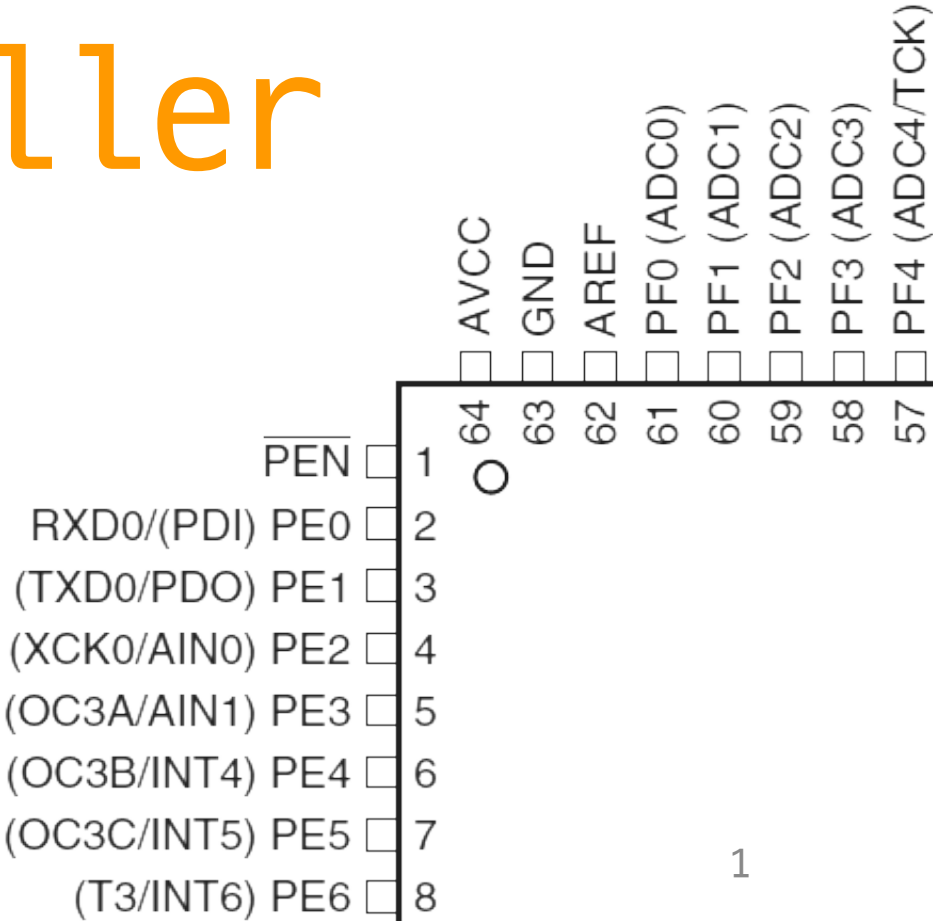
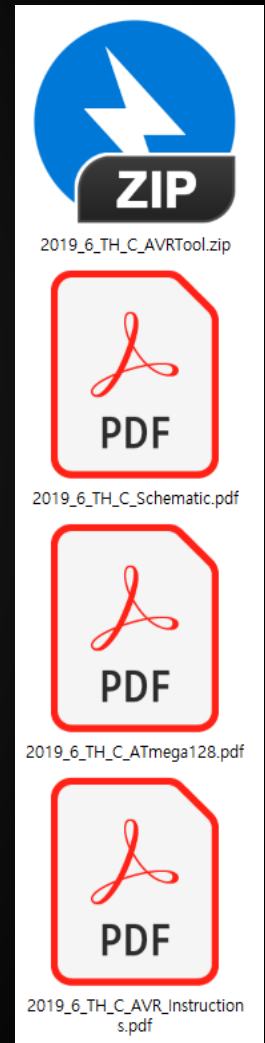


Introduce Microcontroller



•시작하기 전에

- 실험 홈페이지에서 필요한 자료 다운받으세요.
- 2019_6_TH_C_AVRTool.zip
 - AVR 개발 환경 세팅에 필요한 파일들
- 2019_6_TH_C_Schematic.pdf
 - AVR 개발보드의 Schematic
 - Schematic을 확인하지 않고 개발하는 것은 불가능합니다.
 - 꼭 Schematic를 확인하면서 개발하세요.
- 2019_6_TH_C_ATmega128.pdf
 - ATmega128 MCU 사용설명서
 - 강의자료에서 사용설명서를 참고하라고 하면, 이 파일을 참고합니다.
- 2019_6_TH_C_AVR_Instructions.pdf
 - ATmega128 MCU 명령어 설명서



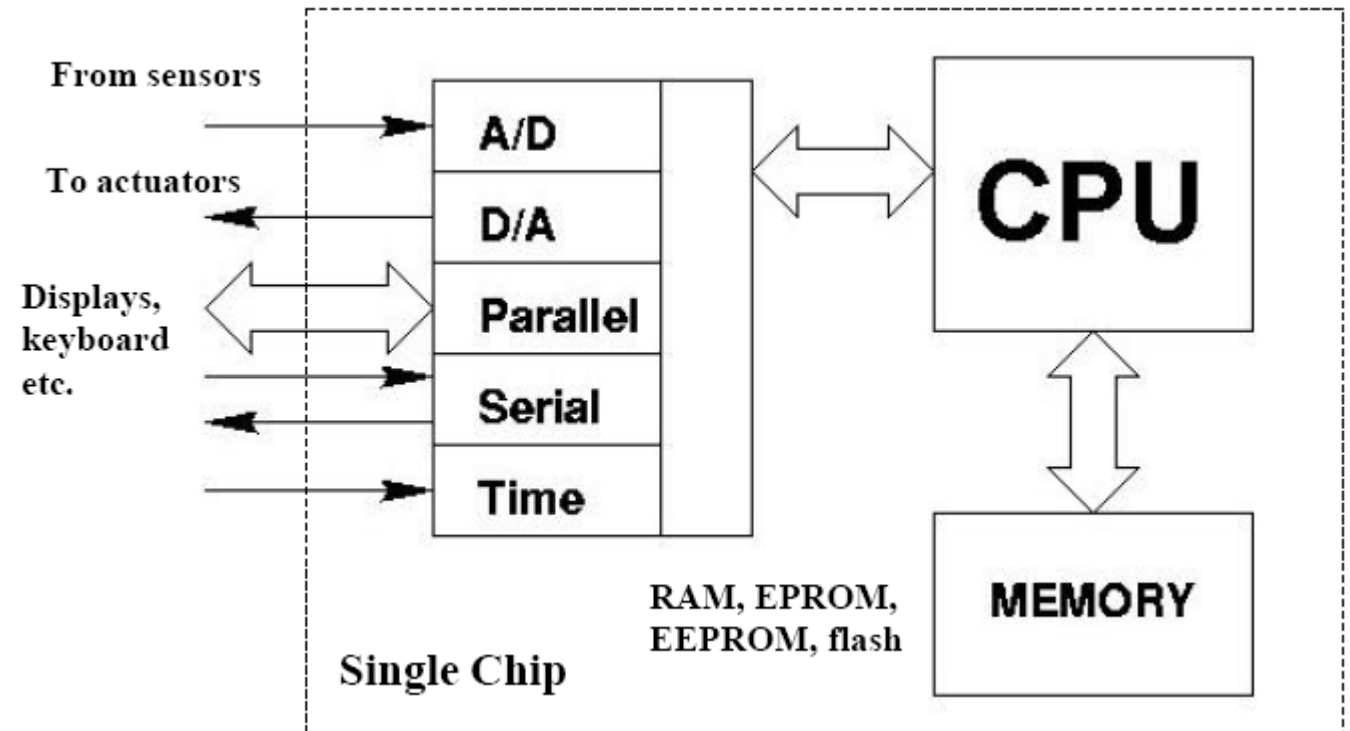
•What is microcontroller

- A microcontroller is a small, low-cost computer-on-a-chip which usually includes:
 - An 8 or 16 bit microprocessor (CPU).
 - A small amount of RAM.
 - Programmable ROM and/or flash memory.
 - Parallel and/or serial I/O.
 - Timers and signal generators.
 - Analog to Digital (A/D) and/or Digital to Analog (D/A) conversion.

•Basics of microcontroller

➤다양한 요소로 구성되어 있음

- 기본적인 연산 및 프로그램 구동을 위한 CPU
- 데이터 저장을 위한 메모리
- 데이터 입출력을 위한 직/병렬 통신 장치
- 타이머
- 그 외에 다양한 주변장치를 내장함



•Microcontrollers

- Intel 8051
- ARM
- Atmel AVR 8/32-bit architecture
- Freescale CF (32-bit), Freescale S08(8-bit) , 68HC11(8-bit)
- Hitachi H8, Hitachi SuperH
- MIPS (32-bit PIC32)
- NEC V850
- PIC (8-bit PIC16, PIC18, 16-bit dsPIC33 / PIC24)
- PowerPC ISE
- PSoC (Programmable System-on-Chip)
- Rabbit 2000
- TI MSP430 (16-bit)
- Toshiba TLCS-870
- Zilog eZ8, eZ80
 - 이외에도 다양한 회사에서 다양한 제품군을 출시중임.

•Atmel AVR Microcontroller

- The AVR is a Modified Harvard architecture 8-bit RISC single chip microcontroller which was developed by Atmel in 1996.
- The AVR was one of the first microcontroller families to use on-chip flash memory for program storage, as opposed to One-Time Programmable ROM, EPROM, or EEPROM used by other microcontrollers at the time.
- Families of AVR
 - tinyAVR – the ATtiny series
 - megaAVR – the ATmega series
 - XMEGA – the ATxmega series
 - Application specific AVR

•Atmel AVR Microcontroller

- High-performance, Low-power AVR® 8-bit Microcontroller
- Advanced RISC Architecture
 - 133 Powerful Instructions – Most Single Clock Cycle Execution
 - 32 x 8 General Purpose Working Registers + Peripheral Control Registers
 - Fully Static Operation
 - Up to 16 MIPS Throughput at 16 MHz
 - On-chip 2-cycle Multiplier

- Nonvolatile Program and Data Memories
 - 128K Bytes of In-System Reprogrammable Flash
 - Endurance: 10,000 Write/Erase Cycles
 - Optional Boot Codewith Independent Lock Bits
 - In-System Programming by On-chip Boot e Section Program
 - True Read-While-Write Operation
 - 4K Bytes EEPROM
 - Endurance: 100,000 Write/Erase Cycles
 - 4K Bytes Internal SRAM
 - Up to 64K Bytes Optional External Memory Space
 - Programming Lock for Software Security
 - SPI Interface for In-System Programming

- Special Microcontroller Features
 - Power-on Reset and Programmable Brown-out Detection
 - Internal Calibrated RC Oscillator
 - External and Internal Interrupt Sources
 - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
 - Software Selectable Clock Frequency
 - ATmega103 Compatibility Mode Selected by a Fuse
 - Global Pull-up Disable

- JTAG (IEEE std. 1149.1 Compliant) Interface
 - Boundary-scan Capabilities According to the JTAG Standard
 - Extensive On-chip Debug Support
 - Programming of Flash, EEPROM, Fuses and Lock Bits through the JTAG Interface

- Peripheral Features
 - Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
 - Two Expanded 16-bit Timer/Counters with Separate Prescaler, Compare Mode and Capture Mode
 - Real Time Counter with Separate Oscillator
 - Two 8-bit PWM Channels
 - 6 PWM Channels with Programmable Resolution from 2 to 16 Bits
 - Output Compare Modulator
 - 8-channel, 10-bit ADC
 - 8 Single-ended Channels
 - 7 Differential Channels
 - 2 Differential Channels with Programmable Gain at 1x, 10x, or 200x
 - Byte-oriented Two-wire Serial Interface
 - Dual Programmable Serial USARTs
 - Master/Slave SPI Serial Interface
 - Programmable Watchdog Timer with On-chip Oscillator
 - On-chip Analog Comparator

