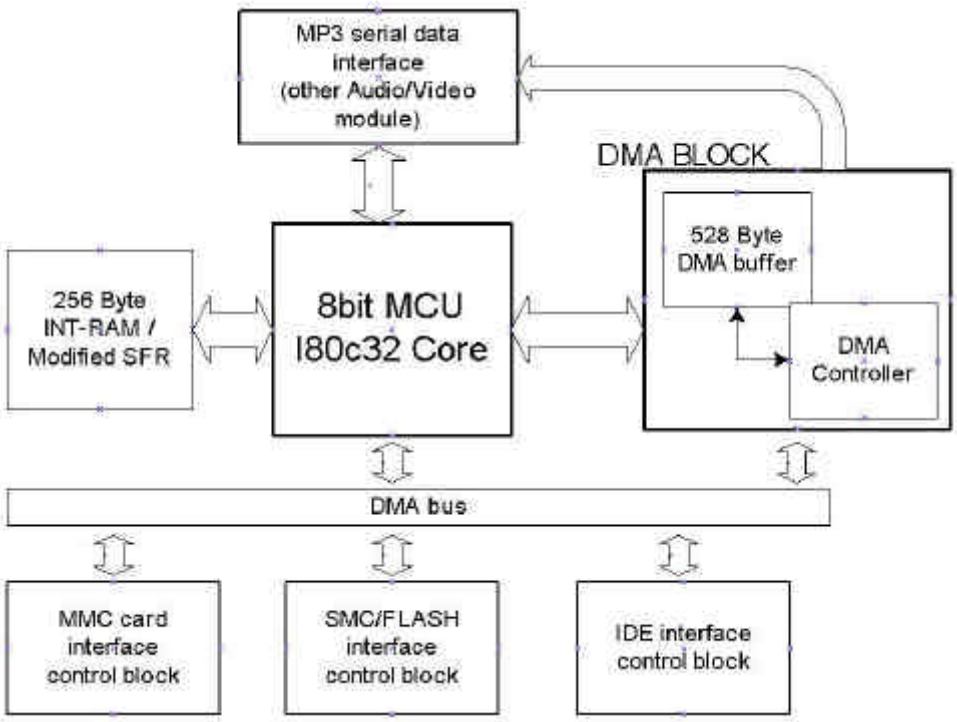
Ⅲ. 디지털오디오 저장매체 Interface용 μController

1. Digital Audio 저장매체 interface용 μController의 개요

구현하고자 하는 마이크로 프로세서의 주된 임무는 디지털 데이터가 저장되어 있는 여러 저장매체에서 데이터의 입출력처리를 맡는 것이다. 여기서 디지털 오디오 데이터용이라고 언급한 것은 실제 구현상에서 검증 대상이 압축된 디지털 오디오데이터를 read/write하는 MP3 player system임을 알려둔다. 설계하고자 하는 인터페이스용 마이크로컨트롤러에 대한 block diagram은 다음과 같다.



[그림3-1] 설계하고자하는 interface용 controller의 block diagram

위의 block diagram에서 256Byte SFR영역은 original compatible 80c32 core에서 수정하여 다른 주변 control 블럭과 인터페이스가 용이하게 register를 추가하여 수정하였으며 각 저장매체별 인터페이스 control 블록과 read/write할 데이터가 처리되는 DMA 블럭을 구성하여 실제 설계검증을 위해 구현해놓은 MP3 stream output block과 i80c32의 데이터 교환을 용이하게 해놓았다.

각 매체와의 기본적인 인터페이스는, 예를 들면 초기화, i80c32 core에서 해당 register의 setting으로 직접 이루어지며 block 단위의 data read/write에서 DMA가 담당하여 한꺼번에 read/write하는 형태로 이루어지게 된다.

2. i8032 core의 기본사양 및 특징

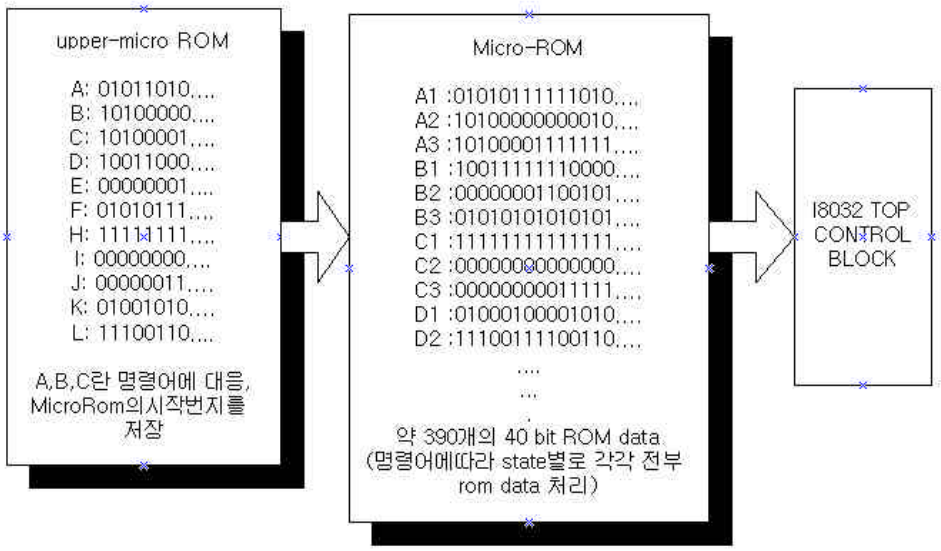
본 논문의 interface용 마이크로컨트롤러의 설계에 핵심적인 i80c32 core에 대한 기본사양 및 특징을 설명하고자 한다. 여기서 사용된 i80c32 core는 부산대학교 전자공학과 VLSI 실험실에서 개발한 것으로 현재 IDEC IP (IP#: IP-PSSFV-000040-01)로 등록되어 있는 i80c32/52/31/51 core를 사용하였으며 이 core는 예전 core의 area를 줄이기 위해 추가로 내부의 microrom - instruction에 따른 전체 block의 control신호를 발생시켜주는 block - 을 수정하여 gate수를 줄인 core이다. Gate수의 count는 옛 LG의 0.8마이크론 SoG공정의 Compass tool library로 synthesis후 측정하였다. 다음은 변화된 gate수를 표로 나타내어 보았다.

