AVR HW 스터디

- 개요
    - 3학년 실험을 대비한 AVR 학습 및 실습

예산

- AVR 보드
- AVR Studio 4
- 케이블선

참여 인원

- Boot4AVR
    - 김지형, 양형균, 이준형, 이재영, 권희정, 이석호, 최희정

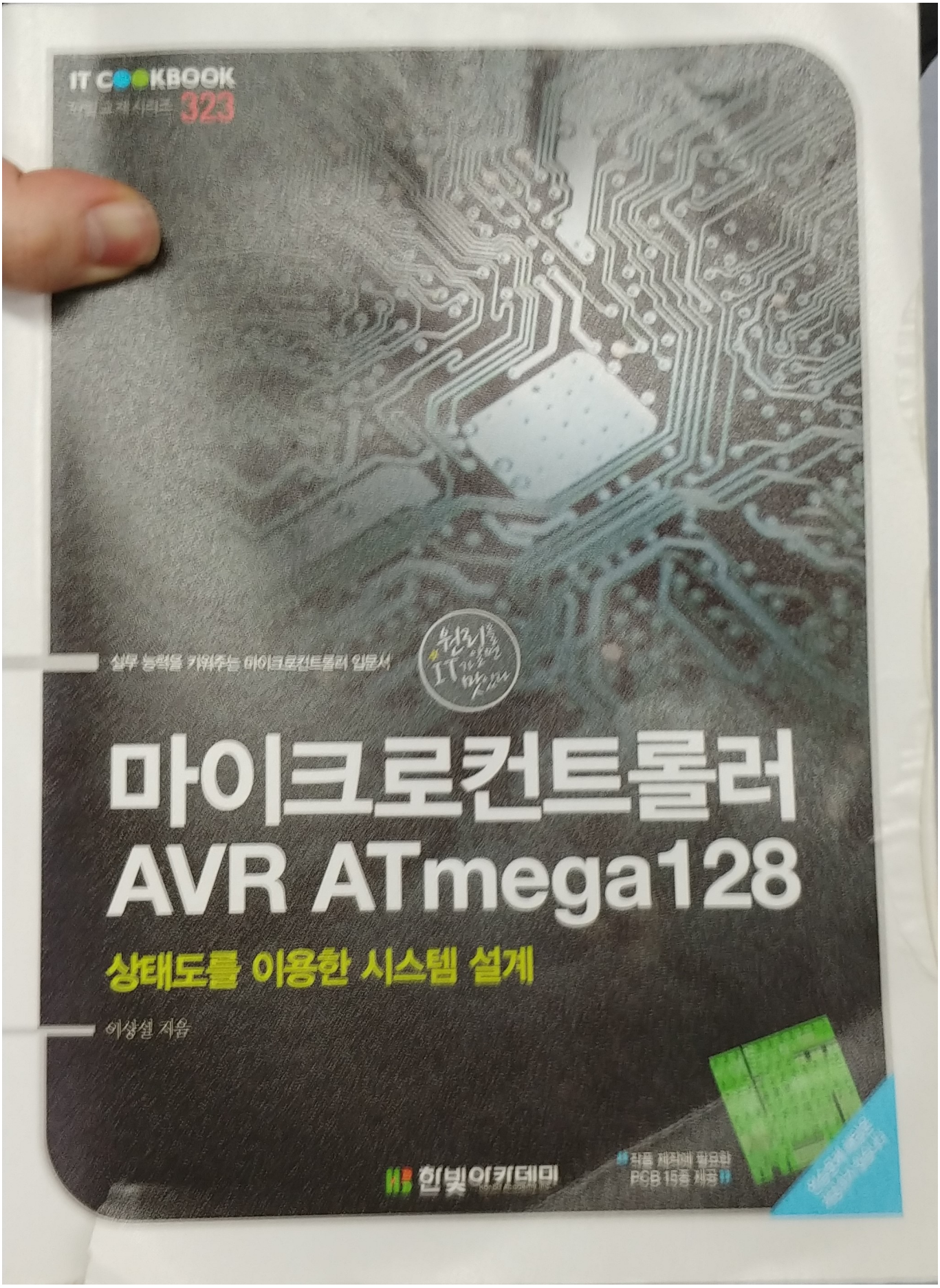
개발 환경 설정

-Software -AVR Studio 4

```
 1. 해당 [영상](https://www.youtube.com/watch?v=4LK3vprsKRM)을 따라 설치.
    조영범 교수님께서 제공해주신 AVR Studio 4 사용. 상위버전은 설정도 복잡하고 오류날
가능성도 높기 때문에 사용을 지양할 것.
 2. msys-1.0-vista64라는 파일 압축 해제 및 C:\WinAVR-20100110\utils\bin 에 붙여넣기.
```

- Hardware
    - AVR Board

추천 자료



마이크로 컨트롤러 AVR ATmega128/이상설 지음/2013년 발행

개발 과정

Day 01

1. AVR보드 LED 점등 코드 해석 및 실습
2. AVR을 활용하여 설계할 물품 토의

Day 02

1. ATmega128의 기본 구조 학습
2. 7segment를 활용한 실습

Day 03

1. Switch를 활용한 실습
2. Interupt를 활용한 타이머 제작

Day01

- 일시 : 2019.01.11 20:00 ~ 22:00
- 작성자 : 유재덕

학습 목표

1. AVR보드 LED 점등 코드 해석 및 실습
2. AVR을 활용하여 설계할 물품 토의

1. AVR보드 LED점등 코드 해석 및 실습

######example01

```
#include <avr/io.h>
#include <util/delay.h>
#define F_CPU 1000000UL

int main(void){
   unsigned int i;
   DDRF= 0xF0;      //DDRX : X reg set up
    PORTF= 0x10;    //set X reg(8bit)

   while(1){
      for(i=0; i<4;i++){
         _delay_ms(1000);
         PORTF=PORTF<<1;    //shift
      }
      PORTF=0x10;
   }
   DDRB=0x80;
   DDRF=0xF0;
   DDRG=0x00;
   PORTF=0x10;
}
```

코드 설명 Port F - 회로에서 LED를 키고 끄는것을 제어하는 역할 DDRF[0:7] - F Port init.

- 만약 DDRF[4] = 1 일때, DDRF[4]는 output이라는 의미
- 만약 DDRF[4] = 0 일때, DDRF[4]는 input이라는 의미

PORTF[0:7] - value of F Port LED 점등의 경우 상위 4bit, 즉 DDRF[0:3]과 PORTF[0:3]만 사용

ex)

