

# JTAG Port를 이용한 On-Board Flash Fusing③

## JTAG Port 제어 SW 설계

두 차례의 연재에 걸쳐 JTAG의 전반적인 개념에 대해 알아보았다. 물론, 지면관계상 자세히 많은 부분들을 다루진 못했지만 On-board상의 Flash를 제어하기 위해 최소한의 필요한 내용들은 언급이 되었다고 판단된다. 다음은 그 내용들을 바탕으로 하여 운용 SW를 제작하는 방법에 대해서 알아보자.

글: 김재성/(주)메리테크 개발팀장  
jskim@meritech.co.kr

### [ 연재순서 ]

- 1 JTAG 이해와 구성(10월호)
- 2 JTAG 동작원리
- 3 Flash Memory 구조 및 동작원리
- 4 BSDL File 구조(11월호)
- 5 JTAG Port 제어 S/W 설계 (12월호)



### On-board Flash Fusing Method

일반적으로 Flash Fusing을 하기 위한 방법은 두 가지로 구분될 수 있다. JTAG의 EXTEST라는 명령을 이용하는 방법, 또 하나는 INTEST 및 DEBUG와 관련된 명령들을 사용하는 것이다. 전자의 경우, Device가 Test Mode인 상태에서 이루어지며 반면에 후자는 Normal Mode 즉 실제 Code가 Running이 되면서 이루어진다. 지면관계상 필자는 전자의 경우만을 예로 설명하겠다. 그리고 환경구성은 기본적으로 PC의 프린터 포트에 JTAG Port가 인터페이스 되어있다고 가정하고 설명을 하겠다.

### 제어 SW 구조

제어 SW의 구조는 표 1과 같이 크게 세 부분으로 구성할 수가 있다.

SW Block	Description
JTAG Port Interface Block	Parallel I/O Basic TAP interface TAPC control Code
Memory Control Block	Chip dependant Code
Flash Control Block	Chip independent (일반적인 예) Flash dependant Code

표 1. 제어 SW 구조

JTAG Port Interface Block은 기본적인 TAP 인터페이스 즉, PC와 JTAG Port 사이의 입출력, 그리고 그와 더불어 EXTEST 명령을 수행하여 외부 핀들의 상태를 지정 또는 읽어 오는 Function들이 구현된다. Memory Control Block은 JTAG Port Interface를 통하여 외부의 메모리를 제어하기 위해 읽기 또는 쓰기 동작을 하기 위해서 그때마다의 필요한 각각의 어드레스 및 데이터 그리고 컨트롤 신호들을 규정하여 구현

한다. 마지막으로 Flash Control Block은 Flash Memory와 관련된 Function들을 구현하게 된다. 자! 그럼 실제의 구현된 SW를 예로 들어 세부내용들을 살펴해보도록 하자. 필자는 SJF2410이라는 기존 Source Code의 Block 구조를 분석하면서 설명하고자 한다. <sup>Ref</sup> <sub>Line</sub>

## 1. JTAG Port Interface Block

```
#define LPT1 0x378
```

사용할 프린터포트를 정의하는 부분

```
#define OutputPpt(value)      _outp((unsigned short)LPT1,value)
#define InputPpt()           _inp((unsigned short)(LPT1+0x1))
```

프린터 포트에 입,출력을 할수 있는 매크로함수 정의

```
// Pin Connections
// TCK :DATA[0]          (2)
// TDI :DATA[1]          (3)
// TMS :DATA[2]          (4)
// TDO :STATUS[7]       (11)
#define TCK_H            0x01
#define TDI_H            0x02
#define TMS_H            0x04
#define TCK_L            0x00
#define TDI_L            0x00
#define TMS_L            0x00
```

TAP 각 핀들과 프린터 포트와의 연결을 정의

```
#define JTAG_SET(value)      OutputPpt(value)
#define JTAG_GET_TDO()      ((InputPpt())&(1<<7)) ? LOW:HIGH )
```