수정한 Entity	기존 gate수	Optimize후	변화된gate	
upper_mrom	545.4	104.8	-440.6	
rom_inc	139.3	0	-139.3	필요없음
next_mux	60.8	0	-60.8	필요없음

수정한 Entity	기존 gate수	Optimize후	변화된gate	
add_sel	87.5	0	-87.5	필요없음
rom_inc_en	19	0	-19	필요없음
adder8	170	96	-74	CLA_Adder로바꿈
pipe_reg	333	235.5	-97.5	
Mrom	4461.8	1027.3	-3434.5	
new_mrom_decoder	0	1053.50	+1053.5	추가된 블럭
branch_check	0	61	+61	추가된 블럭
		TOTAL	-3238.7	
0.8μ SOG공정에서의 Gate변화량			16647.2 => 13408.5	

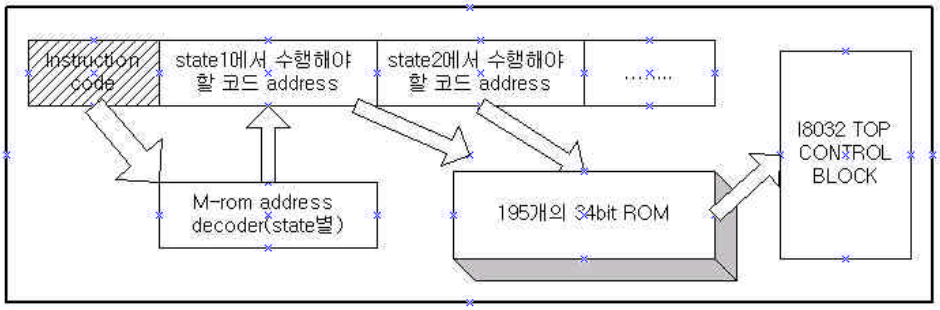
[표3-1] Gate줄이는 작업후의 i8032core Gate의 변화량

게이트 줄이는 작업전의 i8032 명령에 따라 전체 블록을 control하는 신호를 생성시키는 MicroROM의 구조는 다음과 같았다.



[그림3-2] Gate줄이기 전의 i8032core의 MicroROM구조

이 상태의 MicroROM을 인트스럭션에 따라 스테이트별로 수행되는 control code를 sort작업을 통해 같은 코드들을 분류한 다음 한 개의 MicroROM 데이터로 처리한 다음 MicroROM의 address폭을 줄였다. 작업의 전체 diagram은 [표3-3]에 단순화해서 나타내었다. 단순히 synthesis tool에서 최적화 될것이라 믿었던 MicroROM의 gate수를 상당히 많이 줄일수 있었다.