- I/O and Packages
 - 53 Programmable I/O Lines
 - 64-lead TQFP and 64-pad QFN/MLF

- Operating Voltages
 - 2.7 - 5.5V for ATmega128L
 - 4.5 - 5.5V for ATmega128
- Speed Grades
 - 0 - 8 MHz for ATmega128L
 - 0 - 16 MHz for ATmega128

•ATmega128 – Pinout

➤여러 개의 Pin이 모여 하나의 Port를 이룸

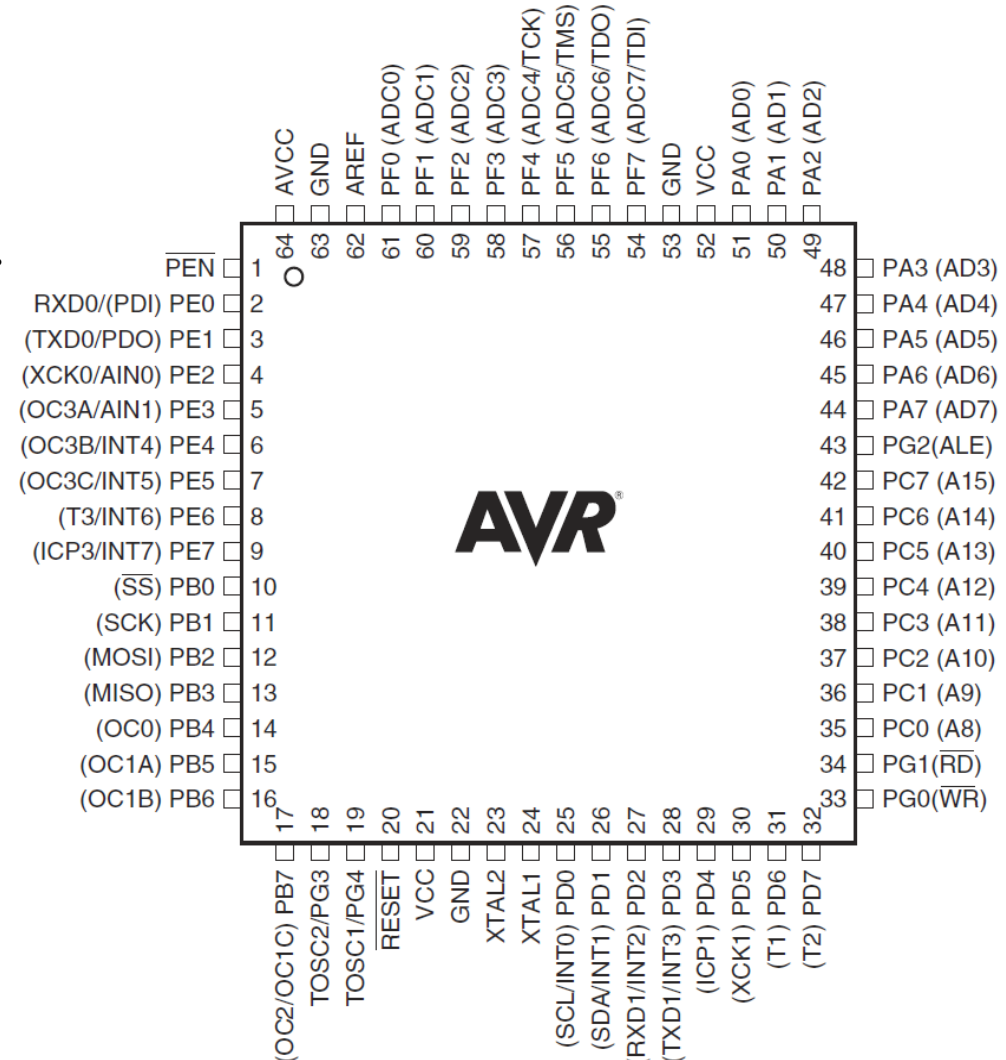
- 각 Port의 이름은 Port x라고 부름.
 - x는 Port 이름으로, A ~ G까지의 알파벳이다.
- ATmega128은 Port A ~ Port G까지, 총 7개의 Port가 있음
- Port G를 제외한 각 Port들은 8개의 Pin을 가지고 있음
 - Port G는 5개의 Pin을 가지고 있음

➤각 Pin의 이름은 Pxn으로 부름

- n은 Pin 번호로, 0 ~ 7의 정수
- 예를 들어 Port A의 0번째 Pin은 PA0라고 부름.

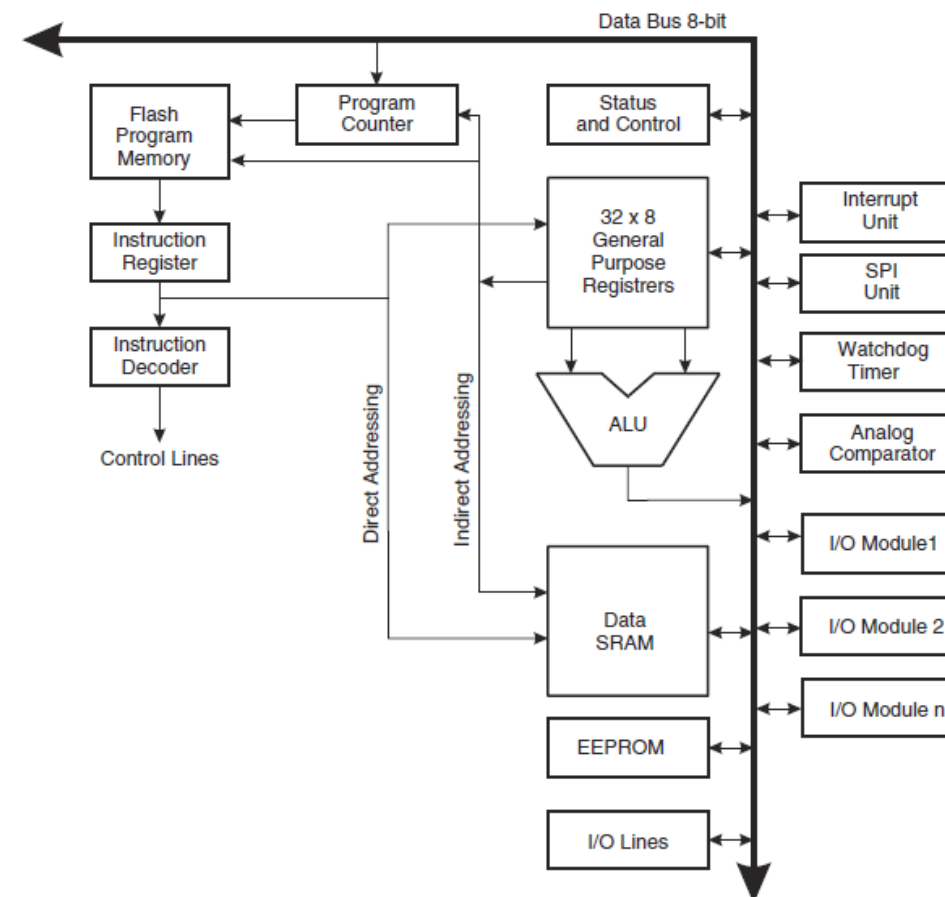
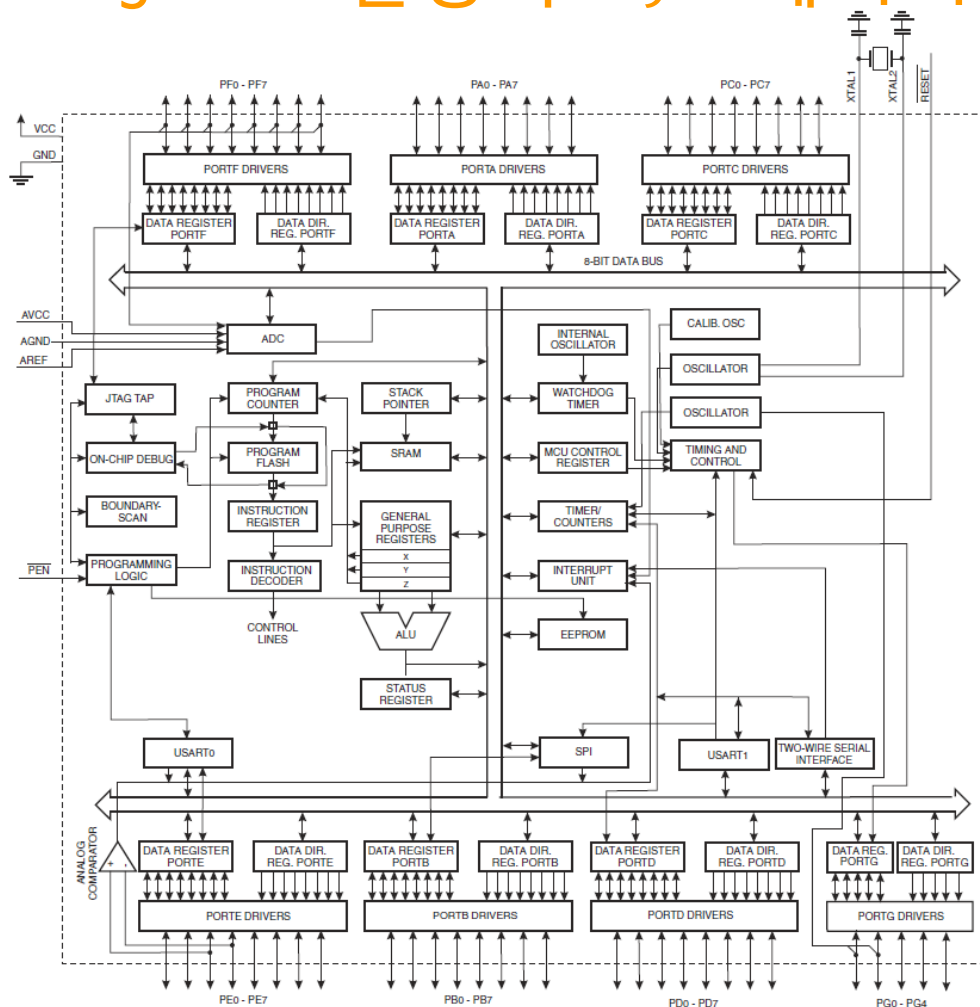
➤각 Pin은 한가지 이상의 기능을 가지고 있음

- 예를 들어 PE0는 단순 GPIO로 활용할 수 있지만, 설정에 따라 UART0의 RX로 사용할 수 있음.



•Atmega128 – Block Diagram

➤ATmega128 설명서 3, 9페이지 참고



•Atmega128 – Pin function (1)

➤ Port A(PA7...PA0)

- 8-bit bi-direction I/O port with internal pull-up registers
- Output buffers have symmetrical drive characteristics with both high sink and source capability
- Tri-stated when a reset condition becomes active

➤ Port B(PB7-PB0)

- 8-bit bi-direction I/O port with internal pull-up registers
- Output buffers have symmetrical drive characteristics with both high sink and source capability
- Tri-stated when a reset condition becomes active

➤ Port C(PC7-PC0)

- In Atmega103 compatibility mode, PortC is output only and C pins are not tri-stated when a reset condition becomes active

➤ Port D,E

- 8-bit bi-direction I/O port with internal pull-up registers
- Output buffers have symmetrical drive characteristics with both high sink and source capability
- Tri-stated when a reset condition becomes active

•Atmega128 – Pin function (2)

➤Port F(PF7–PF0)

- A/D converter input
- Can use as an 8 bit bi-direction I/O port, if A/D Converter is not used
- Serve the function of the JTAG interface.