PC측에서 볼 때 TCK,TDI,TMS는 출력, TDO는 입력 PIN으로 규정되며 TDO는 특히 STATUS port를 사용하여 읽게 된다.

```
void JTAG_Reset(void)
{
    int i;
    for(i=0;i<6;i++)
    {
        JTAG_SET(TDI_H|TMS_H|TCK_L);JTAG_DELAY();
        JTAG_SET(TDI_H|TMS_H|TCK_H);JTAG_DELAY();
    }
}
```

JTAG Reset은 현재 State가 어떤 상태에 있더라도 TMS를 HIGH로 인가한 채 TCK에 6 clock을 인가하면 Test-Logic-Reset 상태에 놓이게 된다.

```
void JTAG_RunTestIdleState( void )
```

```

{
    JTAG_Reset();
    JTAG_SET(TDI_H|TMS_L|TCK_L);JTAG_DELAY();
    JTAG_SET(TDI_H|TMS_L|TCK_H);JTAG_DELAY();
}

```

Run-Test/Idle 상태는 Test-Logic-Rest 상태에서 TMS를 LOW, TCK에 1Clock을 인가하면 만들 수가 있으며 Test Mode로의 진입을 하게 된다.

```

void JTAG_ShiftIRState(char *wrIR)
{
    int size;
    int i;
    int tdi;

    JTAG_SET(TDI_H|TMS_H|TCK_L);JTAG_DELAY(); // Select-DR-Scan
    JTAG_SET(TDI_H|TMS_H|TCK_H);JTAG_DELAY();
    JTAG_SET(TDI_H|TMS_H|TCK_L);JTAG_DELAY(); //Select-IR-Scan
    JTAG_SET(TDI_H|TMS_H|TCK_H);JTAG_DELAY();
    JTAG_SET(TDI_H|TMS_L|TCK_L);JTAG_DELAY(); //Capture-IR
    JTAG_SET(TDI_H|TMS_L|TCK_H);JTAG_DELAY();
    JTAG_SET(TDI_H|TMS_L|TCK_L);JTAG_DELAY(); //Shift-IR
    JTAG_SET(TDI_H|TMS_L|TCK_H);JTAG_DELAY();
    size=strlen(wrIR);

    for( i=0;i<(size-1);i++)
    {
        tdi= (wrIR[i] ==HIGH) ? TDI_H:TDI_L; //Shift-IR
        JTAG_SET(tdi|TMS_L|TCK_L);JTAG_DELAY();
        JTAG_SET(tdi|TMS_L|TCK_H);JTAG_DELAY();
    }
    tdi=(wrIR[i] ==HIGH) ? TDI_H:TDI_L; //i=3
    JTAG_SET(tdi|TMS_H|TCK_L);JTAG_DELAY(); //Exit1-IR
    JTAG_SET(tdi|TMS_H|TCK_H);JTAG_DELAY();
    JTAG_SET(TDI_H|TMS_H|TCK_L);JTAG_DELAY(); //Update-IR
    JTAG_SET(TDI_H|TMS_H|TCK_H);JTAG_DELAY();
    JTAG_SET(TDI_H|TMS_L|TCK_L);JTAG_DELAY(); //Run-Test/Idle
    JTAG_SET(TDI_H|TMS_L|TCK_H);JTAG_DELAY();
}

```

JTAG\_ShiftIRState Function은 기본적으로 Instruction Register에 명령코드를 쓰기 위한 것이며 Run-Test/Idle 상태에서부터 SelectDR-Scan,SelectIR-Scan,Capture-IR, Shift-IR로 천이 후 TDI를 통하여 해당명령 데이터를 Shift하게 된다. 그런 다음 Exit-IR,Update-IR,Run-Test/Idle 상태로 돌아오면서 해당 명령이 실행 된다.

```

void JTAG_ShiftDRState(char *wrDR, char *rdDR)
{
    int size;
    int i;
    int tdi;