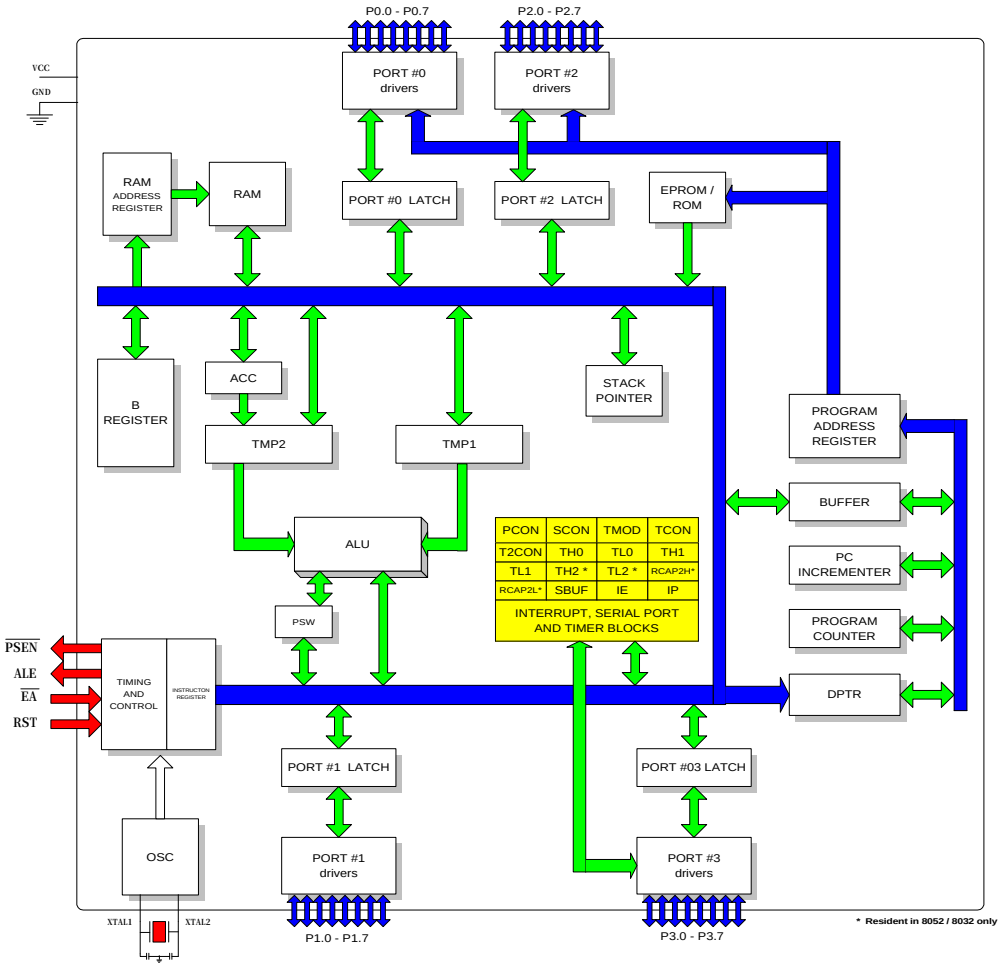


[그림3-3] Gate줄이기 위한 MicroROM의 수정작업

그러나 gate수가 줄어든대신 스테이트별로 마이크로롬 address decoder가 중간에 들어가므로 전체 명령어에 대한 스테이트별 delay가 커지는 단점이 있다. 그러나 파이프라인구조도 아닌 단순 8bit 컨드롤러인 i80c32 core에서는 속도의 증가보다 area측면에서의 감소가 훨씬 매력이 있어 보인다. 이러한 작업을 통하여 만들어진 i80c32 core의 신뢰성 test를 위해 IDEC의 IKOS emulation작업을 통해 기존의 testvector를 모두 수행시켜 그 결과를 비교하여 수정된 core의 검증 작업을 마칠 수 있었다.

아래 [그림 3-3]은 기본 i80c31/32 architecture block diagram을 나타내어 보았다

MCS-51 Architecture Block Diagram



[그림3-4] i80c32 Architecture Block Diagram

3. 디지털 오디오 저장매체 interface를 위한 수정된 i8032 core

위에서 언급한 Gate수가 줄어든 i80c32 core를 사용하여 여러 디지털오디오 데이터가 저장되어 있는 저장매체, FLASH memory, SMC, MMC, IDE(HDD,CDROM)를 인터페이스를 구현함에 있어 그 용이성과 개발시간 단축을 위하여 control register를 할당하여 각각의 기능 블럭을 control하는 것 이 바람직하다. 이번3-3장에서는 i80c32 core를 구현하고자 하는 인터페이스용 컨트롤러에 적용시키기 위해 i80c32 core를 수정한 부분과 추가한 부분을 위주로 설명하겠다

3.1 수정된 SFR(Special Function Register) 영역

원래 i80c32의 internal memory의 상위 128byte SFR(Special Function Register)영역을 수정하여 디지털데이터 저장매체와의 인터페이스를 위한 register들을 삽입했다. 설계를 해 나가는 과정에서 필요로 한 레지스터들은 추가로 계속 수정되었으며 아래 [표3-2]에 전체 수정된 80c32의 SFR영역을 나타내 보았다.

P4,P5,P6,P7은 Bit addressable한 8bit register로 기존의 P0,1,2,3의 Port를 더 확장하였고 Byte addressable한 P8,P9를 추가하여 총 10개의 8bit I/O를 만들었다. 그 외 DMA control을 위한 register 4개와 MMC control에 필요한 4개의 register, FLASH/SMC control을 위한 3개의 register, IDE interface를 위한 4개의 register, MP3 stream을 control하기 위한 3개의 register로 모두 22개의 8bit register를 추가 하였다. i80c32 compiler에서 미리 define해 놓은후 자유롭게 일반 register type으로 간편하게 사용할수 있다.