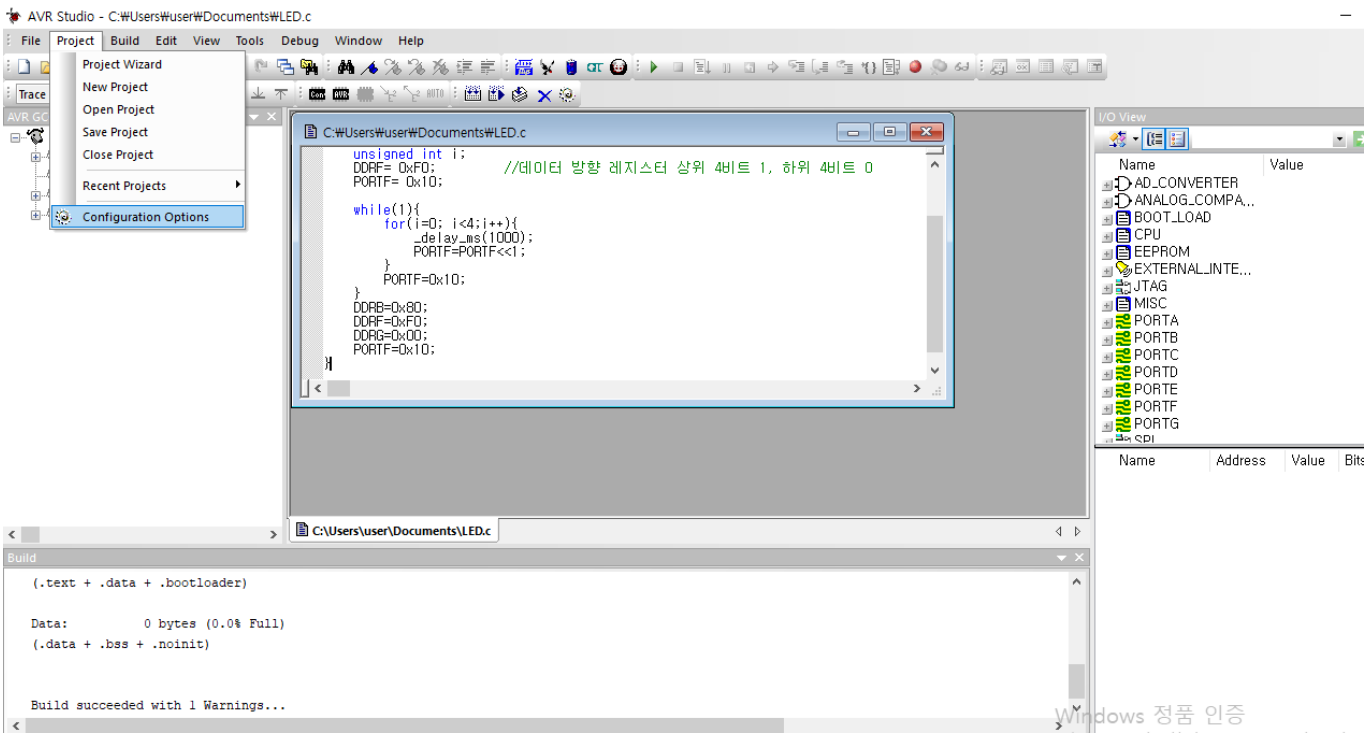
```
DDRF=0xF0;
```

[0,0,0,0,0,0,0,0] —>[1,1,1,1,0,0,0,0] 상위 4 bit를 사용한다고 메인 보드에 예고

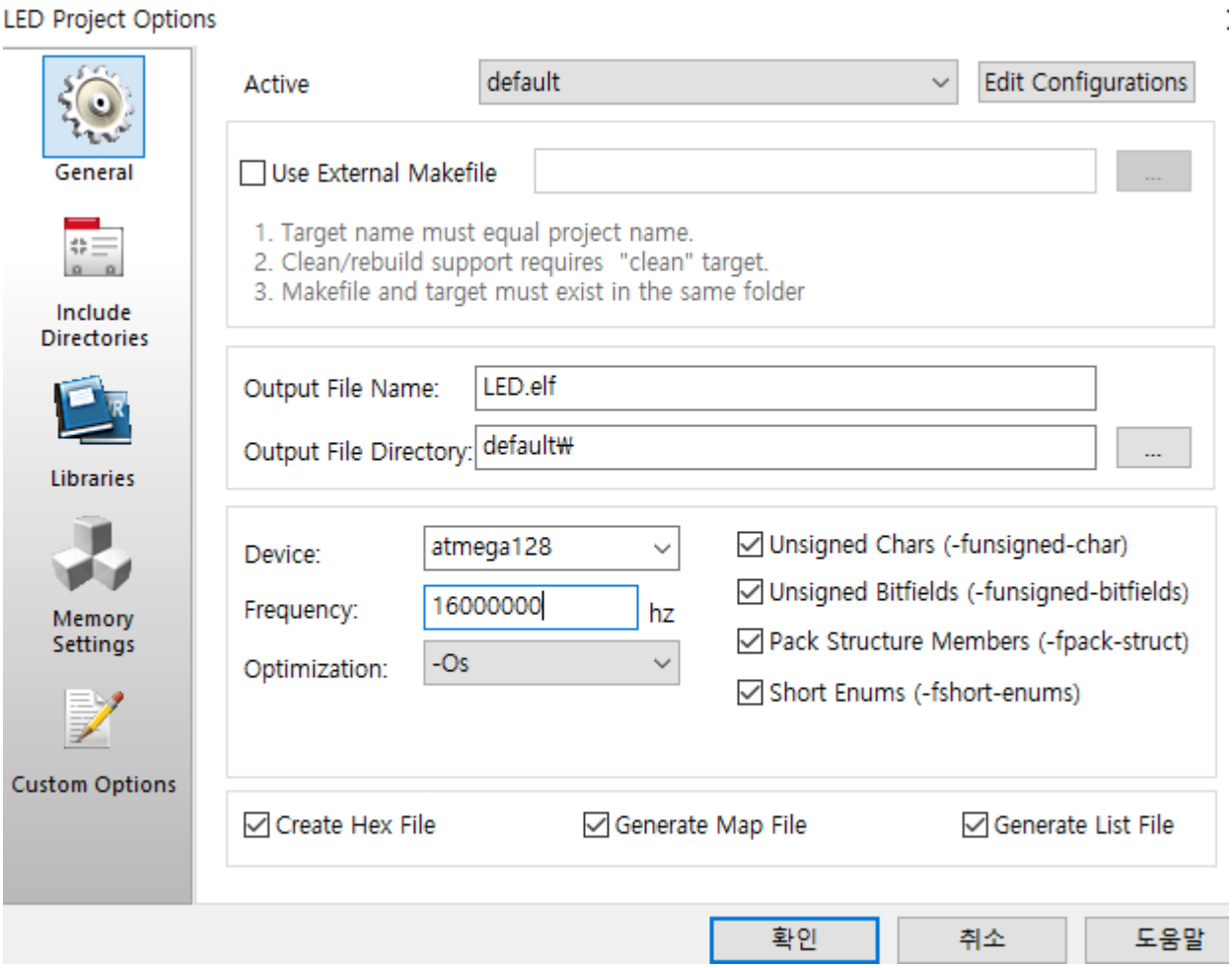
```
PORTF=0x10;
```

[0,0,0,0,0,0,0,0] —>[0,0,0,1,0,0,0,0] 해당 비트의 값을 1로만듦. 1=점등

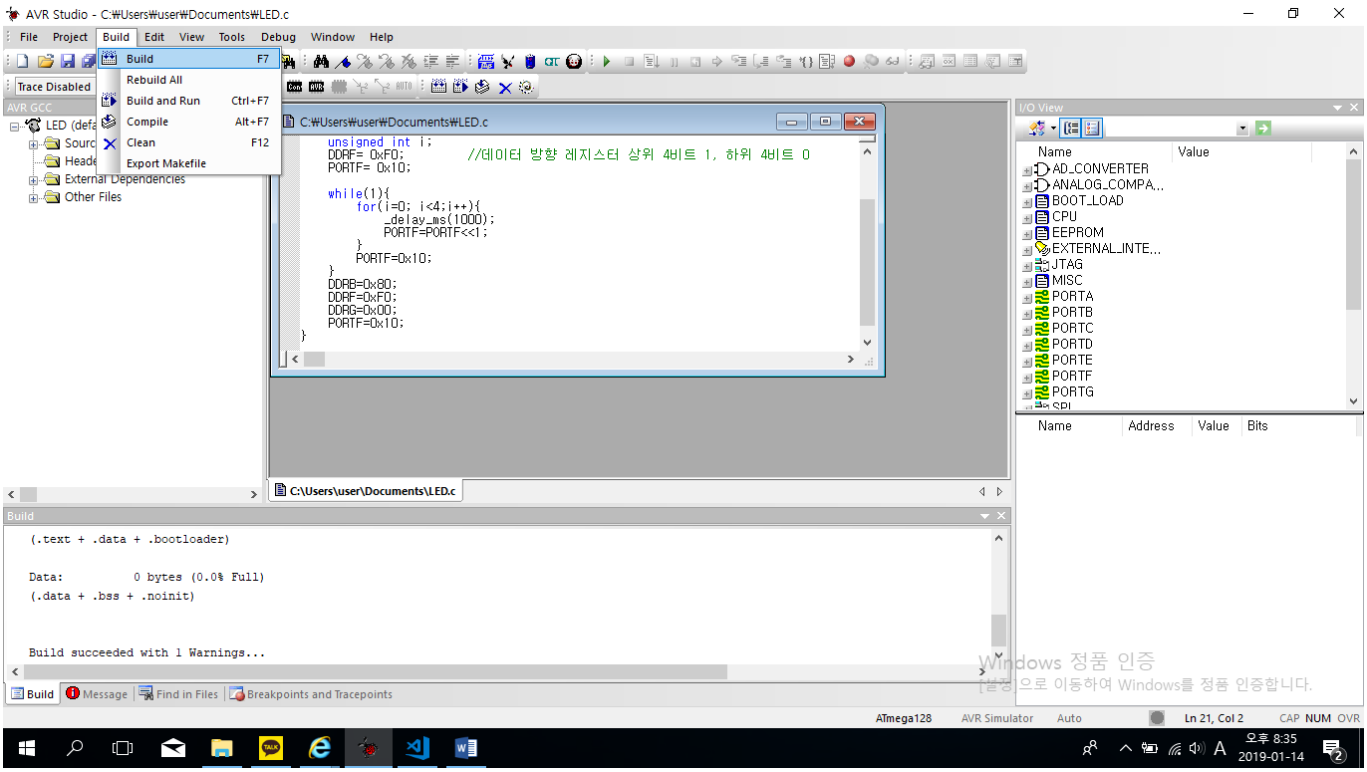
프로그램 업로드



- Click Configuration Options

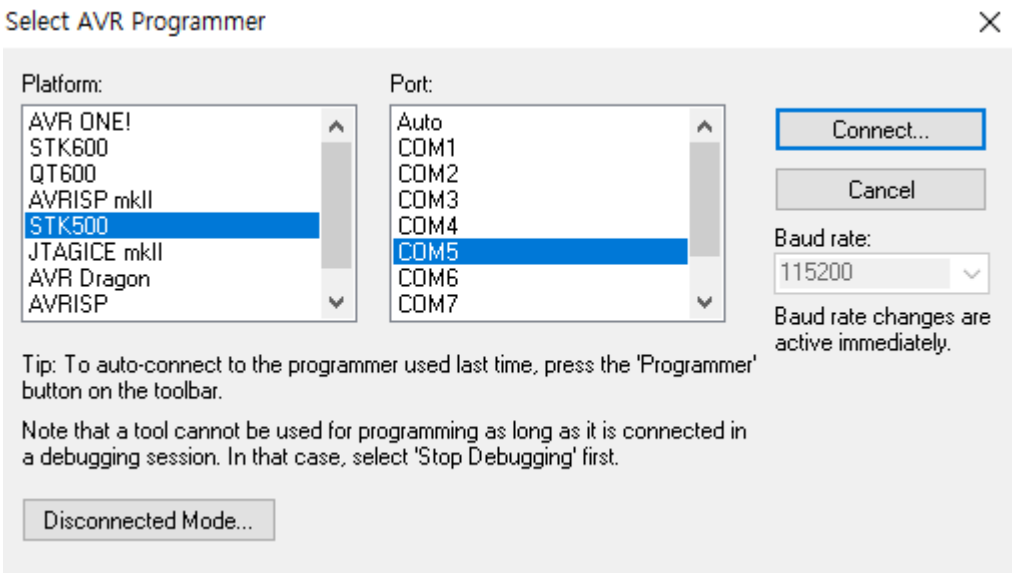


- Set frequency 16,000,000(보드마다 다름)