```

```

JTAG_SET(TDI_H|TMS_H|TCK_L);JTAG_DELAY(); //Select-DR-Scan
JTAG_SET(TDI_H|TMS_H|TCK_H);JTAG_DELAY();
JTAG_SET(TDI_H|TMS_L|TCK_L);JTAG_DELAY(); //Capture-DR
JTAG_SET(TDI_H|TMS_L|TCK_H);JTAG_DELAY();
JTAG_SET(TDI_H|TMS_L|TCK_L);JTAG_DELAY(); //Shift-DR
JTAG_SET(TDI_H|TMS_L|TCK_H);JTAG_DELAY();
size=strlen(wrDR);

for(i=0;i<(size-1);i++)
{
    tdi=(wrDR[i]==HIGH) ? TDI_H:TDI_L;
    JTAG_SET(tdi|TMS_L|TCK_L);JTAG_DELAY();
    JTAG_SET(tdi|TMS_L|TCK_H);JTAG_DELAY(); //Shift-DR
    rdDR[i]=JTAG_GET_TDO();
}
tdi=(wrDR[i]==HIGH) ? TDI_H:TDI_L; //i=S3C2410_MAX_CELL_INDEX
JTAG_SET(tdi|TMS_H|TCK_L);JTAG_DELAY();
JTAG_SET(tdi|TMS_H|TCK_H);JTAG_DELAY(); //Exit1-DR
rdDR[i] = JTAG_GET_TDO();

JTAG_SET(TDI_H|TMS_H|TCK_L);JTAG_DELAY();
JTAG_SET(TDI_H|TMS_H|TCK_H);JTAG_DELAY(); //Update-DR

JTAG_SET(TDI_H|TMS_L|TCK_L);JTAG_DELAY(); //Run-Test/Idle
JTAG_SET(TDI_H|TMS_L|TCK_H);JTAG_DELAY(); //Update-DR
}

void JTAG_ShiftDRStateNoTdo(char *wrDR)
{
    int size;
    int i;
    int tdi;

    //Select-DR-Scan
    JTAG_SET(TDI_H|TMS_H|TCK_L);JTAG_DELAY();
    JTAG_SET(TDI_H|TMS_H|TCK_H);JTAG_DELAY();
    //Capture-DR
    JTAG_SET(TDI_H|TMS_L|TCK_L);JTAG_DELAY();
    JTAG_SET(TDI_H|TMS_L|TCK_H);JTAG_DELAY();
    //Shift-DR
    JTAG_SET(TDI_H|TMS_L|TCK_L);JTAG_DELAY();
    JTAG_SET(TDI_H|TMS_L|TCK_H);JTAG_DELAY();

    size=strlen(wrDR);

```

```

for(i=0;i<(size-1);i++)
{
    tdi=(wrDR[i]==HIGH) ? TDI_H:TDI_L;
    JTAG_SET(tdi|TMS_L|TCK_L);JTAG_DELAY();
    JTAG_SET(tdi|TMS_L|TCK_H);JTAG_DELAY(); //Shift-DR
}
tdi=(wrDR[i]==HIGH) ? TDI_H:TDI_L; //i=S3C2410_MAX_CELL_INDEX
JTAG_SET(tdi|TMS_H|TCK_L);JTAG_DELAY();
JTAG_SET(tdi|TMS_H|TCK_H);JTAG_DELAY(); //Exit1-DR

JTAG_SET(TDI_H|TMS_H|TCK_L);JTAG_DELAY();
JTAG_SET(TDI_H|TMS_H|TCK_H);JTAG_DELAY(); //Update-DR

JTAG_SET(TDI_H|TMS_L|TCK_L);JTAG_DELAY(); //Run-Test/Idle
JTAG_SET(TDI_H|TMS_L|TCK_H);JTAG_DELAY(); //Update-DR
}

```

JTAG\_ShiftDRState와 JTAG\_ShiftDRStateNoTdo 두 개의 Function은 기본적으로 JTAG\_ShiftIRState Function에 의해 규정된 Data Path로 각각 Data Write와 Data Read 기능을 한다. 여기서 JTAG\_ShiftIRState, JTAG\_ShiftDRState, JTAG\_ShiftDRStateNoTdo Function을 사용할 경우 주의해야 할 점은 Shift되는 Instruction과 Data의 길이이다.