예) SFR DMA_CON 0xF9 // 내부 memory 0xF9번지=>DMA_CON reg

```
DMACON   = 0x20;  // MP3 DMA access
while(!(DMA_CON & 0x04));  // wait all 512 byte trasfer
```

F8	P7	DMA_CON	DMA_ADDR_HIGH	DMA_ADDR_LOW	DMA_DATA	MMC_CMD	MMC_DATA	MMC_CNTR
F0	B							
E8	P6							
E0	ACC							
D8	P5	FLASH_CON0	FLASH_CON1	FLASH_COM_ADDR	IDE_CON0	IDE_CON1	IDE_LSB	IDE_MSB
D0	PSW						MP3DATA	MP3CON
C8	T2CON		RCAP2L	RCAP2H	TL2	TH2	P8	P9
C0	P4							
B8	IP							
B0	P3							
A8	IE							
A0	P2							
98	SCON	SBUF						
90	P1							
88	TCON	TMOD	TL0	TL1	TH0	TH1		
80	P0	SP	DPL	DPH				PSON
	0	1	2	3	4	5	6	7

[표3-2] 수정된 i80c32의 SFR 영역

i80c32의 assembly나 C compiler에 pre-define만 해놓고 자유롭게 register를 사용할 수 있다. 필요에 따라 더 추가적인 여러 8bit register를 할당해 놓고 단지 직접 어드레싱모드로 access 가능하다. internal memory에 mapping되어 있으므로 간접 어드레싱모드를 통해 상위 128byte를 register가 아닌 Internal

memory로 access 할 수 있다.

3.2 Interface용 기능블럭

위에서도 설명하였지만 기본 i80c32 core에 추가로 FLASH memory / SMC / MMC / IDE를 interface할 기능 블럭이 포함되고 이렇게 설계한 interface용 마이크로컨트롤러의 실시간 검증을 위해 각 기능 블럭에서 처리한 MP3 data를 serial stream형태로 내 보내줄 수 있는 기능 블럭도 추가로 포함되었다.
각 기능 블럭을 크게 나누면 5가지로 나눌 수 있다.

① DMA block
   => 각 저장매체에서 읽고쓰는 데이터가 처리되는 곳이다. i80c32의 명령 흐름과 관계없이 enable되었을 때 각각의 저장매체 컨트롤 블럭을 직접 control 한다. block단위의 연속된 data 처리를 담당한다.
② FLASH / SMC control
   => 반도체 FLASH 및 SMC를 read/write하거나 erase하는 기능블럭이다. 병렬로 8bit data를 처리하며 해당되는 control register를 setting하여 device pin들과 control register를 맞물려서 DMA나 i80c32 core에서 컨트롤 할 수 있는 구조를 가진다.
③ MMC control
   => Multi-Media Card를 컨트롤 하는 기능블럭이다. serial로 데이터를 처리 하므로 SMC보다 느려서 i80c32의 main clock을 MMC의 기본clock으로 사용 한다.
④ IDE control
   => 기본 ATA command를 수행할수 있게 기본 IDE pin과 control register를 맞물려 놓았으며 16bit data가 처리되므로 기능블럭에서 따

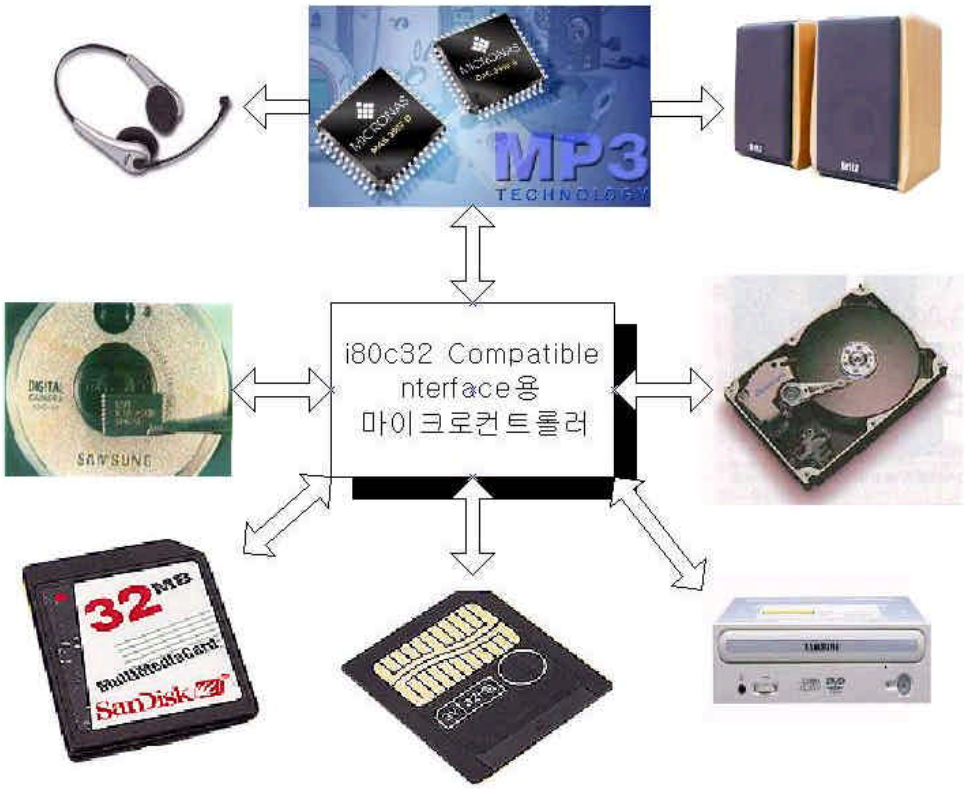
로 독립적으로 처리 하는게 아니라 i80c32의 직접 control로 데이터의 read/write가 수행될 수 있게 software적으로 처리하였다.

