바로 MP3 stream을 play하는 구조로 기능 블럭을 구현하였다. 기본적으로 HDD를 대상으로 하였으며 CDROM은 이러한 ATA가 확장된 ATA-packet interface라 불리는 ATAPI 방식으로 데이터를 주고 받으나 기본 control 및 register는 거의 동일하므로 설계된 interface control register로 CDROM을 control하는 것이 가능하다.

다음은 [그림4-16]는 IDE 장비와 설계된 마이크로프로세스간의 기본적인 인터페이스 diagram이다.



[그림4-16] IDE 장비와 설계된 마이크로프로세스간의 interface diagram

이러한 interface를 가지고 control할 IDE의 기본 control register는 [표2-4] IDE interface I/O function table에서 설명하였으며 table에 나타난 control이 다음 설명할 4개의 register로 구현 가능하다. 다음은 설계에 사용된 4개의 IDE register에 대해 설명하겠다.

■ IDE Control #0 REGISTER : DCh, reset=> 01100011b ,0x63

7(O)	6(O)	5(O)	4(O)	3(O)	2(O)	1(O)	0(O)
RESETb	nDIOW	nDIOR	DA2	DA1	DA0	nCS3FX	nCS1FX

RESET : is asserted for at least 25 microseconds after voltage levels have stabilized during power on and negated thereafer unless the drive needs to be reset at a later time.

nDIOW : is the Write strobe signal. The rising edge of nDIOW clocks data from from the host to the drive.

nDIOR : is the Read strobe signal. The falling edge of nDIOR enables data from the drive onto the host data bus.

DA0-2 : used to select a register or a data port in the drive.

nCS3FX : is a chip select generated by address decoding circuitry from host address lines A3...A9. Usually asserted during I/O operations to ports 3F0 through 3F7. -CS3FX is valid during 8 bit transfers to/from the Control Block Registers, Alternate Status Register, Device Control Register and drive address.

NOTE: The primary host adapter is accessed via I/O addresses 1FX and 3FX while the secondary host adapter is accessed via I/O addresses 17X and 37X. See Ed's note under the paragraph "Register Address Decoding" above.

nCS1FX: is a chip select generated by address decoding circuitry from host address lines A3...A9. Usually asserted during I/O operations to ports 1F0 through 1F7. -CS1FX is used to access the eight hard disk Command Block Registers.

■ IDE Control #1 REGISTER : DDh (Byte access SFR) reset=> 00h

7(I)	6(I)	5(I)	4(I)	3(I)	2(I)	1(I)	0(I)
nDASP	INTRQ	IORDY	NOT USED	NOT USED	NOT USED	NOT USED	NOT USED

nDASP : is a time multiplexed signal which indicates that a drive is active or that Drive 1 is present. It is an open collector output.

During power-on initialization or after reset, -DASP will be asserted by Drive 1 within 400 msec to indicate its presence. Drive 0 will allow up to 450 msec for Drive 1 to assert -DASP. If Drive 1 is not present, Drive 0 may use -DASP to drive an activity LED.

-DASP will be negated following acceptance of the first valid command by Drive 1 or after 31 seconds, whichever comes first. Any time after negation of -DASP, it may be used by either drive as an activity indicator.

INTRQ : is used to interrupt the host system when the drive has a pending interrupt, the drive is selected and the host has enabled drive interrupts by clearing nIEN in the Device Control Register.

INTRQ is negated by:
+ assertion of -RESET.
+ the setting of SRST in the Device Control Register.
+ the host writing to the Command Register.
+ the host reading the Status Register.

NOTE: Some drives may negate INTRQ on PIO data transfer completion, except on a single sector read or on the last sector of a multi-sector read. On PIO transfers INTRQ is