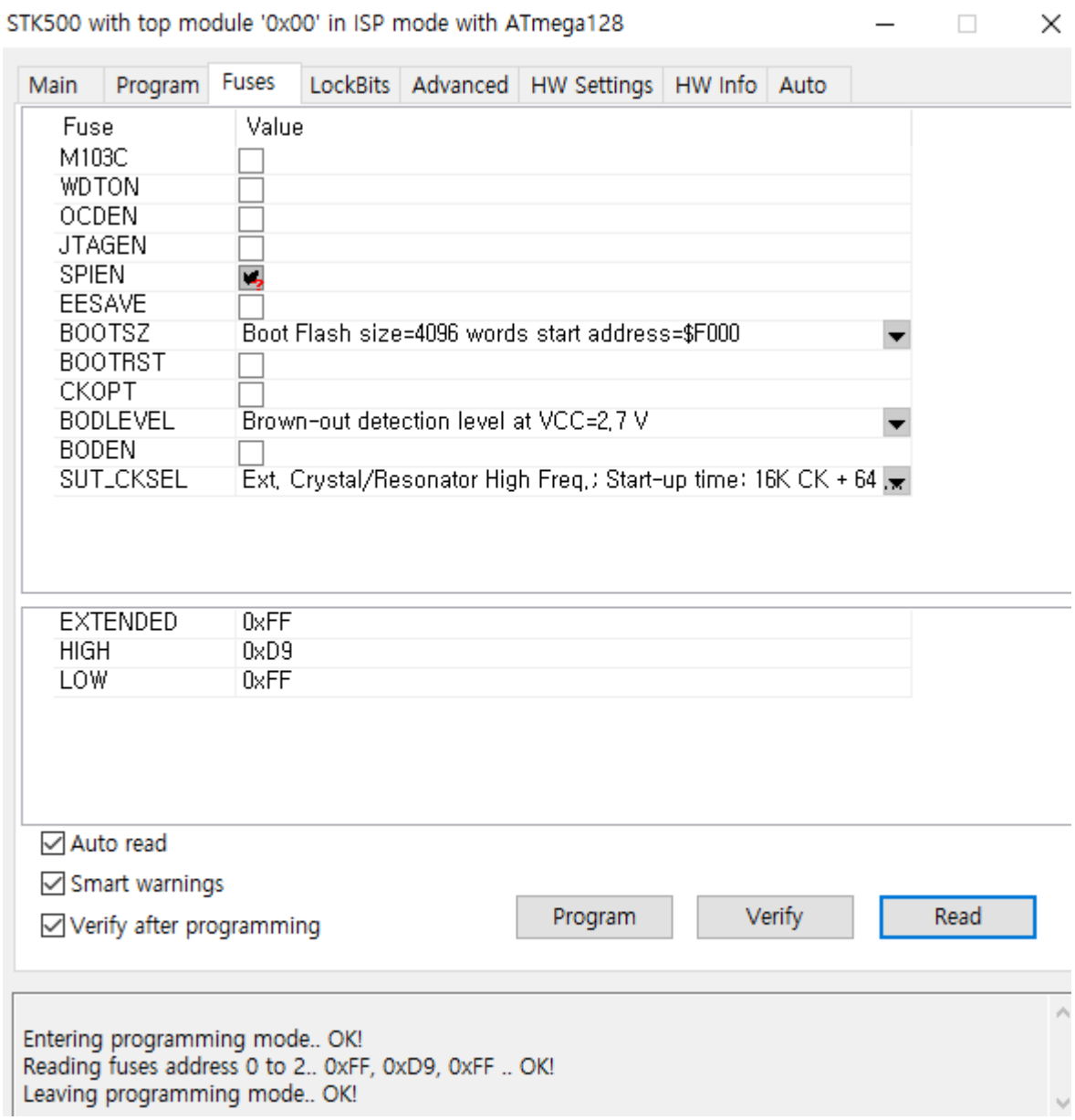


-Build program

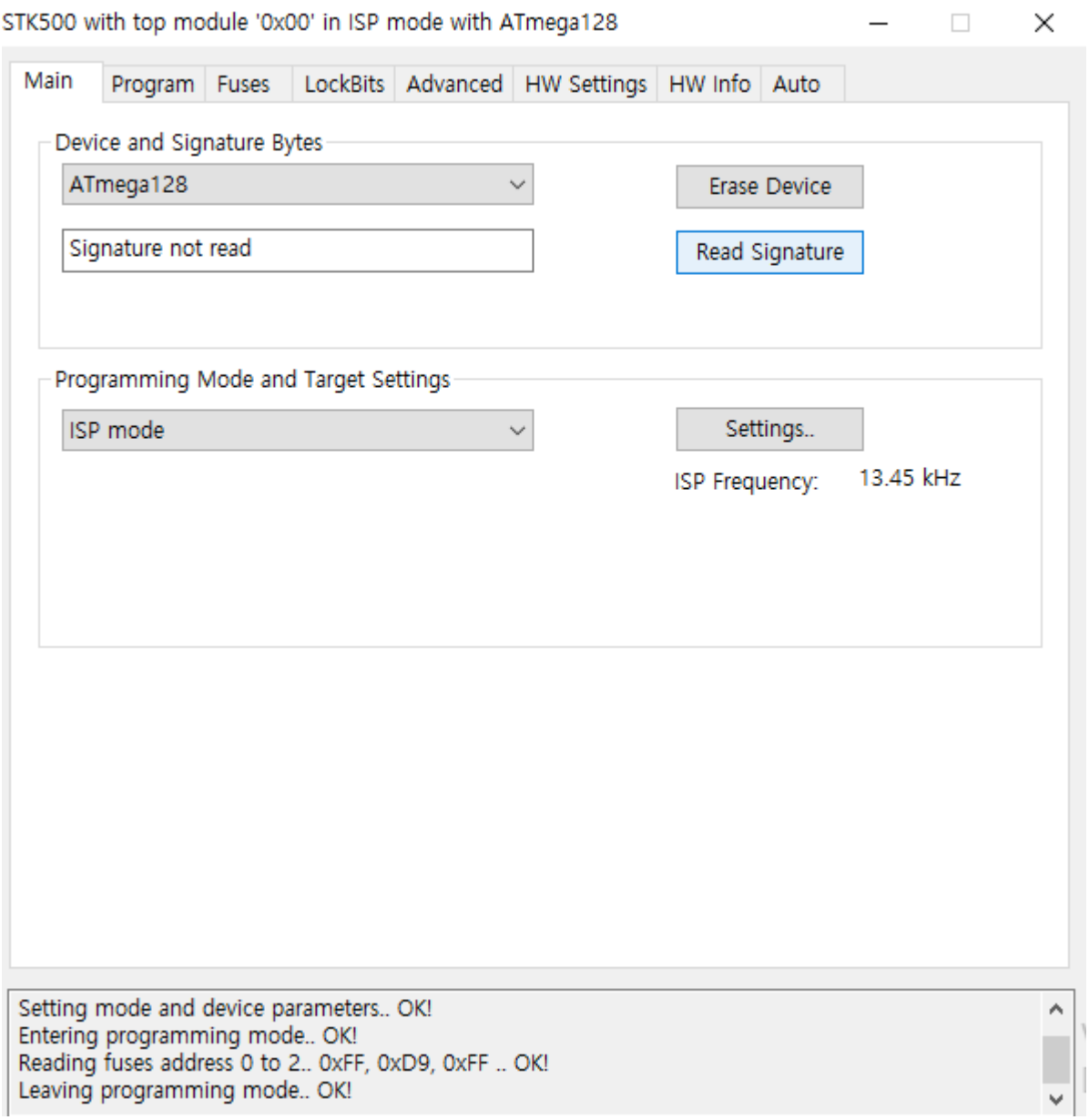
- Click Display 'Connect' Dialog



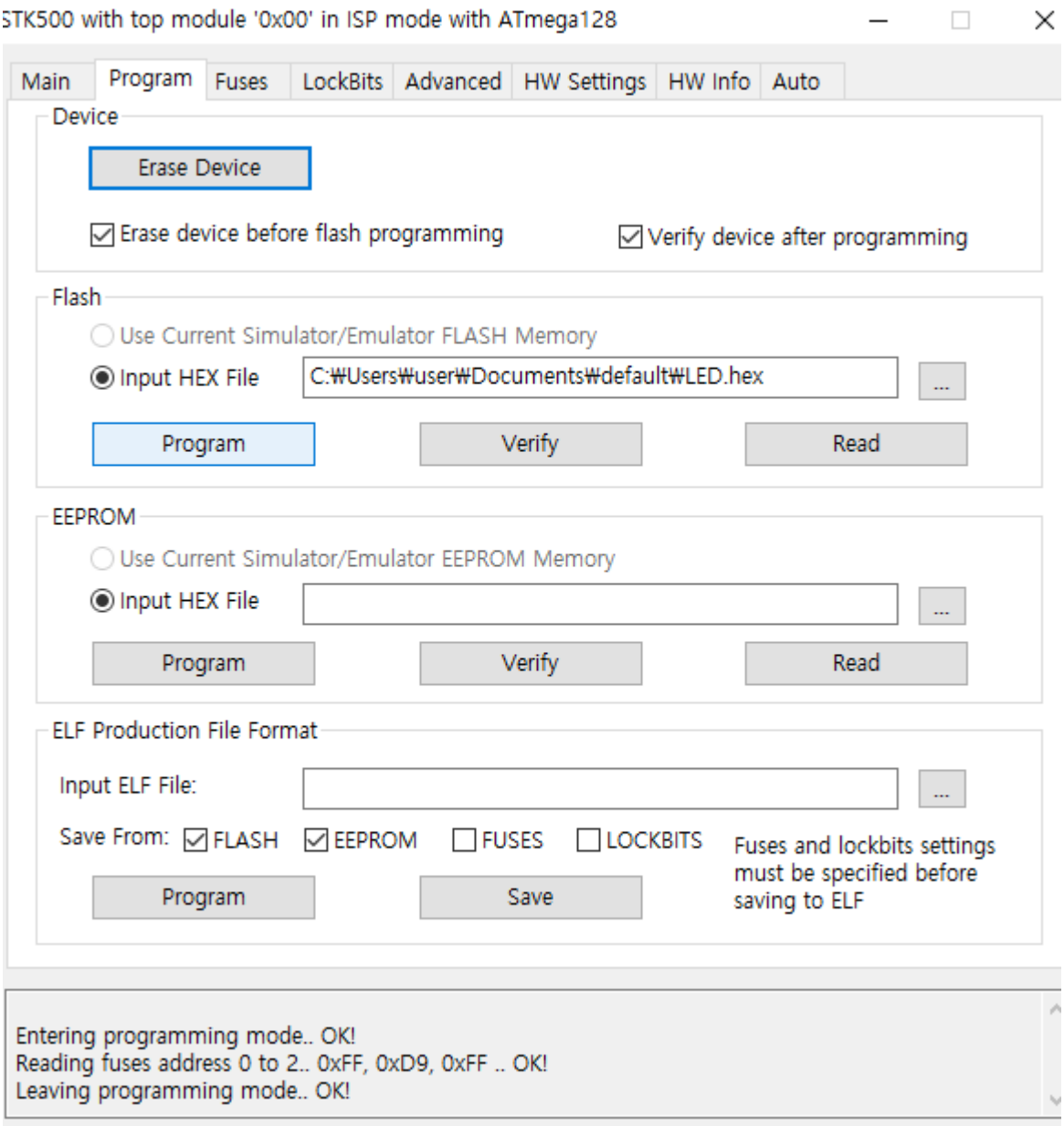
- Check connected port at Device Manager, set Platform(STK5000) and Port. Then, press Connect.