⑤ MP3 stream control

=> 위의 디바이스들에서 들어온 데이터가 DMA block에 들어간 후 DMA 블록에서 순차적으로 MP3 stream data를 받아들여서 외부의 MP3 decoder와 서로 인터페이스 시키는 기능 블럭이다.

각 기능블럭의 세부사항은 Ⅳ장에서 자세히 논하겠다.

4. 설계될 마이크로컨트롤러를 이용한 MP3-Player system의 개요



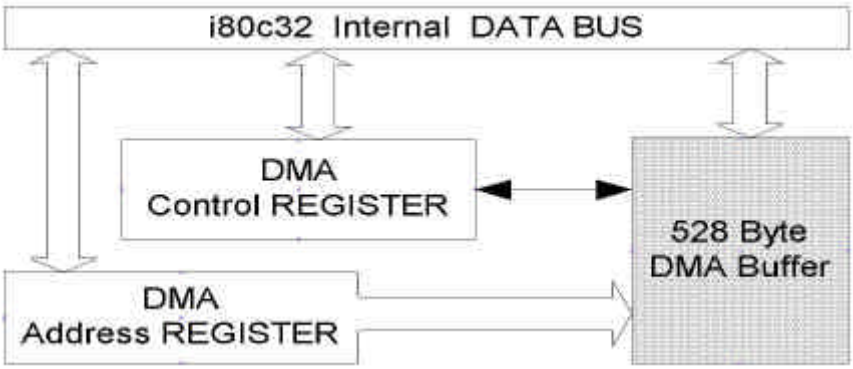
[그림3-5] 설계될 마이크로컨트롤러를 이용한 MP3-player system

Ⅳ. 각 저장매체별 Interface 기능 블럭 설계

1. DMA 블럭의 설계

여기서의 DMA는 528byte의 buffer를 가지고 주변 장치와 간단하게 8bit 512 block씩 데이터를 주고 받을 수 있는 구조로 되어 있다. 528byte는 FLASH나 SMC에서 신뢰성을 높이기 위해서 데이터 영역의 에러 감지 및 교정을 위한 ECC로 사용되는 16byte 때문에 추가로 구성하였으며 설계하는 interface용 마이크로컨트롤러의 area측면에서 1개의 528byte만 buffer를 구성하였다. 2개를 구성하면 한 개의 528byte buffer가 read중일 때 다른 하나의 528buffer로 write기능을 수행한다면 더 빠르게 데이터 입출력을 수행할 수도 있을 것이다. 현재의 구성으로 DMA에서 512byte단위로 read/write한 후 해당 device의 상태를 다시 check후 i80c32 core에서 DMA block을 다시 control해주는 simple한 구조를 가진다.

기본 DMA control block diagram은 다음과 같다.