➤Port G(PG4–PG0)

- 5-bit bi-direction I/O port with internal pull-up registers
- Output buffers have symmetrical drive characteristics with both high sink and source capability
- Tri-stated when a reset condition becomes active

•Atmega128 – Pin function (3)

➤ /PEN (Program Enable)

- Pin number : 1
- For SP1 Serial Programming mode and is internally pulled high
- Holding this pin low during a Power-on Reset

➤ AREF (analog reference pin)

- Pin number : 62
- Analog reference pin for the A/C Converter

➤ /AVCC

- Pin number : 64
- Supply voltage pin for PORT F and A/D converter

➤ RST (Reset)

- Pin number : 20
- Master reset

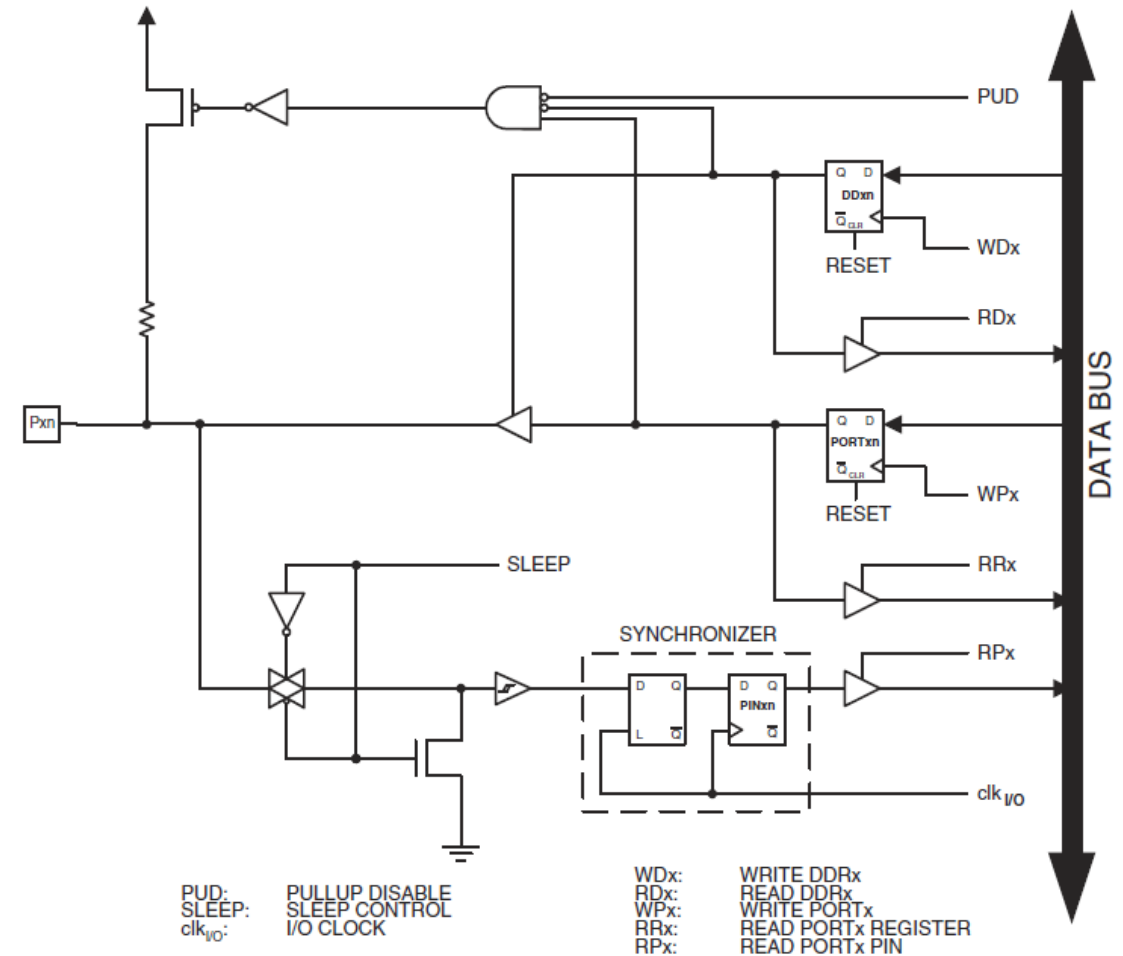
➤ XTAL1, XTAL2

- Pin number : 23,24
- Oscillator input

•AVR GPIO

➤ 각 Pin은 여러 개의 Tri-state buffer와 Flip flop에 의해 제어됨.

- ATmega128 설명서 66페이지 참고
- 각 FF에 저장된 값에 따라 Tri-state buffer를 통해 출력되는 값이 달라짐.
- 각 FF의 값은 몇몇 Register 값에 따라 결정된다.
 - DDRx, PORTx Register
- PINx Register는 Latch를 사용하여 데이터를 받음.
 - Latch는 metastability 현상 피하기 위해 사용
 - 외부에서 값을 입력 받는 경우, 내부 $Clk_{I/O}$ 에서 Edge가 발생하는 순간에 외부 입력 값이 바뀌는 경우에 내부 로직에서 문제가 생김.



Note: 1. WPx, WDx, RRx, RPx, and RDx are common to all pins within the same port. $clk_{I/O}$, SLEEP, and PUD are common to all ports.

•AVR GPIO

➤DDRx Register

- Data Direction Register
- 읽기, 쓰기가 가능
- 해당 Port의 Pin의 입출력 방향을 설정.
- 1로 설정되면 출력, 0으로 설정되면 입력으로 설정하는 것을 의미함.
 - `DDRF = 0xF0;` Port F의 7 ~ 4번 Pin은 출력, 3 ~ 0번 Pin은 입력으로 사용.

➤PINx Register

- Port Input Pins Register
- 읽기만 가능
- 해당 Port의 입출력 값이 저장되어 있음.
- DDRx 값과 상관 없이 작동
 - 즉, DDRx에 의해 출력으로 설정된 Pin이라도, PINx Register를 사용해 현재 Pin의 상태를 확인할 수 있음.
- 그 값을 읽어서 1이면 해당 Pin은 High, 0이면 Low임을 알 수 있음.
 - `k = PINF;` // Port F의 상태를 읽어 k에 저장, k가 `0b10000000`이면 PF7이 High, 나머지는 Low라는 의미임.
- 물리적인 입출력부를 의미하는 Pin과 register의 이름으로써 PINx를 혼동하지 말 것.
 - 이름만 같지, 실제 의미하는 바는 다름.

•AVR GPIO

➤PORTx Register

- Data Register
- 읽기, 쓰기가 가능
- 해당 Port의 Pin이 DDRx Register에 의해 출력 방향으로 설정된 경우, 해당 Pin의 출력 값을 의미함.
1이면 High 출력, 0이면 Low 출력
 - `PORTF = 0x01<<4; // PF4에 High 출력`
- 해당 Port의 Pin이 DDRx Register에 의해 입력 방향으로 설정된 경우, 해당 Pin에 내부 Pull-Up 저항 연결 여부를 결정함.
 - SFIOR Register의 PUD 비트가 High인 경우, 이 설정이 무시되고 모든 Pin의 Pull-Up저항이 연결되지 않음.
- 물리적인 Pin의 집합을 의미하는 Port x와 Register의 이름으로써 PORTx와 혼동하지 말 것.
 - 이름만 같지, 실제 의미하는 바는 다름.
 - 예) "DDRA를 0xFF로 설정하고, PORTA를 0xFF로 설정하면 Port A의 모든 Pin은 High 상태가 된다."라는 문장에서...
 - Port A : ATmega128의 입출력 Port 이름. 여러 개의 Pin으로 구성된 입출력 Port를 의미한다.
 - PORTA : ATmega128의 I/O Register의 이름. 8개의 Bit로 구성되었으며, Port A의 출력 값 또는 내부 Pull-Up 저항 연결 여부를 결정하는 Register이다.

•AVR GPIO

➤DDR_x, PORT_x 설정에 따른 Pin 동작

DD _{xn}	PORT _{xn}	PUD (in SFIOR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	P _{xn} will source current if ext. pulled low.
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)

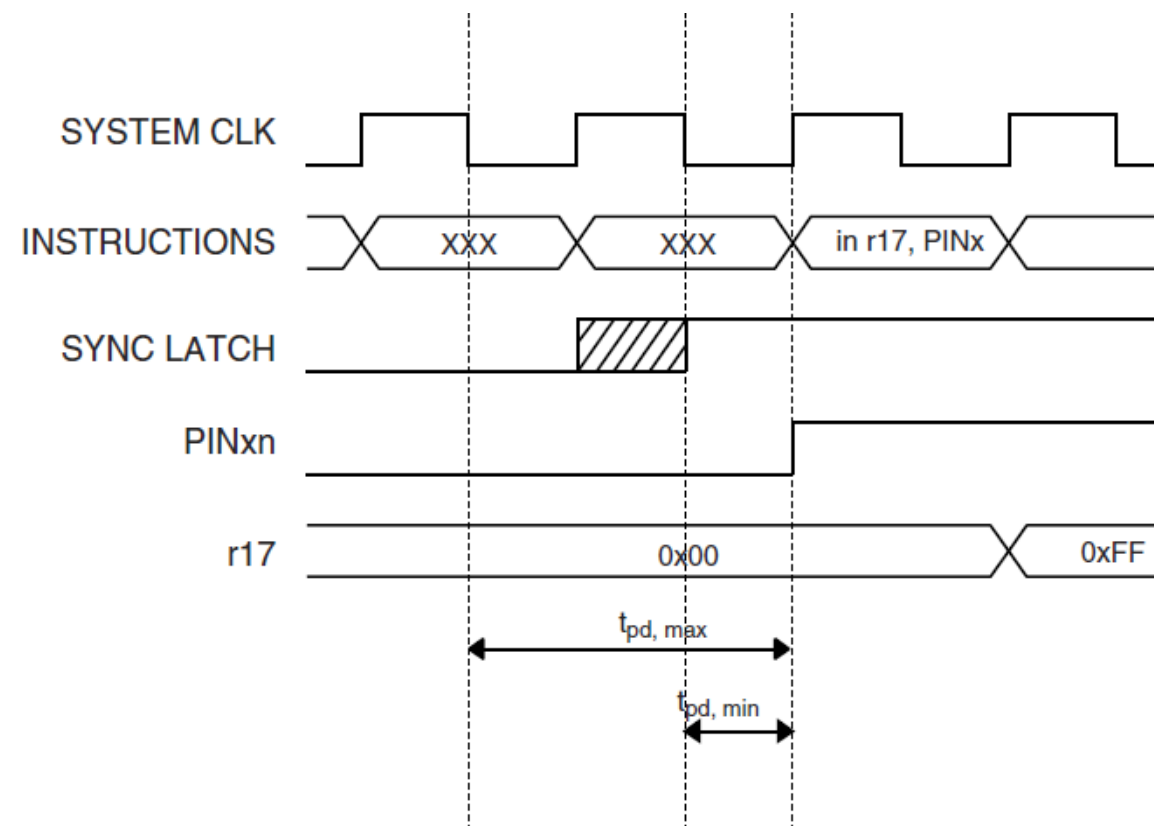
➤ATmega128 설명서 67페이지 참고

•AVR GPIO

➤PINx Register는 Pin의 값을 Latch를 통해 Synchronize하여 받는다.

➤Slide 13페이지, ATmega128 설명서 67 참고.