- Check fuse bit of board.



- Read Signature of board.



Set Input HEX File as coded file, press Program

토의 사항

- 예제 코드의 마지막 네 줄의 용도는 무엇인가?

Trouble Shoot

- 코드에서 설정한 delay에 비해 빠르게 점등되는 보드가 몇몇 있다. 어쨔서 이런 현상이 일어나는가?
    - AVR Studio의 기준 주파수가 있고, 보드 내의 기준 주파수가 있다. 이 둘이 서로 일치하지 않는 상황에서 코드를 업로드할 경우, 보드와 코트 사이에 충돌이 생길 수 있다.
- 몇몇 보드는 AVR Studio에서 액세스 자체를 할 수 없다. 해당 현상의 원인은 무엇인가?
    - fuse bit을 임의로 바꾼 몇몇 보드들이 임의로 바꾼 환경과 보드의 환경이 서로 어긋나면서 보드 자체가 액세스를 거부하는 것으로 추정된다.
- while 문 이후에 있는 코드의 의미는?

    - 사실상 의미 없음. 이번 학기에 수정 예정.

제작 아이디어 토의

1. 타이머
2. 적외선 레이더 서브모터와 적외선 거리 센서를 활용해 적외선 레이더 제작
3. 솔라 트레커 서브모터와 조도센서를 활용해 솔라 트레커를 제작 솔라 트레커 : 빛(광원)의 움직임에 따라 이동하는 장치
4. 적외선 리모콘 IRremote 모듈을 활용하여 적외선 리모콘 제작
5. 버스 정류장 open api 활용 버스 정류장 open api를 활용하여 보다 개선된 방법으로 사용자에게 버스 운행 정보를 제공할 수 있는 장치 제작

Day02

학습 목표

1. ATmega128의 기본 구조 학습
2. 7segment를 활용한 실습

ATmega128의 기본 구조

- 6개의 8비트 양방향 병렬 I/O 포트
    - Port A~ Port F
- 1개의 5비트 양방향 병렬 I/O 포트
    - Port G
- Port A~ Port E
    - 범용 I/O 포트로 사용될 경우 read-modify-write동작 가능. 즉, 입출력 방향 변경 없이 SBI(Set Bit) 및 CBI(Clear Bit) 명령에서 포트의 동작방향이 달라질 수 있다.
- DDRxn
    - x : Port A ~ Port G
    - n : 각 포트의 비트번호
    - 입출력 방향을 설정. 1: 출력 0: 입력
    - Read & Write
- PORTxn
    - 데이터 출력
    - Read & Write
- PINxn
    - 포트 입력
    - Read
- SFIOR (Special Function I/O Register)의 PUD
    - PUD비트를 1로 설정하면 그 기능이 금지 보다 자세한 기능이 궁금하다면 ATmega128 매뉴얼을 참고

7segment를 활용한 실습

example02

```
#define __DELAY_BACKWARD_COMPATIBLE__ //추가! 현재 컴파일러로는 _delay_ms_()에 상수
밖에 집어넣지 못한다. 이거 없으면 에러남!

#include <avr/io.h>
#include <util/delay.h>
unsigned char FND_SEG[10] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7C, 0x07, 0x7F,
0x67 }; //SEG 0 ~ 9
void init_system(void) // Port Initialization
{
DDRA = 0xFF; // Set Port F as output
DDRE = 0x0C; // PortE(3:2) = LED_CTRL, LED_DATA must be output
PORTE = 0x04; // LED_CTRL
PORTA = 0X01; // LEFT SEG ON
}
int main(void)
{
init_system(); //Port Initialization
int j; //segment variable
int k=70;
for(j = 0; j<4 ; j++)
{
PORTE = 0X04;
PORTA = 0X01 << j;
PORTE = 0X08; //LED DATA
PORTA = FND_SEG[j]; //numeral display through Port A
_delay_ms(1000); //Delay
}
while(1) //infinite loop
{
for(j = 0; j<4 ; j++)
{
PORTE = 0X04;
PORTA = 0X01 << j;
PORTE = 0X08;
PORTA = FND_SEG[j];
_delay_ms(k);
}
if(k>=2)
k=k-1;
//Delay is getting be short until 1ms.
}
return 0;
}
```

- DDR A : 사용하기 위해 초기화하는 PIN을 의미
- DDR E : Port E에서 E2, E3 핀 사용
    - PE2 : LED_CTL
    - PE3 : LED_DATA

- Port A : E3과 E4에 따라 의미하는것이 달라짐. 7 Segment Control p.27 참고
  - E2일때 : Port A는 활성화되는 segment들을 결정함
  - E3일때 : Port A는 한 segment 점등하는 LED를 의미함

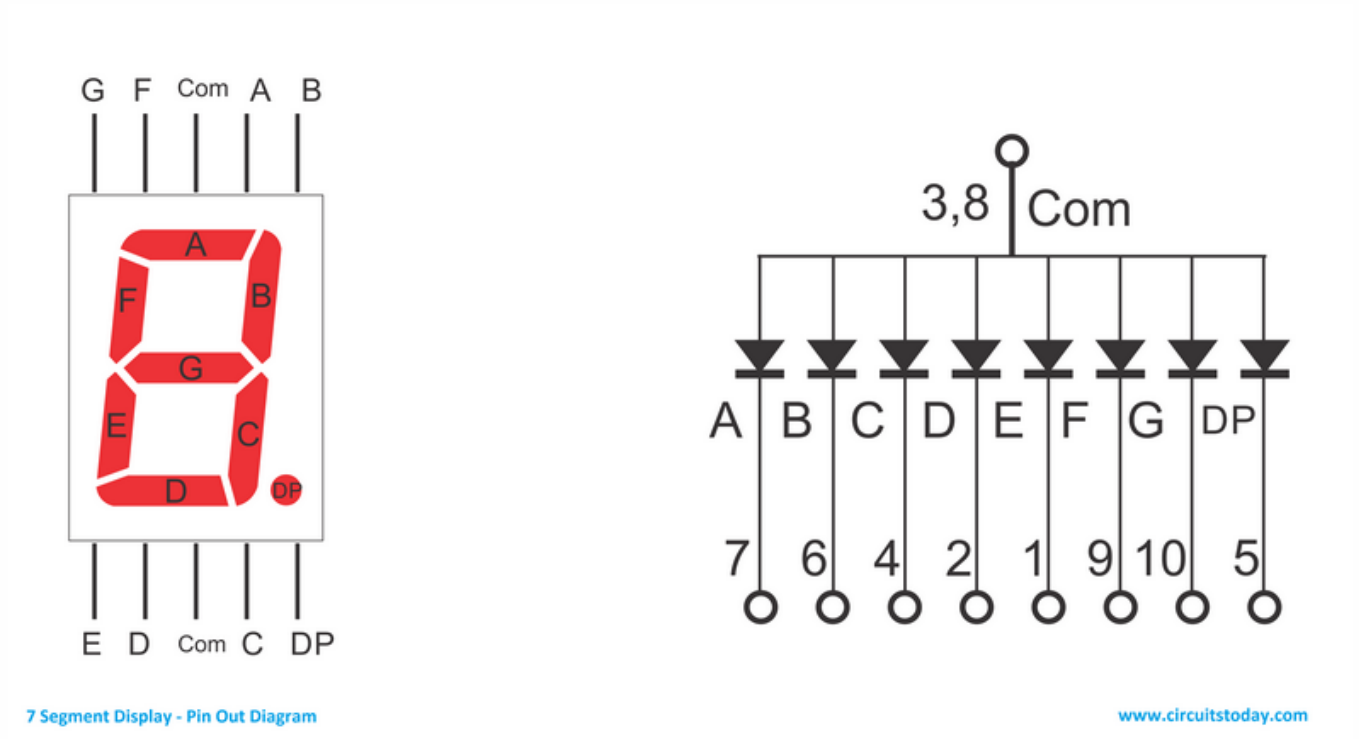
```
for(j = 0; j<4 ; j++)
{
PORTE = 0X04;
PORTA = 0X01 << j;
PORTE = 0X08; //LED DATA
PORTA = FND_SEG[j]; //numeral display through Port A
_delay_ms(1000); //Delay
}
```

Port E = 0X04; -E[0,0,0,0,0,1,0,0] : E2 = 1 -E2=1일때, Port A는 켜지는 LED를 의미한다.

Port A = 0X01 << j -A[0,0,0,0,0,0,0,1] : A1 = 1 -A를 j만큼 shift하면서 1에 해당하는 segment를 점등한다.

Port E = 0X08; -E[0,0,0,0,1,0,0,0] : E4 = 1 -E3=1일때, Port A는 LED에 나타나는 숫자를 의미한다

Port A = FND_SEG[j] -FND(Flexible Number Display)는 segment의 다른 표현 -j라는 숫자가 segment에 표기될 수 있도록, 임의의 Port A의 값들을 조정해줌.



Trouble Shooting

- 예제 코드의 #define 구문을 제외할 경우, 해당 코드는 실행되지 않는다.
  - 과거 _delay_ms() 함수의 매개변수는 변수 또한 쓰일 수 있었다. 그에 비해 최근 버전의 컴파일러는 _delay_ms() 함수의 매개변수로 상수만을 받도록 설계되었다. 결국 이는 코드가 컴파일러의 변화에 따라가지 못해 일어난 오류라 할 수 있다.