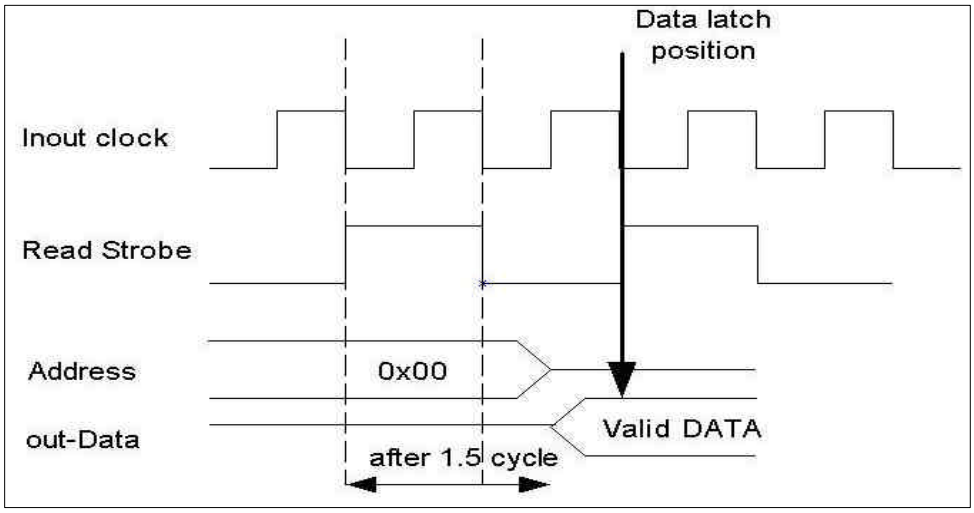
[그림4-1] DMA control block diagram

여기서 DMA 528 byte buffer는 실제 FPGA에서 flip-flop으로 구현하기에는 gate수를 크게 차지하므로 실제 구현할 대상의 FPGA인 Altera FLEX10k250에서 제공하는 lpm (Library of Parameterized module) library를 이용하였다. lpm ram을 사용하는 방법은 vhdl code에서 아래와 같이 사용하면 된다.

```
library lpm;
USE lpm.LPM_COMPONENTS.all;
lpm_ram_dq  GENERIC MAP(
              LPM_WIDTH    => 8,
              LPM_WIDTHAD => 10,                          -- 1024 Byte
              LPM_INDATA           => "UNREGISTERED",
              LPM_ADDRESS_CONTROL => "REGISTERED",
              LPM_OUTDATA          => "REGISTERED"   )
   PORT MAP(
            data      => DMA_RAM_IN,
            address   => DMA_RAM_ADDR,
            we         => DMA_RAM_WE,
            q          => DMA_RAM_OUT, -- before TRI
            --inclock   => VCC_H,
            inclock   => clock,              -- jspark 2001_08_31
            outclock => clock
            );
```

본 논문에서 구현하는 마이크로프로세서의 DMA buffer는 FPGA vendor에 따라 timing이 조금 다르므로 아래의 Altera LPM의 timing대로 설계를 하였으며 다른 FPGA나 SOG형태로 구현 할때는 수정해줄 필요가 있음을 밝혀둔다.

위의 528 byte buffer에 사용되는 LPM ram의 특징은 Syncronus동작을 원칙으로 하며 실제 구현의 편이상 Async동작을 원했으나 원하는 대로 동작을 하지 않아서 확실한 control을 가지고 Synchronus RAM을 구현했다. 실제 해당 address를 주고 write strobe를 준다음 data는 1/2 inout-clock후에 데이터가 valid하게 출력되었다. read할때도 마찬가지이다. 따라서 read/write strobe를 준후 하나의 RAM inout-clock후- 여기서는 i80c32 core의 main clock을 사용했다 - 데이터를 다시 latch시켰다. 아래의 [그림 4-2]는 구현한 DMA buffer의 read에서의 timing을 나타내어 보았다



[그림4-2] DMA buffer의 read/write tiiming

이번엔 DMA control 블록에서 사용하는 register에 대해 설명하고자 한다. 여기서 사용한 register는 위에서 설명한 변경된 SFR영역에서 4개가 추가되어 있는 것을 볼수 있는데, 사용한 register는 DMA_CON, DMA_addr_high, DMA_addr_low, DMA_DATA 이다. 각자의 기능에 대해서는 아래 자세히 설명하겠다.

■ DMA_CON : F9h (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)
C32_access	FLASH	MP3_BLK	MMC	x	x	x	R/Wb

RESET VALUE : 00000000

7bit : C32_access - 8032이 DMA을 access할 때 사용한다.

6bit : FLASH - FLASH moemry contoller가 DMA을 access할 수 있게 한다.

5bit : MP3_BLK - MP3 block에서 DMA을 access할 수 있게 한다.

4bit : MMC - MMC control block에서 DMA를 access할 수 있게 한다.

0bit: R/Wb - DMA를 각 인터페이스별 기능블록에서 read할것인지 write할 것인지를 결정한다. write 할때는 반드시 0로 setting 해놓고 해야 한다. read할때는 상관없다. DMA_DATA 레지스터에 값을 써넣을 때 i8032 버스에서 데이터 충돌을 피하는 목적

※ 이 레지스터를 먼저 setting해 놓아야 DMA을 바로 사용할수 있다.

■ DMA_ADDR_HIIGH: FAh (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)
X	X	X	X	X	DMA_A10	DMA_A9	DMA_A8

RESET VALUE : 00000000

8032가 DMA을 access할 때 지정하는 상위 address를 저장하는 register 사실은 지정할 필요는 없으나 512 byte이상 access할 때 사용한다. 실제 FPGA 상에서는 1024byte의 RAM이 구현되어 있는 상태이다.

■ DMA_ADDR_LOW: **FB**h (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)
DMA_A7	DMA_A6	DMA_A5	DMA_A4	DMA_A3	DMA_A2	DMA_A1	DMA_A0

RESET VALUE : 00000000

8032가 DMA을 access할 때 지정하는 하위 address를 저장하는 register이다 다른 인터페이스 기능블럭에서는 지정해줄 필요가 없다.

■ DMA_DATA: **FC**h (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)
DMA_D7	DMA_D6	DMA_D5	DMA_D4	DMA_D3	DMA_D2	DMA_D1	DMA_D0

RESET VALUE : 00000000

i8032가 DMA을 access할 때 DMA의 값을 임시로 저장하는 register. 실제 존재하는 register가 아니라 DMA_RAM_array의 8bit cell의 값을 나타낸다.

이 register에 8051에서 writing을 하면 RAM_array에 저장된다. 읽어오는 것도 address 저장시켜놓고 읽으면 RAM_array값이 읽어진다.

위 4개의 register로 기본 DMA를 control할수 있으며 각 기능별 DMA control signal들은 각각 따로 DMA block으로 들어오며 DMA_CON register로 control신호들을 mux시킨다. 간단하게 DMA register를 사용하여 내부 DMA의 값을 dump하는 routine을 적어보았다.