➤따라서 0.5 ~ 1.5 클럭 가량의 지연이 발생함.

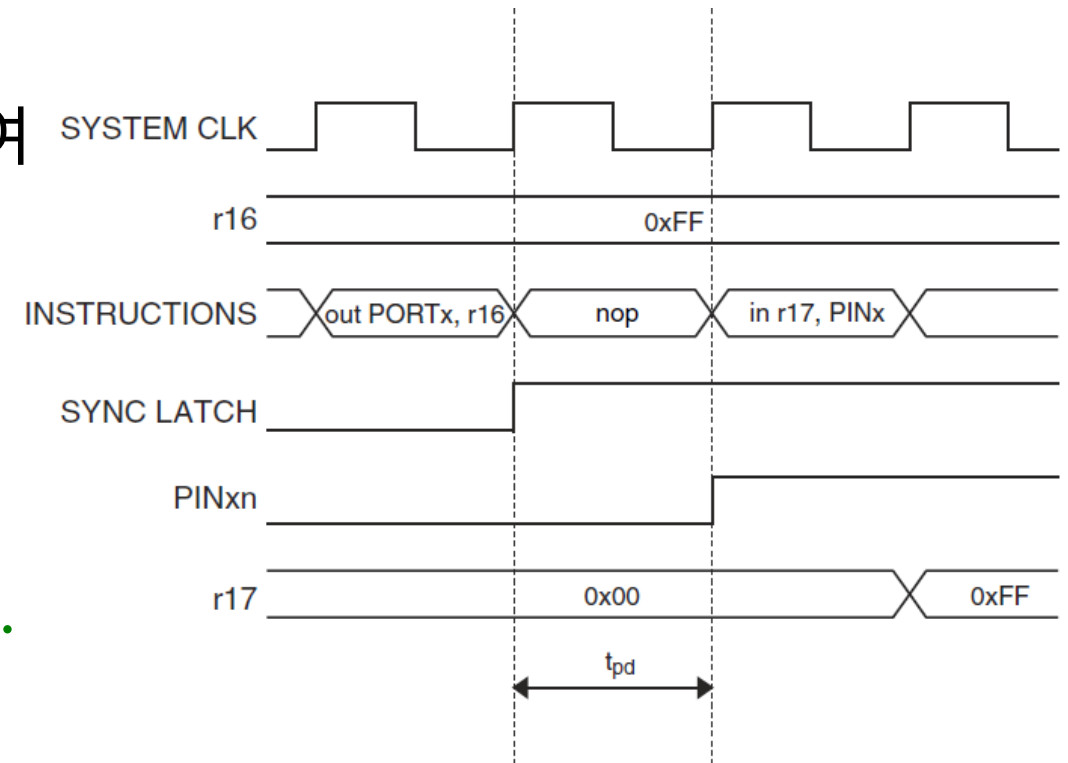


•AVR GPIO

➤PORTx Register 기록 이후 바로 PINx Register를 읽을 때

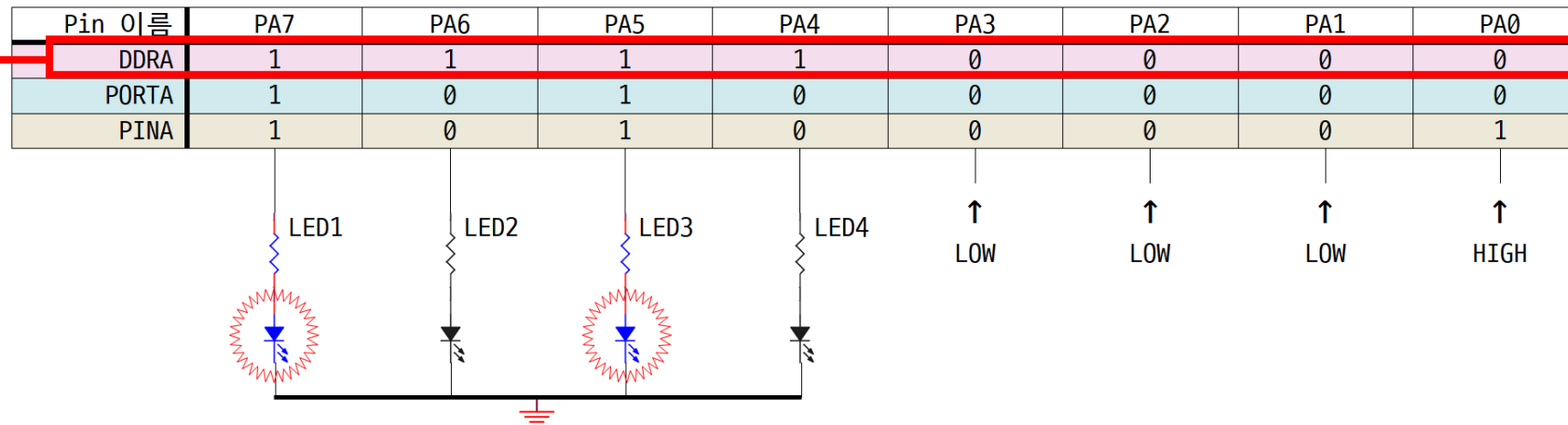
- ATmega128 설명서 68, 69페이지 참고.
- Synchronizer에 의한 Delay를 고려하여
1개 이상의 nop 명령어를 아래 C 코드
예제처럼 삽입해야 함.

```
PORTA = 0xFF;  
asm volatile("nop");  
// 어셈블리어 코드를 사용한다는 의미이다.  
k = PINA;  
// Port A의 상태를 읽어 k에 저장.
```



•AVR GPIO – DDRx, PORTx, PINx 예제

➤예를 들어 Port A에 LED와 외부 입력이 물려 있는 상태에서...

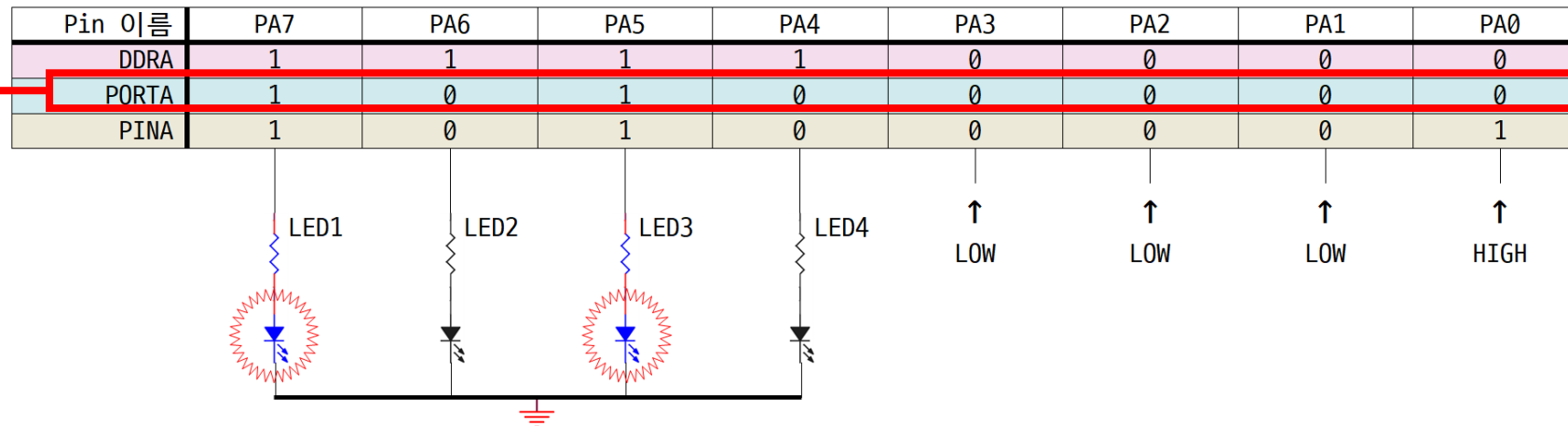


1. PA0 ~ PA3은 외부 신호를 받는 Pin으로 사용 : 해당 DDRA Bit는 0으로 설정
2. PA4 ~ PA7은 LED를 구동하는 Pin으로 사용 : 해당 DDRA Bit는 1으로 설정
3. 따라서 DDRA는 0b11110000(0xF0)로 설정.

- 일반적으로 DDRx Register를 먼저 설정하는 것이 일반적임.
- 입출력 방향이 정해진 다음에 PORTx Register의 값이 의미가 있기 때문.
 - PINx Register는 Port의 입출력 방향과 상관 없이 그 값을 저장하고 있다.

•AVR GPIO – DDRx, PORTx, PINx 예제

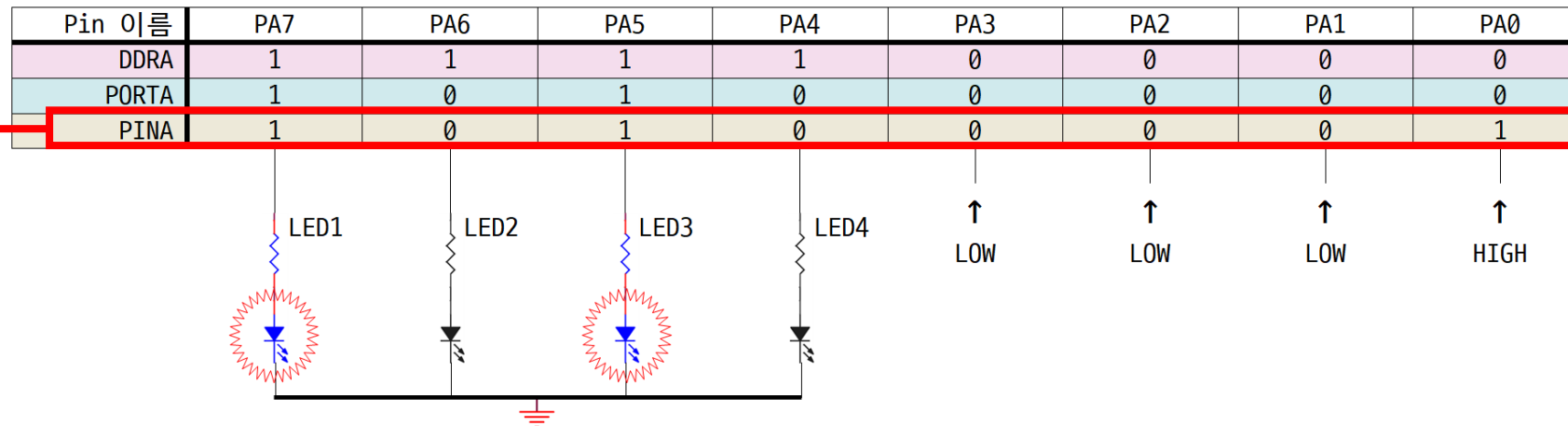
➤예를 들어 Port A에 LED와 외부 입력이 물려 있는 상태에서...