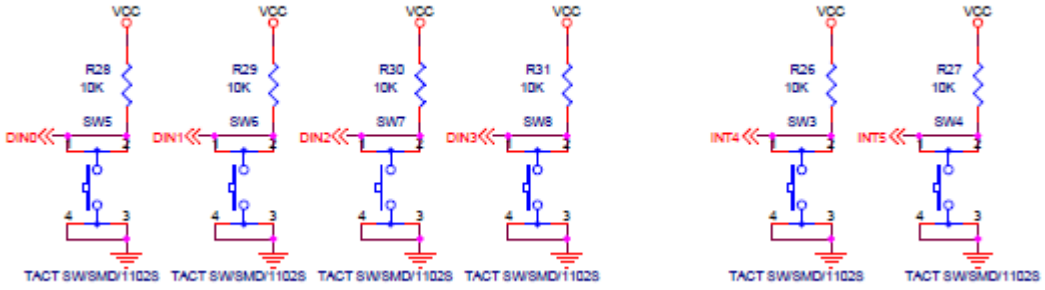
Day03

학습 목표

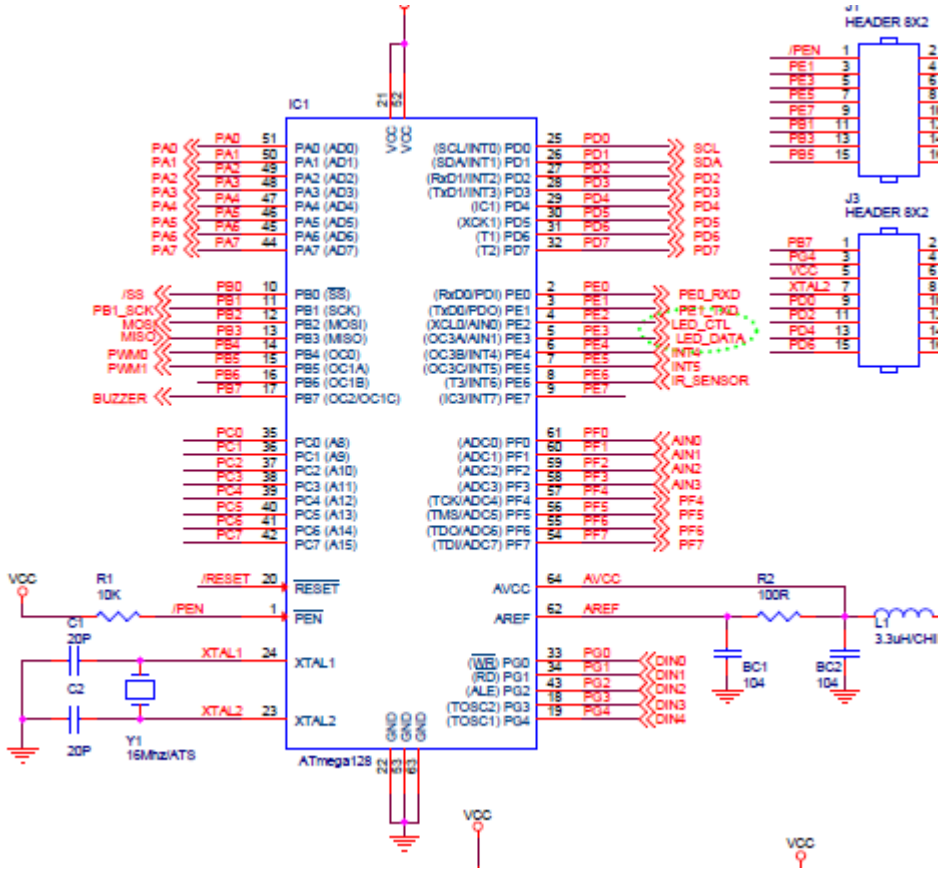
1. Switch 실습
2. Interrupt를 활용해 타이머 제작

1. Switch 실습

- AVR switch 회로



  - 스위치의 input은 DIN[0:4], INT[4:5]에 해당한다.
- AVR 메인 보드 회로



  - DIN[0:4]는 PortG[0:4]의 input에 해당한다.
- 결국, PortG[0:4]를 0으로 설정하여 특정 스위치의 input을 받아올 수 있다.

example

```
#include <avr/io.h>
#include <util/delay.h>
```

```
int main()
{
   unsigned int i = 0;
   DDRF = 0xF0;   //LED 네 개를 output으로 설정
   DDRG = 0x00;   //DDRG[0] = 0, 즉 PinG 0를 input으로 설정
   PORTF = 0x10;  //PortF[5] = 1, 즉 첫번째 LED를 on

   while(1)
   {   if(i>3) {i = 0 ; PORTF = 0x10;} //한바퀴 돌면 초기화

      while(!(~PING & 0x01));    //버튼이 on할때까지 대기
      _delay_ms(10); //10ms=확인 간격
      PORTF = PORTF<<1; //PortF를 shift, 즉 다음 pin의 LED를 on

      while(~PING & 0x01);  //버튼을 off할때까지 대기
      _delay_ms(10);

      i=i+1;
   }

   DDRB = 0x80;
   DDRF = 0xF0;
   PORTF = 0x10;
   DDRG = 0x00;

}
```

코드 해석

```
DDRG = 0x00;
```

- switch의 output은 Pin D의 input이 된다.
- Sw5만 사용하므로 Pin D[0]를 0으로 설정(나머지는 X)

```
DDRF = 0xF0;
```

- LED에 해당하는 Port F를 output으로 설정.

```
PORTF = 0x10;
```

- 디폴트 값으로 첫번째 LED를 on

```
      while(!(~PING & 0x01));
      _delay_ms(10);
```

```
        PORTF = PORTF<<1;
```

- SW5가 눌리면 10ms 후 Port F를 1만큼 shift

```
        while(~PING & 0x01);  //버튼을 off할때까지 대기
        _delay_ms(10);
        i=i+1;
```

- 눌렀던 Sw5를 때면 10ms 후 i값 증가

Trouble Shooting

2. Interrupt를 활용해 타이머 제작

- Interrupt
    - 내부 혹은 외부의 요구에 의해서 현재 실행 중인 프로그램을 잠시 중단하고 보다 시급한 작업을 먼저 수행한 후 다시 실행중이던 프로그램을 복귀하는 것.
        - 내부 인터럽트 : AVR의 내부적인 요인에 의해 발생하는 인터럽트
            - 타이머/카운터, ADC, USART
        - 외부 인터럽트 : 외부의 전기적인 자극에 의해 발생하는 인터럽트
    - ATmega128 board에선 INT4, INT5가 interrupt로 작동 가능하다.
    - Interrupt Service Routine(ISR, 인터럽트 처리 루틴)
        - 인터럽트 발생시 수행되는 작업
    - Interrupt Vector
        - 각 인터럽트에 부여되는 고유한 번호를 의미함.
        - Interrupt Vector Table : 각각 인터럽트에 대한 ISR로 점프하는 어셈블리 명령어(JMP XXXX)가 저장되어있음.
        - 동시에 인터럽트 발생시 벡터 번호가 작은 인터럽트의 우선순위가 더 높다
    - SREG.I : Status Register의 I라는 MSB
        - set : 인터럽트 허용
        - reset : 인터럽트 금지

```
sei() //인터럽트 허용. SREG.I를 set시키는 역할
cli() //인터럽트 금지. SREG.I를 reset 시키는 역할
```

    - EIMSK[0:7](External Interrupt Mask Register)
        - 어떤 입력을 Interupt로 사용할지 결정.

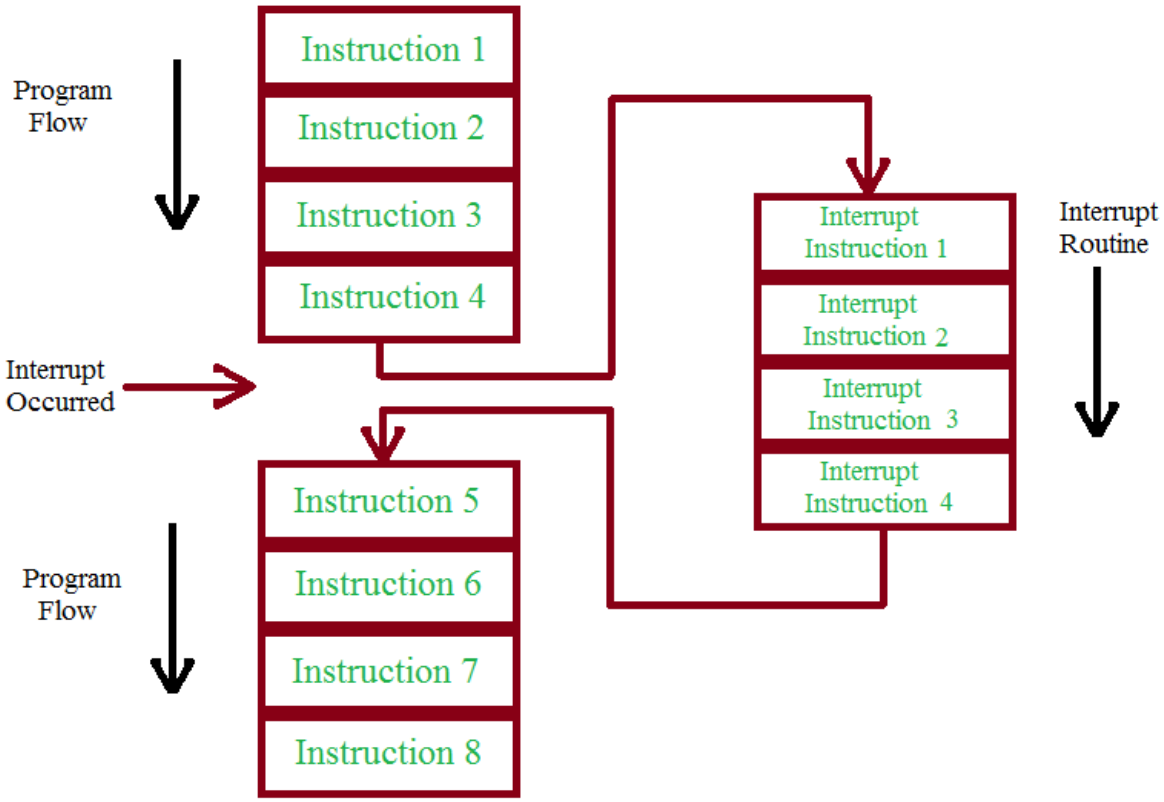
- 1 : 사용
- 0 : 비사용

- EICRx[0:7](External Interrupt Control Register)
  - 어떤 입력에 반응할지 결정. 스위치가 눌리는 순간 interrupt를 작동할지, 아니면 스위치가 눌렸다 다시 올라가는 순간 interrupt를 작동할지 결정해줌.
  - EICRA : INT 0~3까지에 대한 감지 방법 제어. 각 interrupt마다 2bit씩 할당된다.
  - EICRB : INT 4~7까지에 대한 감지 방법 제어. 각 interrupt마다 2bit씩 할당된다.

