AVR - HW2

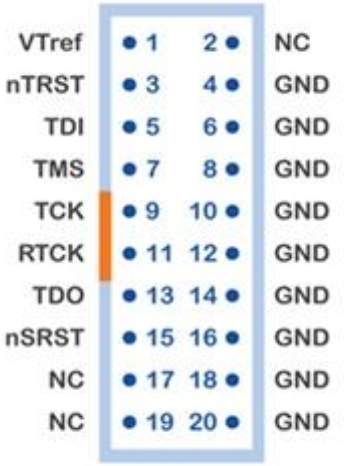
임베디드스쿨1기
Lv1과정
2020. 09. 04
박성환

1. JTAG

1. JTAG 용도

1) 디버깅용

- 하드웨어 시피유 브레이크하거나 러닝하면서 개발자가 직접 시피유를 조절하며 디버깅 가능
- 디버깅하기 위한 장비 비용과 사용법을 익혀야 하는 단점이 있으나 개발 편리성 및 개발 기간 단축에 좋음
- 제이텍 포트 5핀 + 3핀 연결해서 보통 14핀 20핀으로 구성



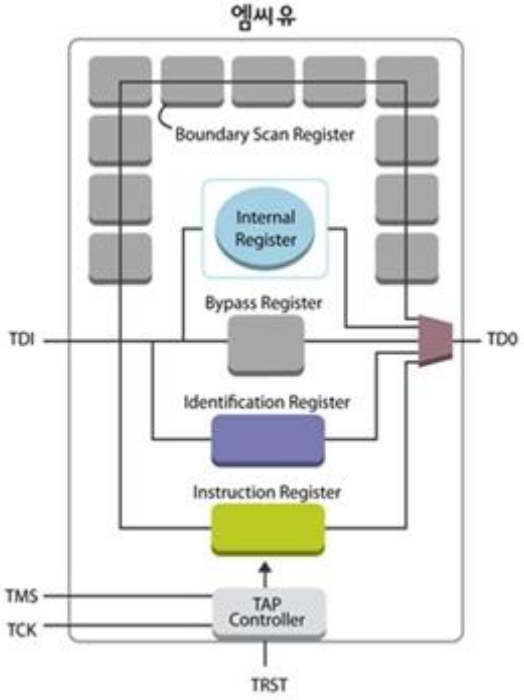
2) 플래시 다운로더용

컴파일한 바이너리 데이터를 플래시 메모리에 다운로드하기 위함

2. JTAG 하드웨어 구성

TDI : 테스트 하기 위한 신호
TDO : 테스트 결과 외부 모니터링 하기 위한 신호
TMS : 테스트 모드 전환선택
TCK : 클럭
TRST : 제이텍 리셋

이 신호들 조합으로 개발자가 소스레벨 디버깅 하거나 플래시 메모리에 다운로드



3. JTAG 활용 점검

-Level 1 : 보드가 죽었을때 Download 용으로 만 사용한다.
-Level 2 : Break / Step 으로 Source가 여기를 지나가는지
안지나가는지만 확인하는 용도로 사용한다.
-Level 3 : 전역변수 및 지역변수의 값을 확인 하는 용도로 사용한다.
-Level 4 : Assembler를 보고 CPU Register가 의미하는 값을 알고
Debugging을 할 수 있는 용도로 사용한다.
-Level 5 : Task의 Stack을 열어서 어떤 함수가 실행되었으며 어떤값이
들어 있는지 확인 하는 용도로 사용한다.
-Level 6 : Exception(Reset/Abort/MMU/)등 및 기타 JTAG의
지원기능을 이용한 Debugging 용도로 사용한다.