1. LED1과 LED3은 켜고, 나머지 LED는 끄고 싶다면 PA7과 PA5에 해당하는 PORTA만 1로 설정
2. 따라서 PORTA는 0b10100000(0xA0)로 설정.

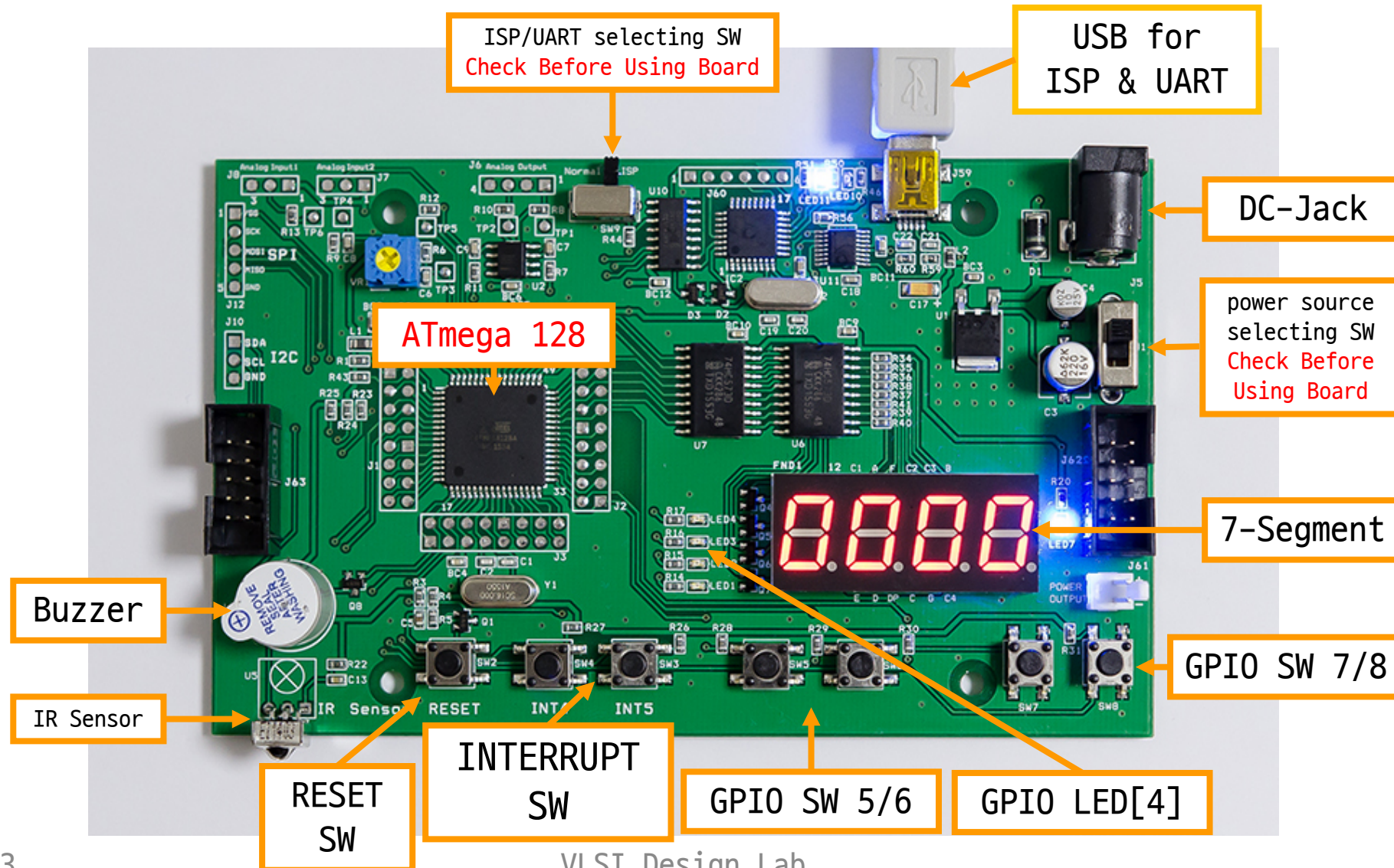
•AVR GPIO – DDRx, PORTx, PINx 예제

➤예를 들어 Port A에 LED와 외부 입력이 물려 있는 상태에서...



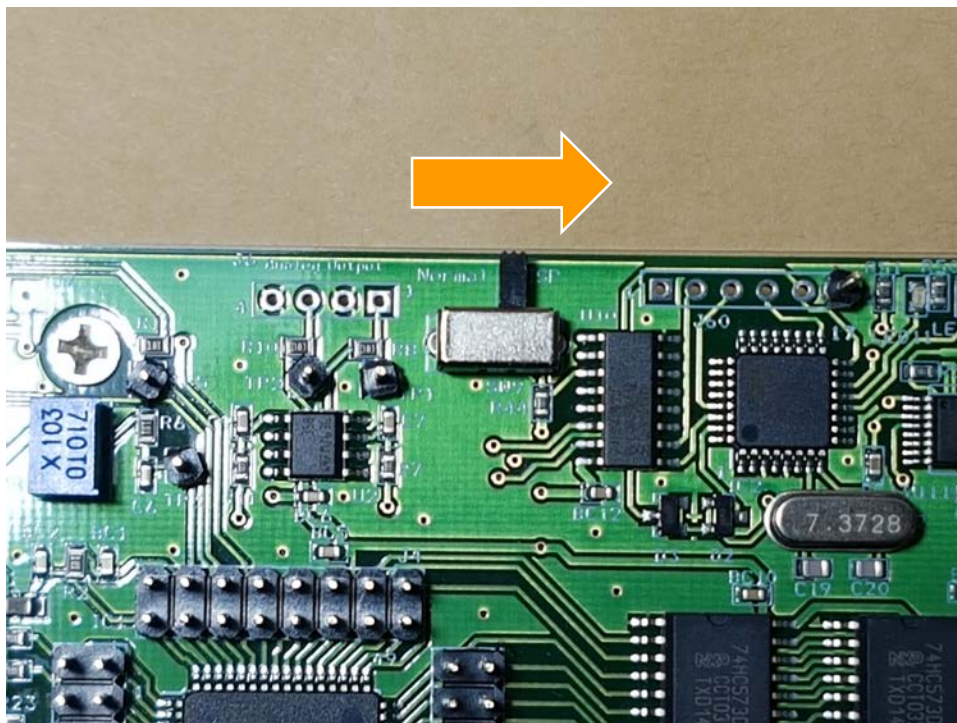
1. 현재 Port A에서 PA7, PA5는 High를 출력하고 있고, PA0에는 외부에서 High 신호가 들어오고 있음
2. 따라서 PINA를 읽으면 0b10100001(0xA1)이 읽힘.

•ATmega128 Training Board (1)

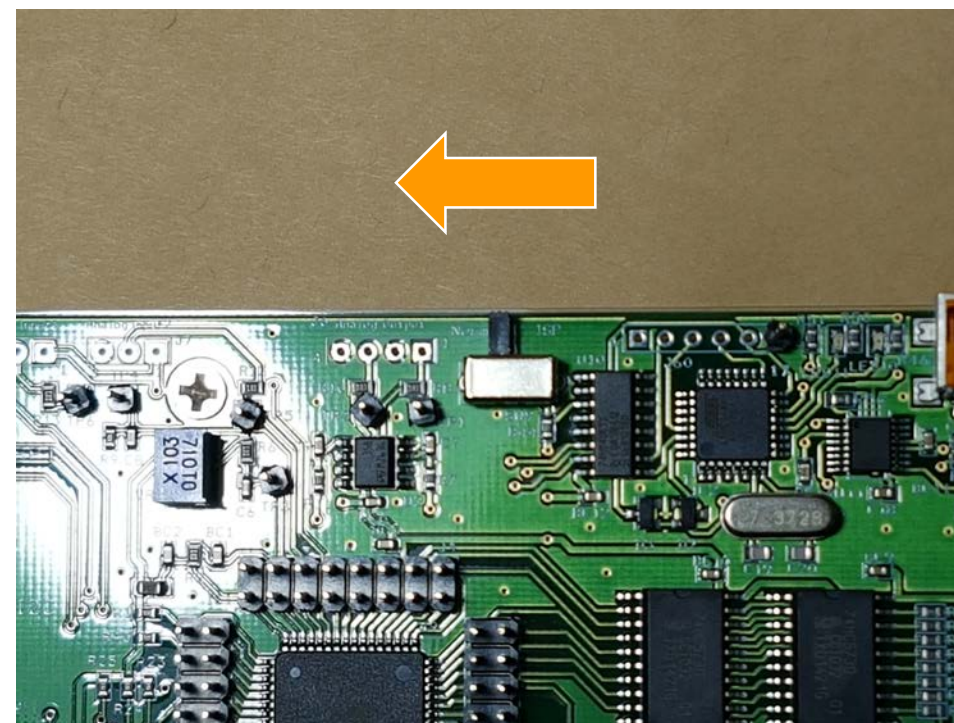


• ATmega128 Training Board (2)

➤ ISP 스위치의 조작



보드에 작성한 프로그램(.hex)을 기록할 때
→ SW9을 ISP방향으로

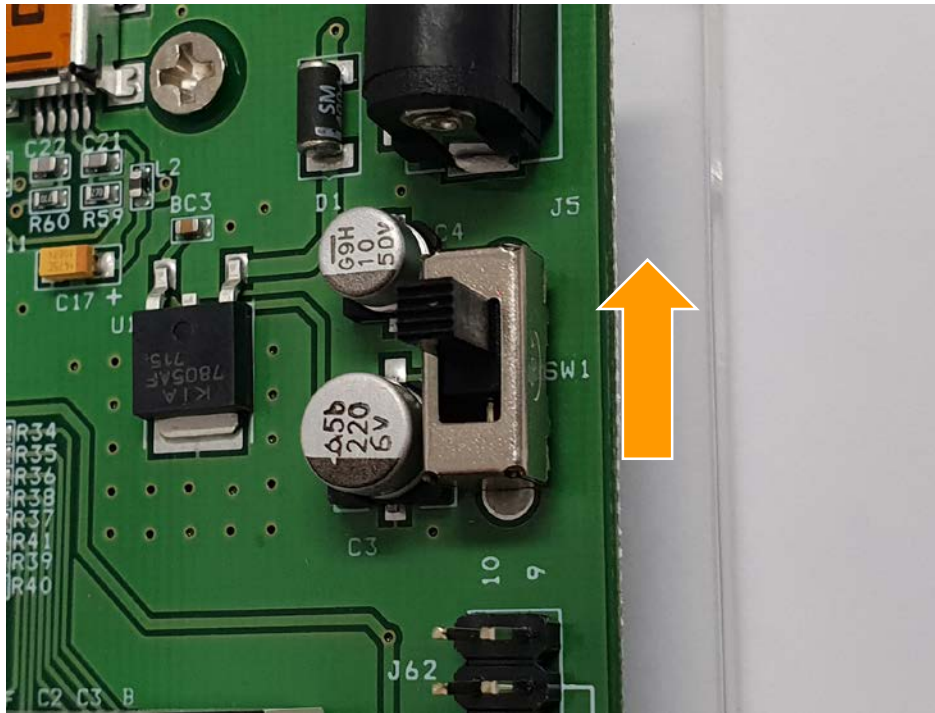


보드의 USART 기능을 사용하고자 할 때
→ SW9을 Normal방향으로

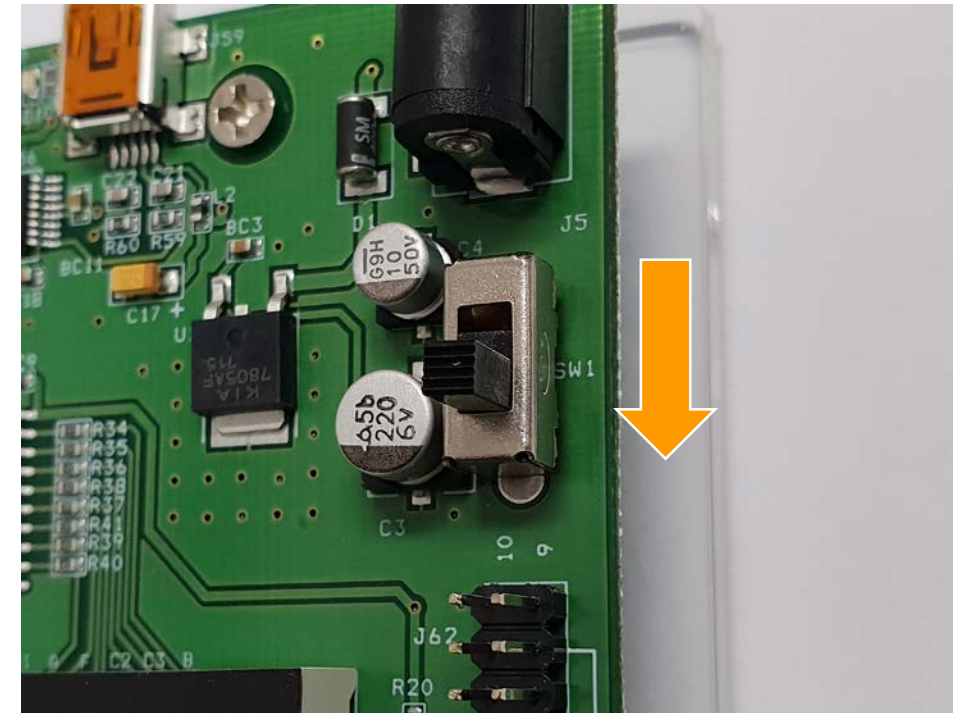
(USART 기능을 사용하지 않더라도, 프로그램을 기록하는 것이 아니면 안정성을 위해 가급적 SW를 Normal방향으로 두고 작동시킵니다)

• ATmega128 Training Board (3)

➤ POWER 스위치의 조작



USB 전원을 사용할 때
-> SW1을 위로
-> 라인트레이서 실습 전까지는 SW1을 위로 올리고 사용합니다.