ISCn1	ISCn0	Description
0	0	input이 GND상태(Low)로 유지될 경우 발생
0	1	위치가 변경할때마다 발생
1	0	Falling edge(High>>Low)
1	1	Rising edge(Low>>High)

- EIRF[0:7](External Interrupt Flag Register)
  - 어떤 interrupt가 트리거가 되었는가를 표시함.
  - 해당 interrupt service routine으로 jump하면 0으로 설정.

- Interrupt 발생 순서



  - Interrupt 발생
  - Interrupt에 할당된 Interrupt Vector에 따라 Interrupt Vector Table 확인
  - Interrupt Vector Table에 저장된 해당 ISR로 점프
  - ISR 실행

        - 복귀

- Polling
    - 특정 주기마다 작업을 중단하며 외부의 요구 사항을 확인하는 것.
    - 내부에서 외부의 요구를 확인한다는 점에서 interrupt와 다름
    - 인터럽트보다 약간 빠름!
    - 그렇다고 인터럽트가 많이 빠른것도 아님. 잉여 시간이 많이 아깝다.

example01

```
#include<avr/io.h>
#include<avr/interrupt.h>

void init_port(void) //초기화
{
   DDRF = 0xF0;
   PORTF = 0x00; //input
   DDRE = 0x00;
   PORTE = 0xFF;
}

void init_interrupt(void)
{
   EIMSK = 0x00;
   EICRA = 0x00;
   EICRB = 0x08;
   EIMSK = 0x20;
   EIFR = 0xFF;   //원래 초반에 초기화 시켜주는 의미
}

int main(void)
{
   init_port();
   init_interrupt();
   sei();
   while(1)
   {
   }
   return 0;
}

ISR(INT5_vect)
{
   PORTF = ~PORTF;
}
```

- INT4 = 1이 되면 interrupt가 발생해 Port F의 값들이 반전된다.

코드 해석

```
void init_port(void) //초기화
{
   DDRF = 0xF0;
   PORTF = 0x00; //input
   DDRE = 0x00;
   PORTE = 0xFF;
}
```

- DDRF : LED부분을 output을 사용
- Port F : 처음엔 전부 off 상태
- DDRE : INT5의 output은 Pin E[5]의 input이 된다. 따라서 DDRE[5] = 0으로 설정

```
- Port E : Port E[5] = 1로 설정(Pull up switch)
void init_interrupt(void)
{
   EIMSK = 0x00;
   EICRA = 0x00;
   EICRB = 0x08;
   EIMSK = 0x20;
   EIFR = 0xFF;
}
```

- EIMSK = 0x00 : 모든 인터럽트 사용 안한다고 초기화
- EICRA = 0x00 : INT 0~3을 00으로, 즉 input이 GND로 유지될때 interrupt 발생하도록 설정
- EICRB = 0x08 : 세번째 비트를 1으로 설정. ISC5 = [0, 1]이 된다. 따라서 INT 5는 모든 위치 변경에 Interrupt 발생
- EIMSK = 0x20 : 여섯번째 비트를 1로 설정. 따라서 INT 5를 활성화한다.
- EIFR = 0xFF : 초기 모든 인터럽트를 활성화(?)

```
ISR(INT5_vect)
{
   PORTF = ~PORTF;
}
```

INT5_vext : PORTF = ~PORTF : LED 상태를 반전

example 02

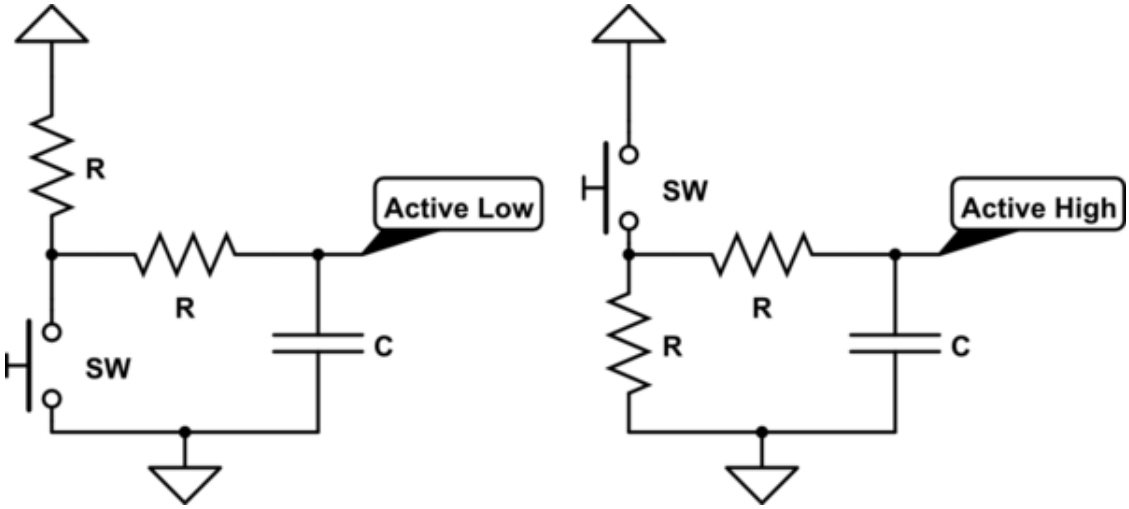
- 해석해보자!

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
```

```
void init_port(void) //초기화
{
        DDRF = 0xF0;
        PORTF = 0x00;
        DDRE = 0x00;
        PORTE = 0xFF;
}

void init_interrupt(void)  //인터럽트 초기화
{
        EIMSK = 0x00; // When changing ISCn1/ISCn0 bits, the interrupt must be
disabled by clearing its Interrupt Enable bit
                                        // in the EIMSK Register.
                                        // Otherwise an interrupt can occur when
the bits are changed.
        EICRA = 0x00;
        EICRB = 0x08;
        EIMSK = 0x20;
        EIFR = 0xFF;
}

int main(void)
{
        init_port();
        init_interrupt();
        sei();

unsigned int i = 0;
   DDRF = 0xF0;
   DDRG = 0x00;
   PORTF = 0x10;

   while(1)
   {   if(i>3) {i = 0 ; PORTF = 0x10;};
      while(!(~PING & 0x01)) ;
      _delay_ms(10);


      PORTF = PORTF<<1;

      while(~PING & 0x01) ;
      _delay_ms(10);
      i=i+1;
   }
        return 0;

}
        ISR(INT5_vect)

{
        PORTF = ~PORTF;
}
```

토의 사항

- exapmle 01
    - EIFR과 Port E는 완전히 다른 비트에 해당되는가? Port E의 output이 INT의 input으로 들어가고, 이를 통해 EIFR의 원소가 결정된다.
    - EIFR와 Port E를 1로 설정해두면 초기 인터럽트가 미리 한번 실행되지 않을까?
        - Switch에는 Pull up과 Pull down 두가지 방식이 있다.



        - Pull up switch : 버튼 눌리면 High에서 Low로 변함(default : High)
        - Pull down switch : 버튼 눌리면 Low에서 Hgit로 변함(default : Low)

Trouble Shooting