S3C2410의 경우 Instruction은 4bit이다. 다시 말해서 JTAG\_ShiftIRState에서 Shift 길이는 4라는 뜻이다. 그리고 JTAG\_ShiftIRState로 Shift되는 명령코드가 EXTEST인 경우 JTAG\_ShiftDRState, JTAG\_ShiftDRStateNoTdo에서 Shift되어야 할 길이는 S3C2410의 Boundary Scan Cell의 개수가 된다.

## 2. Memory Control Block

```

char outCellValue[S2410_MAX_CELL_INDEX+2];
char inCellValue[S2410_MAX_CELL_INDEX+2];
int dataOutCellIndex[32];
int dataInCellIndex[32];
int addrCellIndex[27];

```

**전체 Boundary Scan Cell의 개수에 해당하는 IN, OUT용 Array 선언**  
Boundary Scan Cell 중 DATA 및 ADDRESS에 PIN의 인덱스를 저장할 수 있는 Array 선언

```

void S2410_SetPin(int index, char value)
{
    outCellValue[index] = value;
}

```

```
char S2410_GetPin(int index)
{
    return inCellValue[index];
}
```

S2410\_SetPin, S2410\_GetPin은 BSC Array 중 인덱스에 해당 하는 곳에 Data를 설정 또는 설정된 값을 얻는다.

```
void S2410_SetAddr(U32 addr)
{
    int i;

    for(i=0; i<=26; i++)
    {
        if(addr & (1<<i))
            outCellValue[addrCellIndex[i]]=HIGH;
        else
            outCellValue[addrCellIndex[i]]=LOW;
    }
}
```

BSC Array 중 Address에 해당하는 index에 Address 설정

```
void S2410_SetDataHW(U16 data)
{
    int i;

    for(i=0; i<16; i++)
    {
        if(data & (1<<i))
            outCellValue[dataOutCellIndex[i]]=HIGH;
        else
            outCellValue[dataOutCellIndex[i]]=LOW;
    }
}
```

```
U16 S2410_GetDataHW(void)
{
    int i;
    U16 data=0;

    for(i=0; i<16; i++)
    {
        if(inCellValue[dataInCellIndex[i]]==HIGH)
        {
```

```

        data = (data | (1<<i));
    }
}
return data;
}

```

S2410\_SetDataHW,S2410\_GetDataHW는 BSC Array 중 Data에 해당하는 index에 half word 즉, 16bit 폭의 data를 설정하거나 읽는다.

```

//*****
// Fast Version
// Rd16Q: fast version by removing nGCS_to_nWEnOE setup time.
// Wr16QQ: more fast version by omitting nGCS deassertion.
//*****
U16 MRW_Rd16Q(U32 addr,int en_nBE,U32 bs)
{
    S2410_SetPin(DATA0_7_CON ,HIGH);
    S2410_SetPin(DATA8_15_CON ,HIGH);
    S2410_SetPin(DATA16_23_CON ,HIGH);
    S2410_SetPin(DATA24_31_CON ,HIGH);
    S2410_SetAddr(addr);
    S2410_Assert_nGCS(addr);
    S2410_SetPin(nOE,LOW);
    if(en_nBE)
    {
        if(bs&(1<<0))S2410_SetPin(nBE0,LOW);
        if(bs&(1<<1))S2410_SetPin(nBE1 ,LOW);
    }
    JTAG_ShiftDRStateNoTdo(outCellValue);

    S2410_SetPin(nOE,HIGH);
    S2410_Deassert_nGCS(addr);
    if(en_nBE)
    {
        S2410_SetPin(nBE0,HIGH);
        S2410_SetPin(nBE1 ,HIGH);
    }
    JTAG_ShiftDRState(outCellValue,inCellValue);
    return S2410_GetDataHW();
}

void MRW_Wr16QQ(U32 addr,U16 data,int en_nBE,U32 bs) // Very fast version nGCS is never inactive for speed-up.

{
    S2410_SetPin(DATA0_7_CON ,LOW);