DC잭 전원을 사용할 때
-> SW1을 아래로
-> 라인트레이서 실습 시 DC잭 전원을 사용합니다. 그때는 SW1을 밑으로 내립니다.

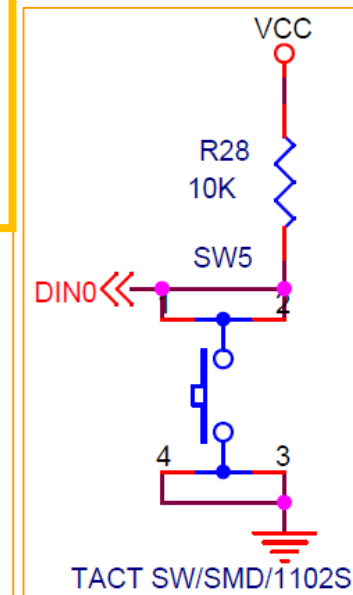
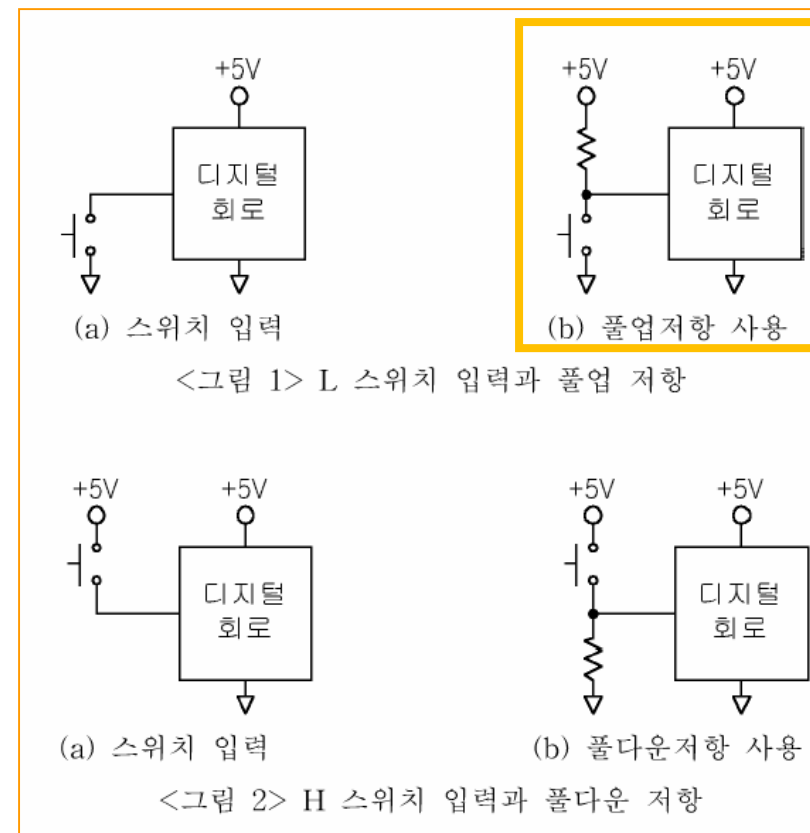
•ATmega128 Training Board (4)

➤ Pull-Up저항

- 외부 I/O에서 명확한 논리 값을 만들기 위해 사용.
- 스위치를 Open하면 Pin이 V_{cc} 와 연결됨
 - 5V 연결 - 명확한 High상태 발생
- 스위치를 Close하면 Pin이 GND와 연결됨
 - 0V 연결 - 명확한 Low 상태 발생
- 본 실험에서 사용하는 개발보드의 모든 Push SW는 Pull-Up저항을 연결함.

➤ Pull-Up/Down저항을 쓰지 않는다면?

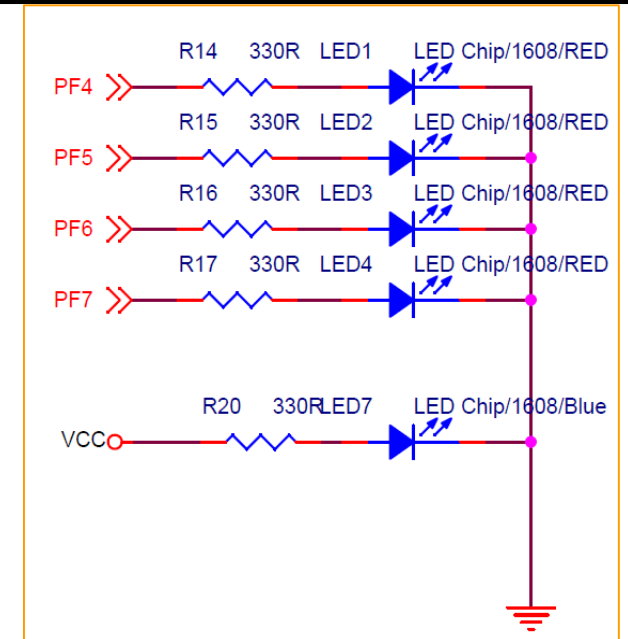
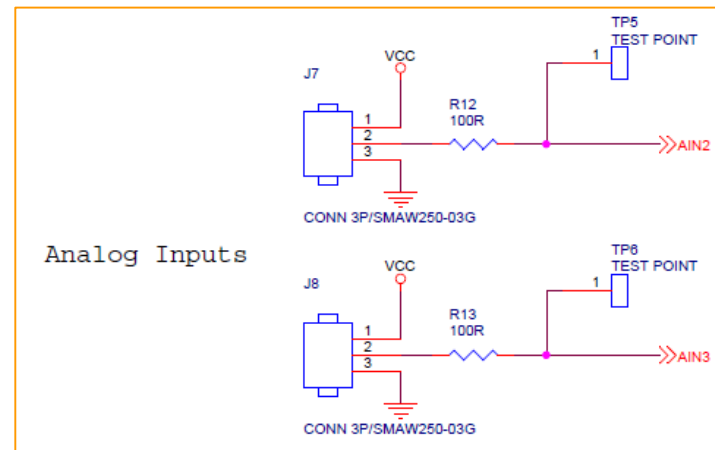
- Pin이 정확히 어떤 상태에 있는지 보장할 수 없음
 - 즉, High인지 Low인지 명확한 근거가 없음.
 - "Floating" 상태라고 함.



•AVR GPIO – 개발 보드에서 설정

➤Schematic 1, 2, 3 Page 참고

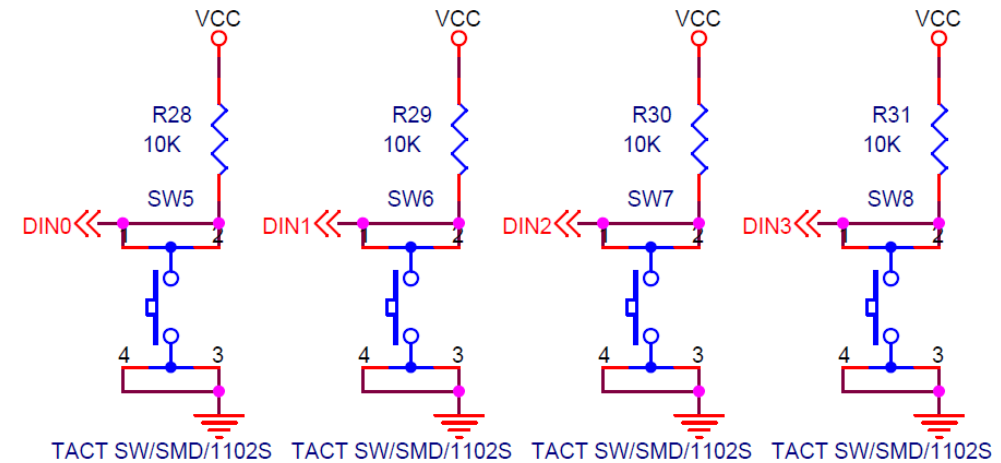
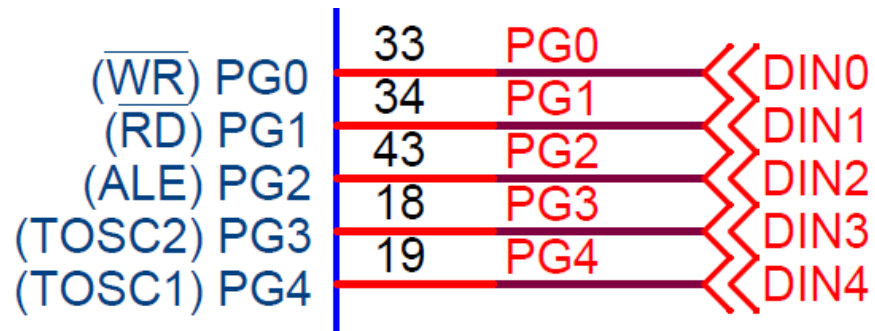
(ADC0) PF0	61	PF0	AIN0
(ADC1) PF1	60	PF1	AIN1
(ADC2) PF2	59	PF2	AIN2
(ADC3) PF3	58	PF3	AIN3
(TCK/ADC4) PF4	57	PF4	PF4
(TMS/ADC5) PF5	56	PF5	PF5
(TDO/ADC6) PF6	55	PF6	PF6
(TDI/ADC7) PF7	54	PF7	PF7



- PF0 ~ PF3는 외부 아날로그 입력을 받도록 설계됨.
 - PF0 ~ PF3는 **입력으로 설정해야** 함.
- PF4 ~ PF7은 LED에 연결되어 있음
 - PF4 ~ PF7을 **출력으로 설정해야** 함.
- 따라서 DDRF = **0xF0**;로 설정해야 함.
- 만약 LED1만 켜고, 나머지 LED는 끄고 싶다면?
 - PORTF = **0x10**; // **0b00010000**

•AVR GPIO - 개발 보드에서 설정

➤Schematic 1, 3 Page 참고



➤Port G의 모든 Pin은 MCU 외부에서 신호를 받는 역할을 함.

➤따라서 PG0 ~ PG4는 **입력으로 설정해야** 함.

➤따라서 DDRG = **0x00**;로 설정해야 함.