2. ELF HEX BIN

1. **ELF(Executable and Linking Format)**
   - 실행 파일, 개체 코드 및 공유 라이브러리의 일반적인 표준 형식으로 컴파일러의 출력으로 AVR-GCC에 의해 생성
   - 디버깅 정보(변수 이름, 함수 이름, 줄 번호 등)과 같이 실행 불가능한 추가 데이터가 많이 포함되어 있음
   - 플래시에 프로그래밍되어 있지 않으며 ELF 파일에서 코드를 추출해서 HEX로 만들어 플래시에 적재하는 것

2. **HEX**
   - AVR 프로세서 실제 적용하는 파일로 순수 코드로 이루어져 있음

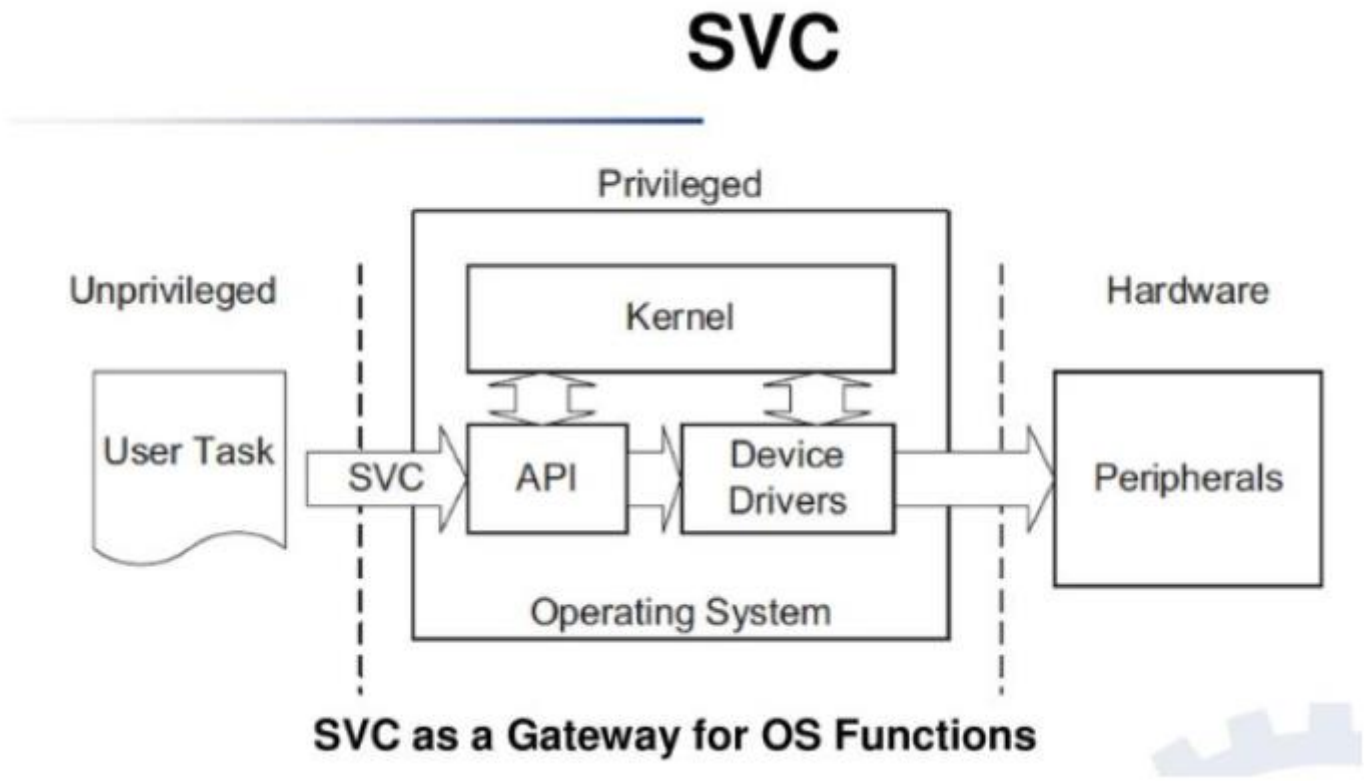
3. **Bin 과 Hex 차이**
   - Bin : 말그대로 0과 1로 이루어져 있음
   - Hex : Bin파일을 Hex로 바꾼 후, 이것을 사람이 볼 수 있는 ASCII 코드로 변환한 파일