```
ISR(INT5_vect)
{
   PORTF = ~PORTF;
}
```

[AVR] 코드와는 달리 INT4를 누를때 interrupt 발생한다.

- Schematic과 비교해보면, 저항 R27과 INT5가 연결되어 있다. 즉, Board의 INT4가 Schematic의 INT5와 같다.

Day04

학습 목표

1. 질문

질문

조교님 이메일 - d7000@konkuk.ac.kr

- 초기에 flag register을 0xFF로 초기화화는 이유

> flag register을 1로 설정해주면 해당 flag는 0으로 초기화된다. 임의로 초기화는 불가능하며 순간적으로 인터럽트가 X상태인걸 방지하기 위해 해주는 작업이다.

- polling과 sampling

> edge를 감지를 한다 -> sampling 을 감지. -> sampling을 어떻게 해주느냐에 대한 차이! sleep모드 사용 시 일정 clock를 정지.(전원 아끼기용) -> I/O 클럭 일부가 죽음-> 어떤 애들은 제대로 작동 불가능

- sampling vs polling?

> sampling : 언제 신호의 변경이 감지됐다고 하드웨어에서 감지 polling : 언제 신호가 변경됐다고 소프트웨어에서 감지 interrupt : sampling 기반 try catch : 에러 처리에 대한 것, 인터럽트가 아니다. c에서 event listener : OS쪽에 등록된 세팅. 인터럽트와는 다르다.

- interrupt 사이의 우선도?

> INT 0~7까지 순서대로 실행 reset : 최우선적인 하드웨어적인 인터럽트. INT 0번보다 먼저 실행된다.

- interrupt가 실행되는 방식?

> 하드웨어에 대한 인터럽트는 그냥 PC가 넘어가는 식으로 실행된다. 이때, jal에서 링크에 대한 명령어는 ISR쪽에서 수행한다.

- delay 함수에 변수를 넣으면 안되는 이유?

> delay 함수 : no operation에 의해 구현이 됨 -> 컴파일러가 변수로 넣으면 안된다고 경고를 해주는 것-> 컴파일러에 따라 다르다.

- switch의 민감도를 조정하는 방법

> chattering이라는 방법이 있음. 기간을 길게 주면 동작에 대한 고려를 해야됨. switch의 물리적인 특성에 따라 시간을 다르게 보정해야된다.

- 공학은 어떤 학문인가

> 디지털 시스템을 구현하는게 핵심이다. -> 자신이 원하는 알고리즘을 하드웨어로 구현할 줄 알아야됨. -> 자신이 원하는 chip을 만들어서 실제로 구현이 되나 구현 -> 실제로 그게 구현이 되면 졸업 가능.(or 논문) -> 디지털 칩 시스템과 인베디드 시스템이 핵심 유형임 -> 8대 공정

- 베릴로그란 무엇인가?

> 하드웨어 자체를 설계하는 언어 -> 메인 메모리, capaciter 등등

- HDL?

> 하드웨어를 설계 가능 -> 수동소자 자체를 설계를 하는건 힘들다.-> 하드웨어를 말로써 풀어서 말할 수 있는게 HDL언어라 보면 됨. 베릴로그 HDL은 실제와 거의 비슷함. -> 어느정도 배열 시스템지원

- flip flap vs latch?

> clock의 기반이 되는 flipflap이 훨씬 중요

- FPGA?

> 병행 기능을 본인이 직접 짜야함.

- idec.or.kr : 베릴로그(HDL) 무료 강좌 있음. 원하면 회원가입해서 들어볼 것
- HDL vs VHDL?

> 하드웨어를 작성할때 사용하는 HDL 중 VHDL이 있음

- FPGA(feild programalbe build array) - 리눅스 올리는게 힘들다. -> 정확히 말하면 게이트들을 잘 연결해 주는게 FPGA
- DE1보드의 경우 arm 코어가 옆에 하나 있음. 거기에 보통 FPGA를 올림.
- 디지털 설계 방향 = 실험+응논(김창범 교수님) + 마프(박성정 교수님) + 컴구(박성정 2학년 2학기)(+물성+임베디드 자료구조(?)+알응(?) + C++(객체 지향) + 제어 공학(굳이 추천 안함.))
- 아날로그 - 전자기학 2
- 디지털 설계와 반도체는? 반도체도 상당히 갈래가 많음. 반도체 물성? 반도체 설계? 아날로그 반도체? 등등
- 사실상 아날로그 물성 반도체를 포괄해서 반도체라고 한다. 진심으로 그쪽으로 나갈가면 반도체 관련 과목을 들어보자.
- 인베디드는 완전히 반조체 과목은 아님.
- FPGA - 병렬 처리에 특화가 되있음. 똑같은걸 수십개 동시에 돌리기 가능.
- FPGA에 멀티 쓰레드는 사실상 효율이 낮은 작업. 하드웨어 논리 게이트의 속도가 너무 느림.
- FPGA는 보통 Cloud를 못따라간다. 해당 작업을 연계하기 위해선 상당히 비싼 FPGA보드를 사야한다.
- AVR - 16MH -> free scaler 예제를 참고할것.
- 전자회로 2 - 라플라스 변환 공부해둘것.
- time interrupt+ 7segment -> 한바퀴 돌기-> 하나씩 쌓여서 네모 모양 만든 후 다시 네모 모양 하나씩 사라지게 하기
- timer - interrupt로 clock, 즉 시간이 지나가는것을 interrupt로 수행
- 라인 트레이서 - 하드웨어와 소프트웨어 모두 잡아야됨! -> 센서를 직접 제작

Day08

학습 목표

1. 타이머 구현

1. 타이머 구현

1. 레지스터

- 제어 레지스터(t/c control register, TCCRn, n=0,2) : 동작 모드 및 분주비 설정

Timer/Counter Control Register – TCCR0

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	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	

  - FOC0 : 해당 비트에 1을 쓰면 파형 생성 유닛에 즉각적인 비교일치를 전달.
  - WGMn[0:1] : 파형 생성 모드를 결정하는 비트. Normal, PWM, Phase Correct PWM, CTC, Fast PWM의 모드 조정 가능.(2번에 각 기능 정리)

Mode	WGM01[1] (CTC0)	WGM00[1] (PWM0)	Timer/Counter Mode of Operation	TOP	Update of OCR0 at	TOV0 Flag Set on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	Immediate	MAX
3	1	1	Fast PWM	0xFF	BOTTOM	MAX

  - COMn[0:1] : 비교일치 출력 모드로서 OCn의 동작을 제어한다. 이 비트 중 하나 이상의 비트가 set되면 공유하는 범용 I/O핀의 기능을 대신한다. 이때, OCn 핀을 사용하기 위해선 범용 I/O핀의 방향을 출력으로 설정해야된다.

Table 53. Compare Output Mode, non-PWM Mode

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Toggle OC0 on compare match
1	0	Clear OC0 on compare match
1	1	Set OC0 on compare match

Table 54. Compare Output Mode, Fast PWM Mode[1]

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match, set OC0 at TOP
1	1	Set OC0 on compare match, clear OC0 at TOP

Table 55. Compare Output Mode, Phase Correct PWM Mode[1]

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match when up-counting. Set OC0 on compare match when downcounting.
1	1	Set OC0 on compare match when up-counting. Clear OC0 on compare match when downcounting.

  - CSn[0:2] : 클록 선정 비트

Table 46. Clock Select Bit Description

CS22	CS21	CS20	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	clk_{T2S}/(No prescaling)
0	1	0	clk_{T2S}/8 (From prescaler)
0	1	1	clk_{T2S}/32 (From prescaler)
1	0	0	clk_{T2S}/64 (From prescaler)
1	0	1	clk_{T2S}/128 (From prescaler)
1	1	0	clk_{T2S}/256 (From prescaler)
1	1	1	clk_{T2S}/1024 (From prescaler)

- 타이머/카운터 레지스터(t/c register, TCNTn, n=0,2) : 실제 시스템 내부 클럭수에 따라 카운팅하는 레지스터. 비트를 조정해 0~255사이의 숫자로 시작점을 바꿀 수 있다.

Timer/Counter Register – TCNT0

Bit	7	6	5	4	3	2	1	0	
	TCNT0[7:0]								TCNT0
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	

The Timer/Counter Register gives direct access, both for read and write operations, to the Timer/Counter unit 8-bit counter. Writing to the TCNT0 Register blocks (removes) the compare match on the following timer clock. Modifying the counter (TCNT0) while the counter is running, introduces a risk of missing a compare match between TCNT0 and the OCR0 Register.

- 만약 TCNT0 = 0x06이라면 6부터 카운팅 시작.

- 출력 비교 레지스터(output compare register, OCRn, n=0,2) : TCNT와 비교해 OCn단자에 출력을 발생하기 위한 8비트 값을 저장하는 레지스터.
- 타이머/카운터 인터럽트 플래그 레지스터(t/c interrupt flag register, TIFR)

Timer/Counter Interrupt Flag Register – TIFR

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
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 1 – OCF0: Output Compare Flag 0**

The OCF0 bit is set (one) when a compare match occurs between the Timer/Counter0 and the data in OCR0 – Output Compare Register0. OCF0 is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, OCF0 is cleared by writing a logic one to the flag. When the I-bit in SREG, OCIE0 (Timer/Counter0 Compare Match Interrupt Enable), and OCF0 are set (one), the Timer/Counter0 Compare Match Interrupt is executed.

- **Bit 0 – TOV0: Timer/Counter0 Overflow Flag**

The bit TOV0 is set (one) when an overflow occurs in Timer/Counter0. TOV0 is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, TOV0 is cleared by writing a logic one to the flag. When the SREG I-bit, TOIE0 (Timer/Counter0 Overflow Interrupt Enable), and TOV0 are set (one), the Timer/Counter0 Overflow Interrupt is executed. In PWM mode, this bit is set when Timer/Counter0 changes counting direction at $00.

- OCFn : 출력 비교 레지스터. 타이머 TCNTn과 비교 레지스터 OCRn이 같을때 OCRn이 set된다. 해당 ISR이 실행되면 자동으로 clear된다.
- TOVn : 타이머값 TCNTn에 오버플로우가 발생하면 set 된다. 해당 ISR이 실행되면 자동으로 clear된다.

- 비동기 상태 레지스터(asynchronous status register, ASSRn, n=0) : 타이머/카운터 0의 동작을 설정하는 레지스터

비트	7	6	5	4	3	2	1	0	
0x30(0x50)	-	-	-	-	AS0	TCN0UB	OCR0UB	TCR0UB	ASSR
읽기/쓰기	읽기/쓰기	읽기/쓰기	읽기/쓰기	읽기/쓰기	읽기/쓰기	읽기/쓰기	읽기/쓰기	읽기/쓰기	
초기값	0	0	0	0	0	0	0	0	

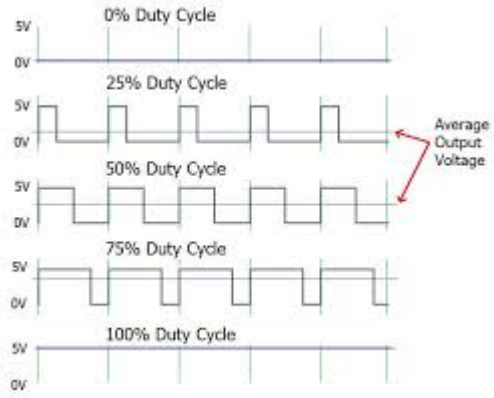
- AS0 : set일때 내부 클럭 clk/O가 선택되며 타이머로 작동, clear일때 TOSC에 입력되는 클럭이 선택되며 카운터로 작동
- TCN0UB : 카운터로 동작할때 TCNT0에 새로운 값이 쓰여지면 set되며 쓰기를 완료했을때 자동으로 clear
- OCR0UB : 카운터로 동작할때 OCR0에 새로운 값이 쓰여지면 set되며 쓰기를 완료했을때 자동으로 clear
- TCR0UB : 카운터로 동작할때 TCCR0에 새로운 값이 쓰여지면 set되며 쓰기를 완료했을때 자동으로 clear

2. 파형 생성 모드

1. 용어 정리

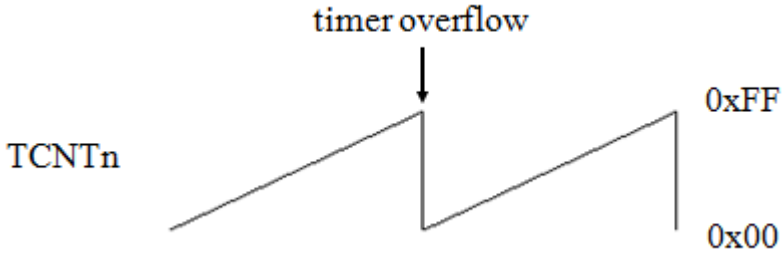
1. BOTTOM : 0, 혹은 카운터가 0이 될때
2. MAX : 0xFF,혹은 카운터가 0xFF가 될때

3. TOP : MAX 혹은 OCRn에 저장된 값
4. PWM(Pulse width Modulation) : 펄스 폭 변조. HIGH 상태의 길이 W를 조절하는 것을 말한다. PWM이 클수록 한 주기당 평균 전압은 높아지고, 반대로 작을수록 한 주기당 평균 전압은 낮아진다.



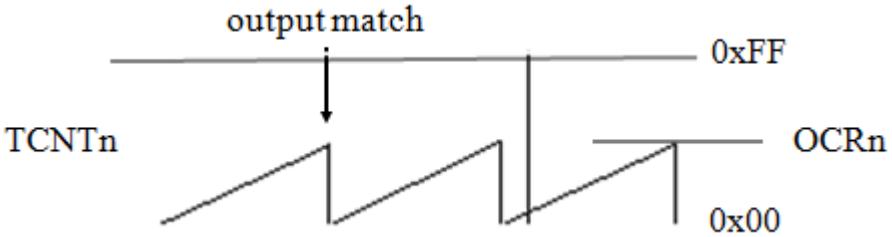
5. inverting compare output mode : TCNT0 값과 OCR0와 일치때 OC0가 set되고 TCNT0가 BOTTOM일때 clear
6. non-inverting compare output mode : TCNT0가 OCR0와 일치때 OC0가 clear되고 TCNT0가 BOTTOM일때 set.

2. Normal interrupt mode



- WGMn[0:1]=0
- TCNT0에서 overflow가 발생할때 인터럽트가 발생하는 모드
- COM00, COM01 값을 통해 inverting compare output mode와 non-inverting compare output mode를 설정할 수 있으며, 해당 포트 핀의 데이터 방향이 출력일때 OC값이 실제로 출력된다.

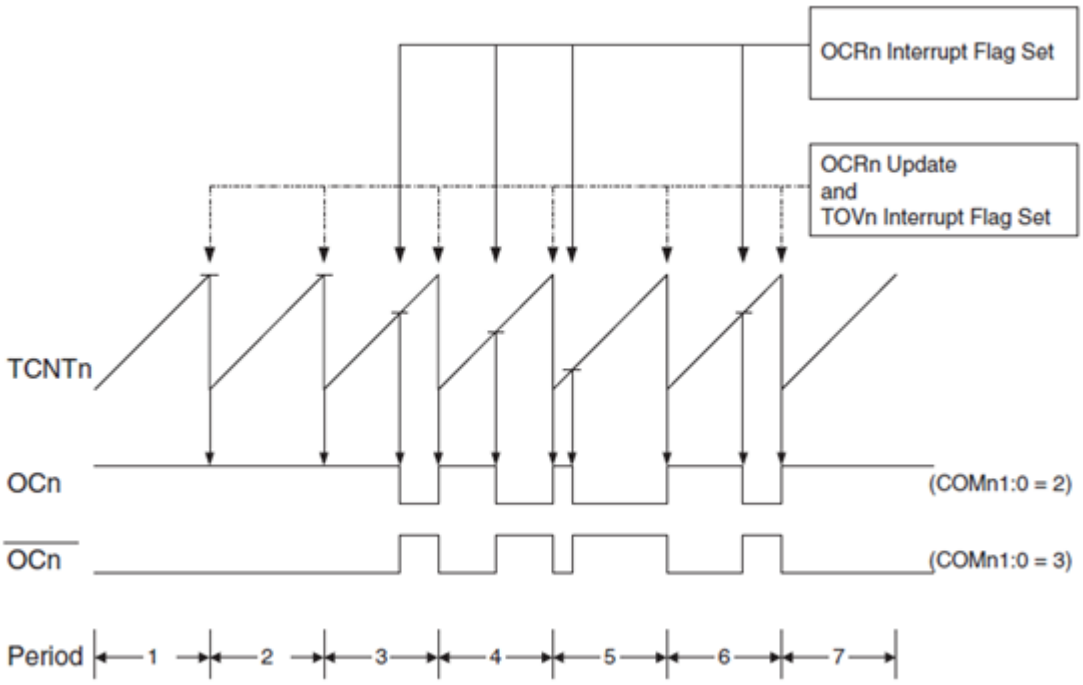
3. Compare Match interrupt(CTC)



- WGMn[0:1]=2
- TCNT0값과 ORC0의 값이 일치할때 인터럽트가 발생하는 모드
- COM00, COM01 값을 통해 inverting compare output mode와 non-inverting compare output mode를 설정할 수 있으며, 해당 포트 핀의 데이터 방향이 출력일때 OC값이 실제로

출력된다.

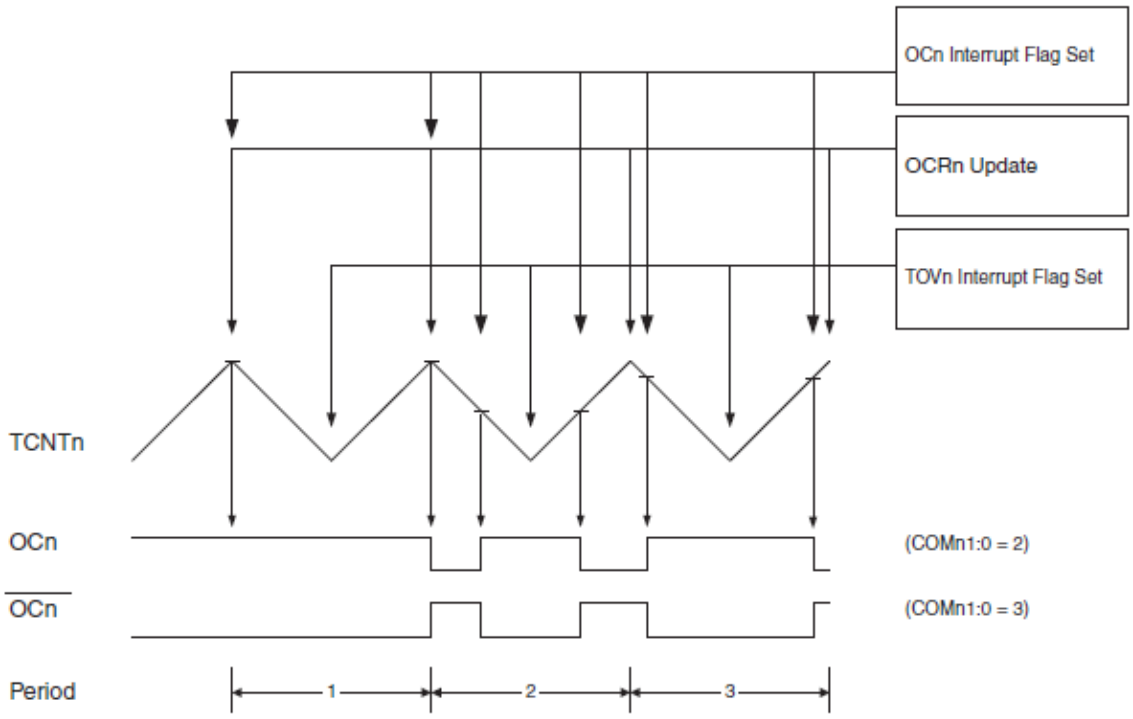
4. Fast PWM mode



- WGMm[0:1]=3
- Fast PWM mode에선 TCNT0는 MAX와 일치할때까지 증가하며 TCNT에서 overflow가 발생할때 OC0는 set된다.
- TCNT0의 값과 OCRn의 값이 일치할때, OC0는 clear된다.
- 일반적으로 오버플로우 핸들러 루틴을 통해 OCR의 값을 설정한다.
- COM00, COM01 값을 통해 inverting compare output mode와 non-inverting compare output mode를 설정할 수 있으며, 해당 포트 핀의 데이터 방향이 출력일때 OC값이 실제로 출력된다.

5. Phase Correct PWM mode

Figure 40. Phase Correct PWM Mode, Timing Diagram



- WGMn[0:1]=1
- 좌우 대칭 모양의 파형을 생성한다.
- TCNT0가 TOP일때, flag 레지스터를 set한다. 즉,CPU는 이때 overflow가 발생한것으로 인식하며, 인터럽트를 활성화 시키면 이 시점마다 인터럽트를 발생시킬 수 있다.
- TCNT0가 BOTTOM일때 TOV Flag 레지스터를 초기화한다.
- TCNT0와 OCR0가 일치할때 OC0의 값이 반전된다.
- COM00, COM01 값을 통해 inverting compare output mode와 non-inverting compare output mode를 설정할 수 있으며, 해당 포트 핀의 데이터 방향이 출력일때 OC값이 실제로 출력된다.