```

```

S2410_SetPin(DATA8_15_CON ,LOW);
S2410_SetPin(DATA16_23_CON ,LOW);
S2410_SetPin(DATA24_31_CON ,LOW);
S2410_SetAddr(addr);
S2410_Assert_nGCS(addr);
S2410_SetPin(nWE,LOW);
if(bs&(1<<0))S2410_SetPin(nBE0,LOW); //used as nWBE/nBE
if(bs&(1<<1))S2410_SetPin(nBE1,LOW);
S2410_SetDataHW(data);
JTAG_ShiftDRStateNoTdo(outCellValue);

S2410_SetPin(nWE,HIGH);
if(!len_nBE)
{
    S2410_SetPin(nBE0,HIGH); //nWBE is deasserted here.
    S2410_SetPin(nBE1,HIGH);
}
JTAG_ShiftDRStateNoTdo(outCellValue);
}

```

MRW\_Wr16QQ,MRW\_Rd16Q은 Memory Operation을 하기 위해 필요한 Address,Data및 기타 Control 신호에 해당되는 값들을 BSC Array에 설정하고 이를 JTAG\_ShiftDRState, JTAG\_ShiftDRStateNoTdo Function을 통해 실제 PIN에 인가 또는 PIN으로부터 상태를 읽어온다.

### 3. Flash Control Block

```

#define _WR(addr,data) MRW_Wr16QQ(addr<<1,data,0,0x3)
#define _RD(addr) MRW_Rd16Q(addr<<1,0,0x3)
#define _RESET() _WR(0x0,0xf0f0)
#define BADDR2WADDR(addr) (addr>>1)

```

Flash Memory Operation을 위한 기본적인 Command 입출력에 관한 매크로 함수로서 Memory Control Block에서 정의되었던 Function들을 이용하게 된다.

```

static int AM29F800_CheckId(void)
{
    U16 manId,devId;

    _RESET();

    _WR(0x555,0xaaaa);
    _WR(0x2aa,0x5555);
    _WR(0x555,0x9090);
}

```

```

manId=_RD(0x0);

_RESET(); // New 5V AM29F800 needs this command.
_WR(0x555,0xaaaa);
_WR(0x2aa,0x5555);
_WR(0x555,0x9090);
devId=_RD(0x1);

_RESET();

printf("Manufacture ID=%4x(0x0001), Device ID(0x225B)=%4x%Wn",manId,devId);

if(manId==0x0001 && devId==0x225b)return 1;
else return 0;
}
Flash Memory의 ID Code를 읽어온다.

```

```

void AM29F800_EraseSector(int targetAddr)
{
    printf("Sector Erase is started!%Wn");

    _RESET();

    _WR(0x555,0xaaaa);
    _WR(0x2aa,0x5555);
    _WR(0x555,0x8080);
    _WR(0x555,0xaaaa);
    _WR(0x2aa,0x5555);
    _WR(BADDR2WADDR(targetAddr),0x3030);

    _WAIT();

    _RESET();
}

```

Flash Memory는 특성상 임의의 데이터를 써넣기 위해서는 반드시 지우는 작업이 필요하다. 지울 때는 Block별 Erase가 기본이 된다. 따라서 Erase할 때 주어지는 Base Address는 각 Block의 시작번지이어야만 된다. Device에 따라 Chip 전체를 지우는 명령을 제공하는 것도 있다.

```

int AM29F800_ProgFlash(U32 realAddr,U16 data)
{
    _WR(0x555,0xaaaa);
    _WR(0x2aa,0x5555);
    _WR(0x555,0xa0a0);

    _WR(BADDR2WADDR(realAddr),data);
    //return _WAIT(); //not needed at JTAG access
    return 1;
}

```