   - 차이 : **hex 파일에 프로그램 시작 주소등의 정보 등이 담겨져 있음, 그냥 프로그램 하면 됨**
     Bin 파일은 시작주소 정보가 없기 때문에 Writing시 시작주소를 입력해줘야 함(Flash Program 생각)

3. SW Interrupt (SWI)

1. SVC(SuperVisor Call)
-User mode(=application, unprivilleged)에서 직접 접근을 막고 SVC를 통해서 **Supervisor(=os, privilledged) mode**로 접근해서 중요한 데이터 접근이나 드라이버 변경이 이루어지도록 내부적으로 호출하는 일종의 인터럽트

2. SVC 인터럽트 발생 경우
- 사용자가 의도적으로 SVC 명령을 써서 호출한 경우
- 복잡한 입출력 처리를 해야하는 경우
- 기억장치 할당 및 오퍼레이터와 대화를 해야하는 경우



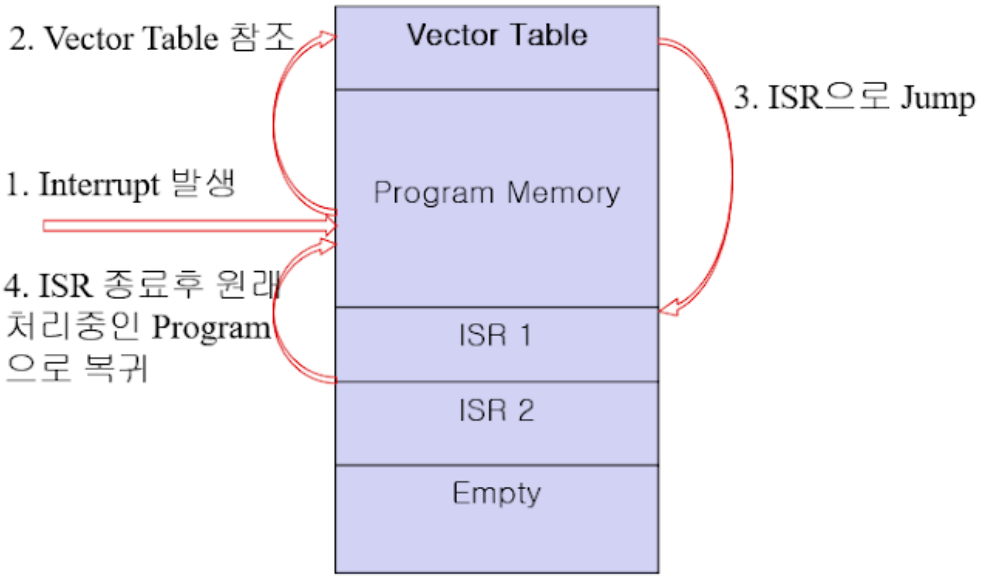
3. HW Interrupt

외부 인터럽트

외부 인터럽트는 다음과 같이 입·출력장치, 타이밍 장치, 전원 등 외부적인 요인에 의해 발생한다.

- 전원 이상 인터럽트(Power Failed Interrupt) : 정전이 되거나 전원 이상이 있는 경우
- 기계 착오 인터럽트(Machine Check Interrupt) : CPU의 기능적인 오류 동작이 발생한 경우
- 외부 신호 인터럽트(External Interrupt)
  - 타이머에 의해 규정된 시간(Time Slice)을 알리는 경우
  - 키보드로 인터럽트 키를 누른 경우
  - 외부 장치로부터 인터럽트 요청이 있는 경우
- 입·출력 인터럽트(I/O Interrupt)
  - 입·출력 데이터의 오류나 이상 현상이 발생한 경우
  - 입·출력장치가 데이터의 전송을 요구하거나 전송이 끝났음을 알릴 경우

인터럽트 처리 과정



4. ATmega328P 외부인터럽트 레지스터

① SREG(Status Register)

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- MCU의 현 상태 및 최근 수치 명령 실행에 대한 결과를 포함한다.
- 상태레지스터는 모든 ALU 연산을 수행 후 갱신된다.
- Bit7. I (Global Interrupt Enable) : 모든 인터럽트 활성화 비트

② EIMSK(External Interrupt Mask Register)

Bit	7	6	5	4	3	2	1	0	
	INT7	INT6	INT5	INT4	INT3	INT2	INT1	IINT0	EIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Bit 7..0 – INT 7~0 Q 비트를 SET(1) 시키면 해당 외부 인터럽트 핀이 활성화 된다.
- 단, SREG의 I비트가 1로 SET된 상태여야 한다.