```
PORTC |= 0x01;    //Set bit 0 only
PORTC &= ~0x01;   //Clear bit 0 only
PORTC ^= 0x01;    //Toggle bit 0 only
PORTC & 0x01;     //Test bit 0 only
PORTC = 0x80;     //Set bit 7 only
                  //BV(0) = 1 << x
PORTC |=  _BV(0); //Set bit 0 only
PORTC &= ~(_BV(1));  //Clear bit 1 only
PORTC ^= _BV(7); //Toggle bit 7 only
PORTC |=(_BV(0)|_BV(2)|_BV(7)); //Set bits 0,2,7
PORTC &= ~(_BV(1)|_BV(2)|_VB(6)); //Clear bit 1,2,6
PORTC ^=(_VB(5)|_BV(3)); //Toggle bits 3,5
```

example

```
#include <avr/io.h>
#include <avr/interrupt.h>

volatile int temp =0;
int t = 15625;
ISR(TIMER0_OVF_vect)
        {
        temp++;
        if(temp==t)
                {
                temp=0;
                PORTF = (PORTF==(0x80))? 0x10:PORTF<<1;
                //기호 ? 앞이 참이면 0x10, 거짓이면 PORTF<<1
                }
        }

void timer_setting()
        {
        TCCR0 =  _BV(1)|_BV(1)|_BV(1);   //원하는 분주기를 넣을것 굳이 이런 형식 쓰는
이유가?
        TCNT0 = 0x00;  //
        TIMSK = 0x01;
        TIFR = 0xFF;
        }

int main(void)
        {
        timer_setting();
        DDRF = 0xFF;
        PORTF = 0x10;
        sei();
        while(1)
                {
                }

        return 0;

        }
```

TCCR0 = 0x07 :

참고 자료 : https://webnautes.tistory.com/987 https://wjs890204.tistory.com/754