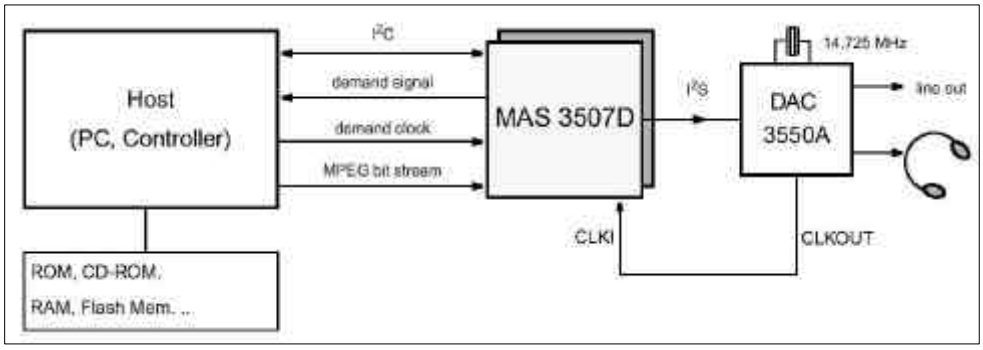
```
Byte xdata display_buffer[256];
void UART_dma_display(void) {
      Word wi=0;          Byte i=0;

      DMACON   = 0x81;   // c51 access

      DMAADDRH = 0x00;   // dma high address setting
      DMAADDRL = 0x00;   // dma low address setting
      for(wi=0;wi<=255;wi++) {
        DMAADDRL = wi;   // dma low address setting
        display_buffer[wi]=DMADATA;
       }
      for(wi=0;wi<=15;wi++) {
          for (i=0;i<16;i++) {TXD_hex(display_buffer[16*wi+i]); TXD_char(' ');}
          for (i=0;i<16;i++) TXD_char(display_buffer[16*wi+i]);
          TXD_CR_LF(); }
      DMAADDRH = 0x01;
      for(wi=0;wi<=255;wi++) {
        DMAADDRL = wi;   // dma low address setting
        display_buffer[wi]=DMADATA;
      }
      for(wi=0;wi<=15;wi++) {
          for (i=0;i<16;i++) {TXD_hex(display_buffer[16*wi+i]); TXD_char(' ');}
          for (i=0;i<16;i++) TXD_char(display_buffer[16*wi+i]);
          TXD_CR_LF(); }
          TXD_CR_LF(); }
```

2. MP3-stream output control 블럭의 설계

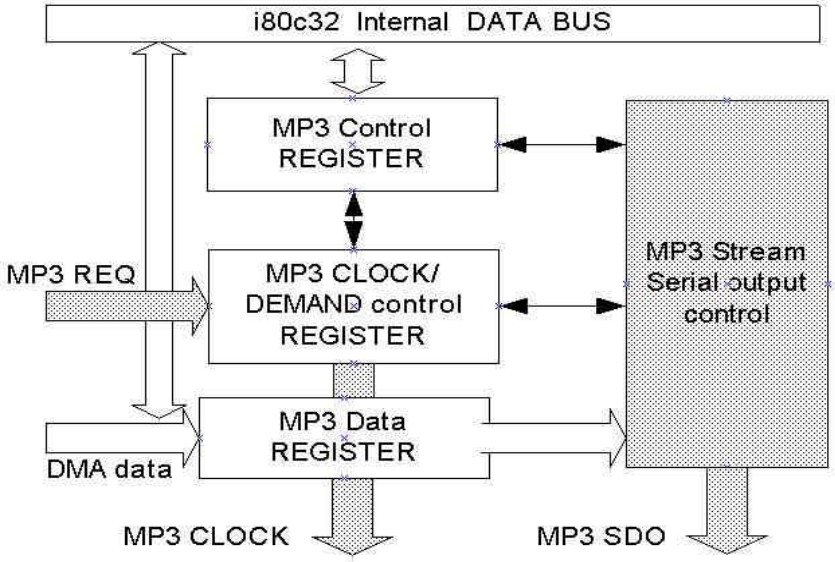
여러 저장장치에서 읽어온 데이터는 DMA에 저장되며 DMA에 저장된 MP3 stream 데이터는 i80c32의 control에 따라 MP3 control 블록을 통하여 외부 MP3 decoder에 Serial 형태로 clock과 함께 출력시켜주는 역할을 한다. FPGA 검증에 사용한 MP3 decoder는 흔히 많이 사용하는 Micronas MAS3507D, MPEG1 layer1,2,3 decoder이다. 아래의 [그림4-3]은 mp3 decoder system의 interface diagram이다. 다른 MP3 decoder 들도 마찬가지로 demand신호에 따라 serial clock과 stream data를 받아들인다. parallel로 8bit씩 받는 방식도 있으나 MP3 데이터는 보통 최대 192Kbps정도의 속도로 전달되면 되므로 일반적으로 serial stream data로 처리되곤 한다.



[그림4-3] MP3 decoder와 controller사이의 interface block diagram

본 논문에서 구현한 MP3 decoder와의 인터페이스는 serial 방식을 원칙으로 하며 demand 신호에 따라 DMA data를 MP3 control block에서 출력시켜주며 상태 flag check를 통해 다음 data buffer를 reading해서 출력시켜줄 준비를 하게 된다.