•예제 1

```
#include <avr/io.h>
int main()
{
    DDRF = 0xF0;
    PORTF = 0x00;
    DDRC = 0x00;

    while(1)
    {
        switch (PING&0x0F)
        {
            case 0x0E:
                PORTF = 0x01<<4;
                break;
            case 0x0D:
                PORTF = 0x02<<4;
                break;
            case 0x0B:
                PORTF = 0x04<<4;
                break;
            case 0x07:
                PORTF = 0x08<<4;
                break;
            default:
                PORTF = 0x00;
                break;
        }
    }
}
```

•개발환경 설정

➤ 순서대로 진행합니다!!!

1. AVR Tool chain 설치

1. WinAVR 설치
2. avr-toolchain 설치

2. AvrStudio 설치

1. 본 강의에선 AvrStudio 4.19를 사용합니다.

3. 프로젝트 생성 및 설정

4. 프로그램 작성

5. 프로그램 빌드

6. 프로그램 다운로드 및 실행

1.AVR Tool chain

1. 배포한 파일 중 WinAVR-20100110-install.exe를 실행시킵니다.

➤ 긍정적인 대답을 선택하여 설치를 완료합니다.

2. 배포한 파일 중 avr-toolchain-installer-3.3.0.710-win32.win32.x86.exe를 실행시킵니다.

➤ 긍정적인 대답을 선택하여 설치를 완료합니다.

➤ 설치할 때 설치 경로를 기억합니다.

➤ 변경하지 않으면 기본적으로 C:\WinAVR-20100110에 설치됨.

➤ Tool chain??

➤ AVR 개발 보드에서 AVR 프로그램을 개발 및 컴파일 할 수 있을까요?

➤ 사실상 불가능

➤ 따라서 AVR보다 고성능의 PC에서 개발함. -> Cross Compile!!!

➤ 작성한 소스코드를 AVR 프로세서에 맞도록 컴파일 및 링킹 할 컴파일러와 링커, 그리고 라이브러리 등이 필요함.

➤ 필요한 개발 도구를 모은 집합을 Tool chain이라고 함.

2. AvrStudio 설치

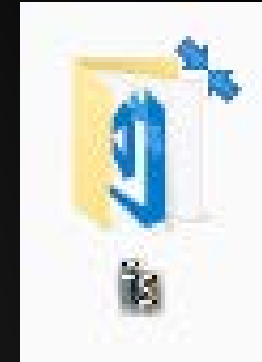
1. AvrStudio419Setup.exe를 실행시킵니다.

1. 긍정적인 대답을 선택하여 설치를 완료합니다.
2. Jungo USB 설치도 함께 진행합니다.

3. 프로젝트 생성

1. 폴더 생성

1. 새로운 프로그램을 작성할 때 마다 새로운 폴더를 생성하여 프로젝트 끼리 헛갈려서 문제가 생기지 않도록 합니다.
2. 폴더 경로에는 절대 한글이 있으면 안됩니다.
 - 빌드 불가능합니다.
 - 상당수의 외산 개발 툴은 경로에 한글이 있으면 안됩니다.



```
Build started 15.4.2019 at 05:50:28
• make: Makefile: No such file or directory
make: *** Ã, "Û `Makefile',.¡ ..µé ±ÔÃcÀì %œÃ%. .0Ãã.
Build failed with 1 errors and 0 warnings...
```