③ EICRA(External Interrupt Control Register A)

Bit	7	6	5	4	3	2	1	0	
	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- 외부 인터럽트의 트리거 방식을 LOW / 상승엣지 / 하강엣지 중에 선택하는 레지스터

ISCn1	ISCn0	설명
0	0	INTn의 Low 신호시에 일반적인 인터럽트 요청을 한다.
0	1	(reserved)
1	0	INTn의 하강엣지 신호시에 일반 비동기적인 인터럽트를 요청한다
1	1	INTn의 상승엣지 신호시에 일반 비동기적인 인터럽트를 요청한다

④ EICRB(External Interrupt Control Register B)

Bit	7	6	5	4	3	2	1	0	
	ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40	EICRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- 외부 인터럽트의 트리거 방식을 LOW / 상승엣지 / 하강엣지 중에 선택하는 레지스터

ISCn1	ISCn0	설명
0	0	INTn의 Low 신호시에 일반적인 인터럽트 요청을 한다.
0	1	어떠한 신호의 변화에서든 일반적인 인터럽트 요청을 한다.
1	0	INTn의 두 샘플 신호간의 하강엣지 신호시에 일반적인 인터럽트를 요청한다
1	1	INTn의 두 샘플 신호간의 상승엣지 신호시에 일반적인 인터럽트를 요청한다

⑤ EIFR(External Interrupt Flag Register)

Bit	7	6	5	4	3	2	1	0	
	INTF7	INTF6	INTF5	INTF4	INTF3	INTF2	INTF1	IINTF0	EIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Bit 7..0 – 외/내부로부터 INT7:0 핀에 인터럽트가 요청되면 해당 비트가 Set(1) 된다. 그 후에 인터럽트 루틴이 실행될 때 해당 비트가 Clear(0)된다.

5. Review (AVR – LED_ON_OFF(1))

```
 * Created: 2020-09-15 오후 9:42:31
 * Author : Gone
 */

#define F_CPU 16000000L //io.h 먼저 써야함 내부에 F_CPU가 선언되어 있어서, L이 literal 상수다 알려주는 의미
#include <avr/io.h>
#include <util/delay.h>

/* 이미 함수의 정의되어 있는 내용을 사용함*/
#if 0
int main(void)
{
    /* Replace with your application code */
    while (1)
    {
        DDRB |= (0x1 << PINB1);
        //DDRB = 0x02;

        PORTB |= (0x1 << PINB1);
        //PORTB = 0x02;
        _delay_ms(1000);

        PORTB &= ~(0x1 << PINB1);
        //PORTB = 0x00;
        _delay_ms(1000);
    }
}
#endif
```

5. Review (AVR – LED_ON_OFF(2))

***Datasheet와 Struct 활용하여 직접 Register 할당하기**

```
/* Datasheet와 구조체를 이용해서 직접 Register 할당하기 */
#if 1
#define PORTB_REG 0x23 //Base Register, 상수

typedef struct io_port{
    unsigned char pin;
    uint8_t ddr;
    uint8_t port;
} io_port;

int main(void)
{
    io_port *portB = (void*)PORTB_REG; //상수기 때문에 주소를 쓰려면 포인터형을 케스팅함
    /* Replace with your application code */
    while (1)
    {
        portB->ddr = 0x2;
        portB->port = 0x2;
        _delay_ms(1000);
        portB->port = 0x00;
        _delay_ms(1000);
    }

}
#endif
```

감사합니다.