설계한 MP3 control block의 기본구조는 다음과 같다.



[그림4-4] MP3 stream out control block diagram

여기서 사용하는 control register는 2개로 구성되어 있으며 MP3 stream를 저장하는 MP3_DATA register와 MP3 decoder와의 interface를 위한 demand 신호, DMA상태 flag bit를 표시하고 있는 MP3_CNTR register로 되어 있다. 아래에 2가지 register의 구조를 나타내었다.

■ MP3DATA : D6h (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)
DATA7	DATA6	DATA5	DATA4	DATA3	DATA2	DATA1	DATA0

RESET VALUE : 00000000

Mp3 stream data를 이 레지스터에 넣는 즉시 내부 DMA 512 memory에 차

레대로 저장된다. 값이 write되면서 자동으로 address가 증가됨에 유의하자. 나중에 한꺼번에 이 데이터가 SDO형태로 나가게 된다. 내부 DMA에 의해서 쓰여지므로 C51에서 따로 access할 경우 512byte frame의 마지막 값이 읽어진다. 이 register는 어떤 저장매체를 통해 DMA에 write하는 경우 외 i80c32 core에서 데이터를 write할 때 사용된다. 따라서 예를 들면 HDD에서 데이터를 읽어올 경우 IDE control block에서 자동으로 DMA에 데이터를 write하게 된다.

■ MP3CNTR : D7h (Byte access SFR)

7(I/O)	6(I/O)	5(I/O)	4(I)	3(O)	2(O)	1(I)	0(I)
X	X	X	MP3REQ	MP3SDO_END	MP3SDO512_END	MP3SDO_DIR_SEL	MP3CLKEN

RESET VALUE : 00000000

0 bit : MP3CLKEN - 이 bit를 HIGH로 만들어 주면 MP3 데이터를 clock과 함께 출력시켜준다.

1 bit : MP3SDO_DIR_SEL - HIGH일 때 8bit stream data를 MSB 순서대로 serial로 출력시켜주며 LOW일때는 반대로 LSB먼저 serial로 출력시켜준다.

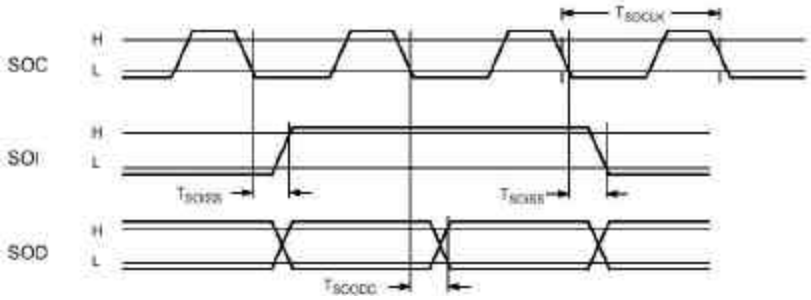
2 bit : MP3SDO512_END - 내부 DMA에 들어있는 512 byte의 MP3 stream data를 모두 출력시킨후 HIGH로 발생된다. 다음번의 MP3CLKEN신호가 하드웨어적으로 LOW로 만들어준다.

3 bit : MP3SDO_END - 1byte SDO output done flag

4 bit : MP3REQ - external MP3 Decoder DATA REQUEST

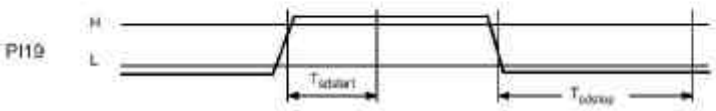
설계된 MP3 block은 다음과 같은 timing으로 MP3 decoder와 인터페이스 되며 본 검증에 사용된 MAS3507외 다른 decoder들도 이와 유사하다

Symbol	Parameter	Pin Name	Min.	Typ.	Max.	Unit	Test Conditions
t_{SOCLK}	I²S Clock Output Period	SOC		325		ns	48 kHz Stereo 32 bit/sample
t_{SOISS}	I²S Wordstrobe Hold Time after falling edge of clock	SOC, SOI	10		$t_{SOCLK}/2$	ns	
t_{SOODC}	I²S Data Hold Time after falling edge of clock	SOC, SOD	10		$t_{SOCLK}/2$	ns	



[그림4-5] MP3 stream Serial oout Timing

Symbol	Parameter	Pin Name	Min.	Typ.	Max.	Unit	Test Conditions
$T_{sdstart}$	Reaction time for data source	PI19	3.1		5.7	ms	f_s = 48 kHz, 320...64 kbit/s
$T_{sdstart}$	Reaction time for data source		4.2		9.2	ms	f_s = 24 kHz, 320...32 kbit/s
T_{sdstar}	Reaction time for data source		23.1		25.6	ms	f_s = 12 kHz, 64...16 kbit/s
T_{sdstar}	Reaction time for data source		34.8		38.4	ms	f_s = 8 kHz, 64...8 kbit/s
T_{sdstop}	Reaction time for data source				1.3	ms	



[그림4-6] MP3 stream Request Timing