2. AvrStudio 실행

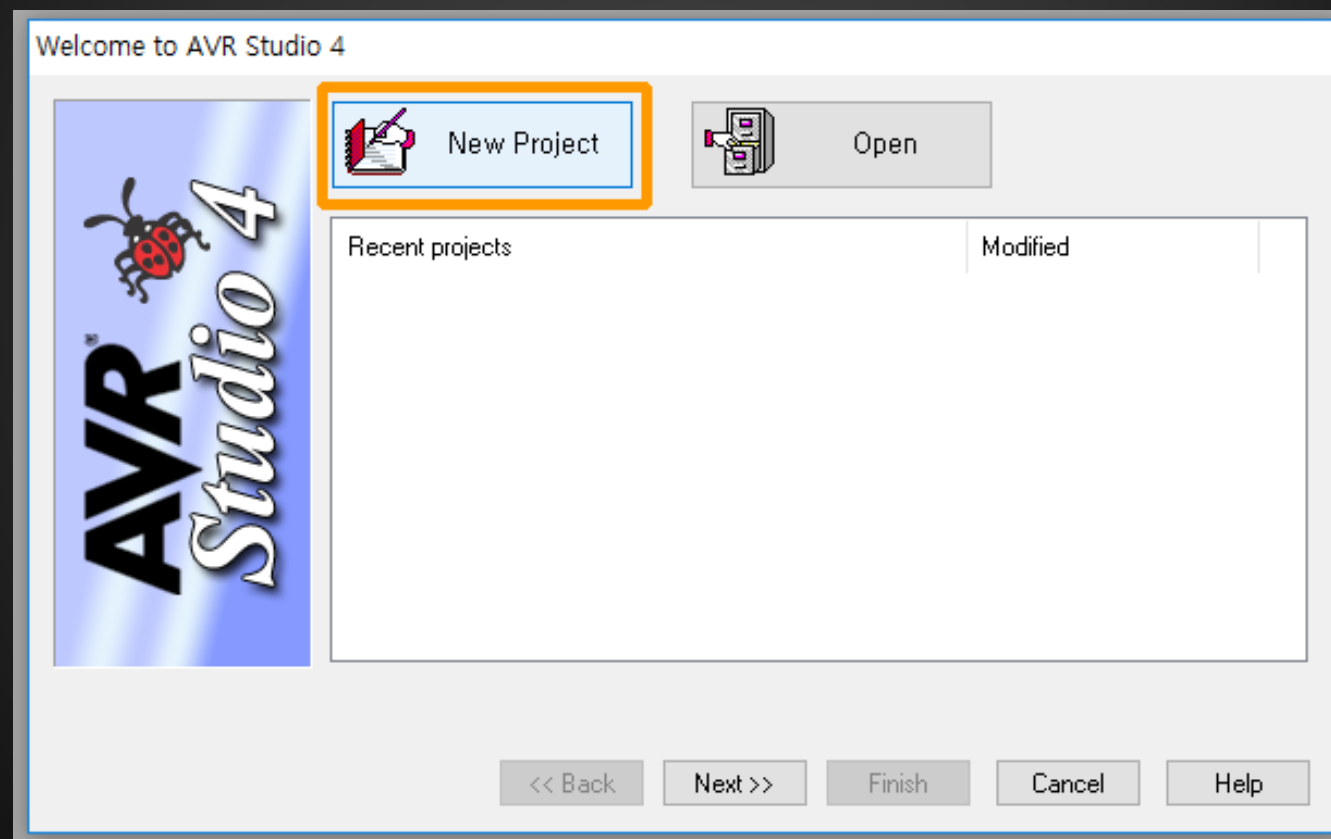
1. 시작 - Atmel AVR Tools 클릭



3. 프로젝트 생성

3. 새 프로젝트 생성

➤ New Project 클릭



3. 프로젝트 생성

4. 프로젝트 옵션 설정

1. AVR GCC 선택

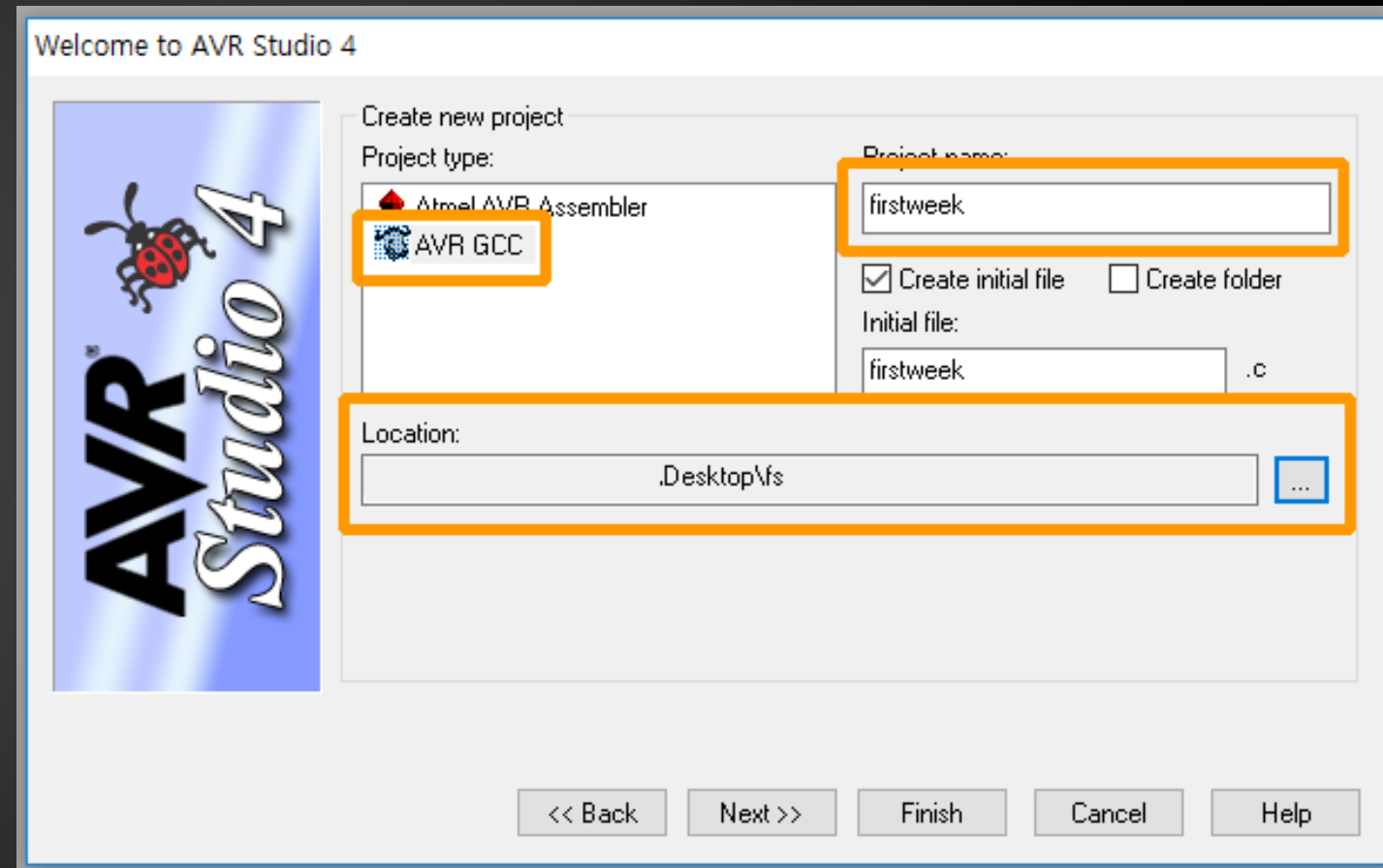
2. Project name 설정

1. 반드시 영문자 및 숫자
2. 첫 글자는 숫자를 사용하지 말 것
3. 띄어쓰기 없이

3. Location 설정

1. 맨 처음에 생성한 폴더로 설정

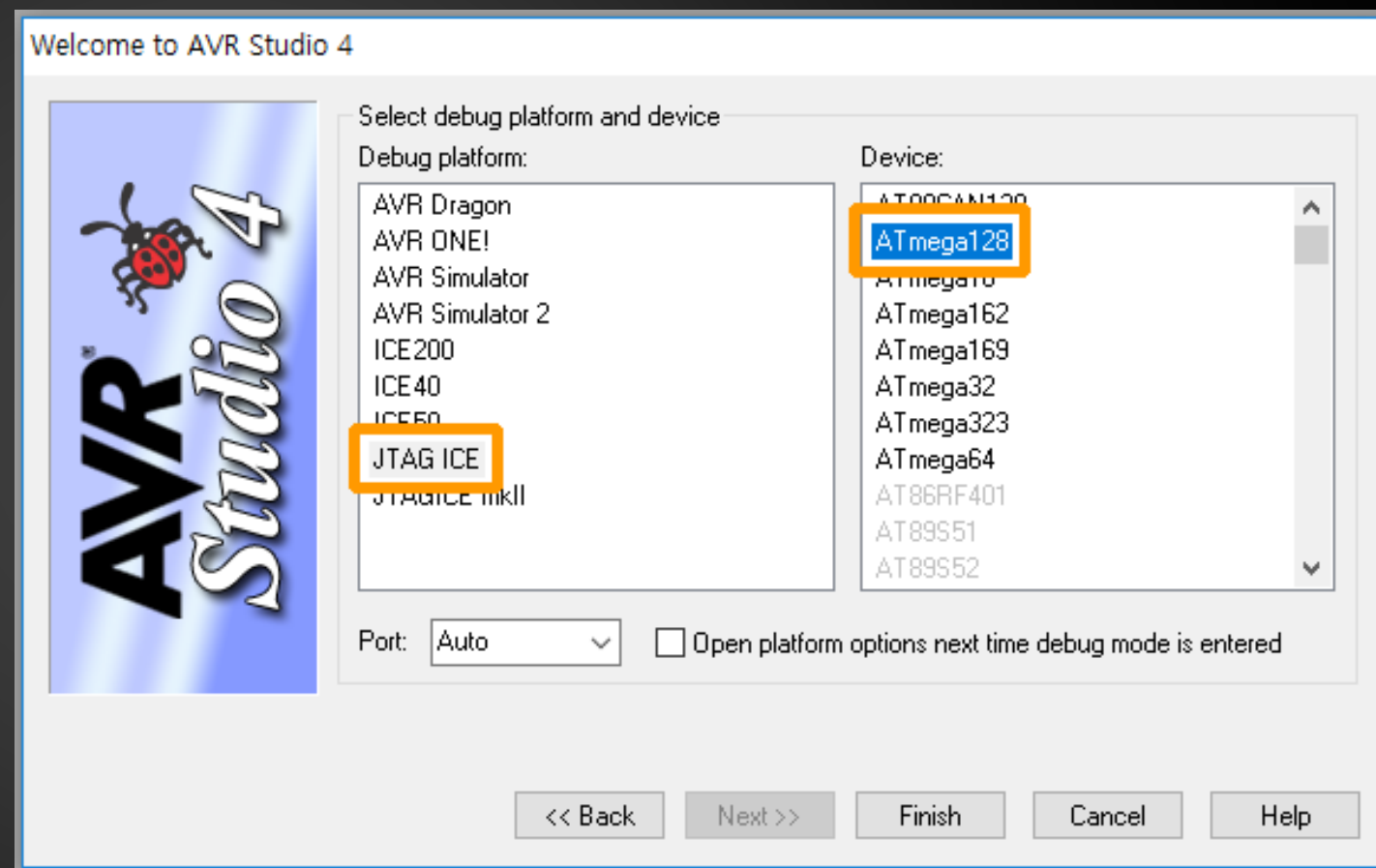
4. 하단에 Next >> 클릭



3. 프로젝트 생성

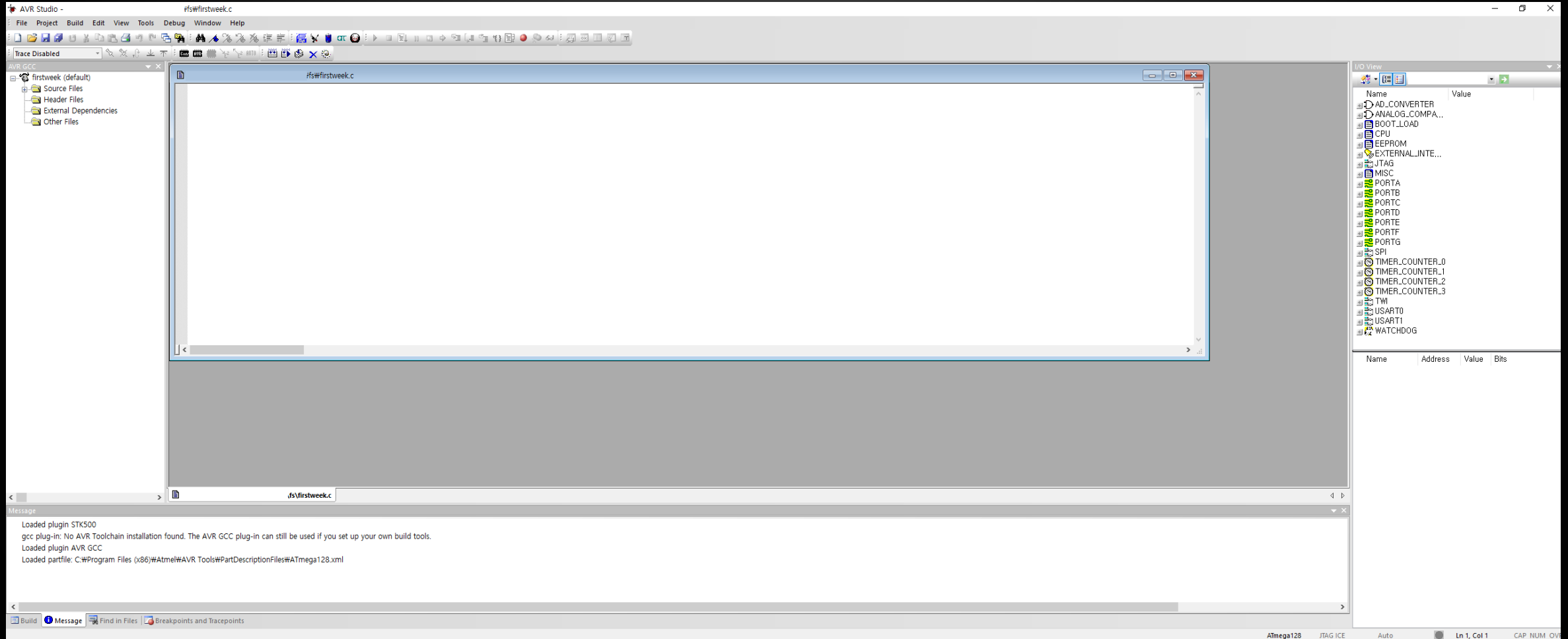
5. 디버그 플랫폼 설정

1. JTAG ICE 선택
2. ATmega128 선택
3. Finish 클릭



3. 프로젝트 생성

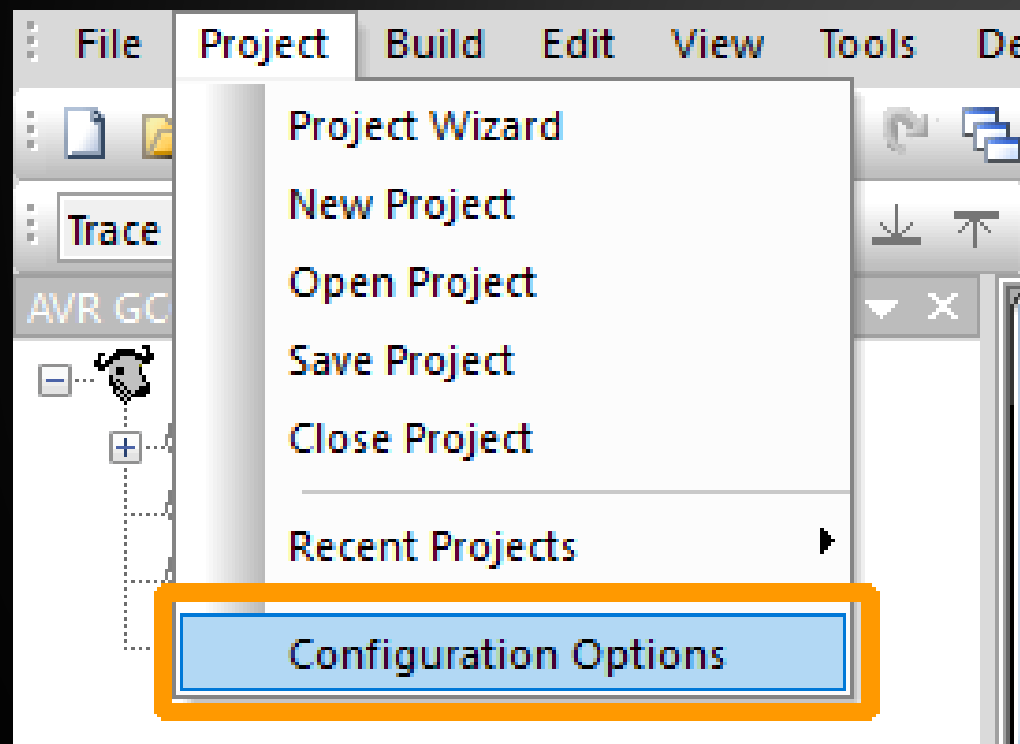
6. 프로젝트 생성 완료



3. 프로젝트 생성

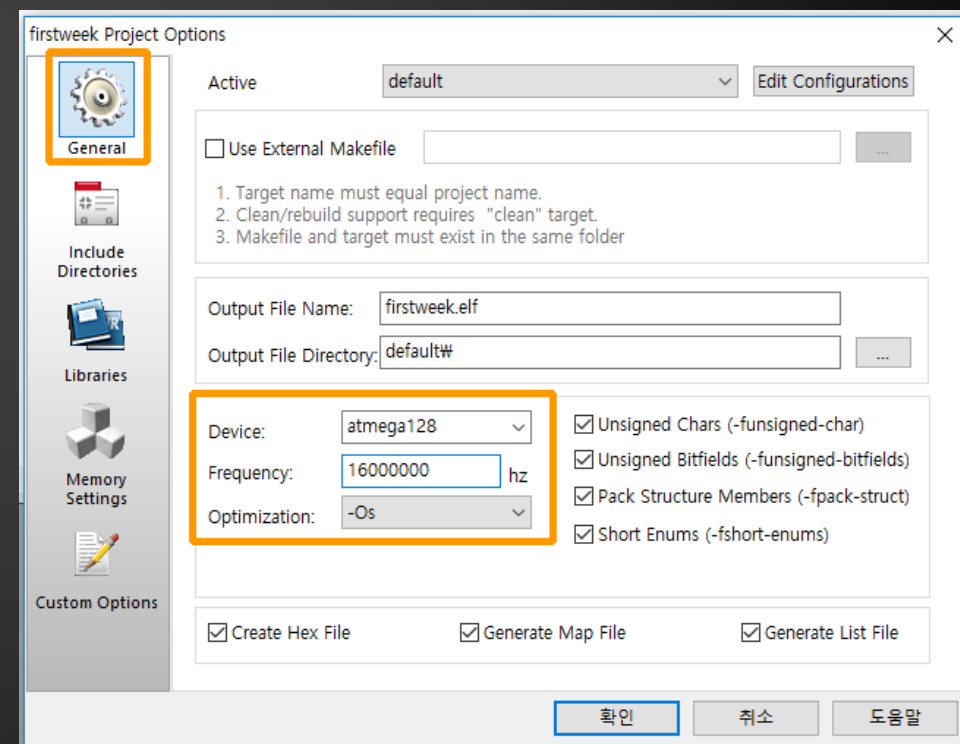
7. 프로젝트 설정

1. 좌측 상단에서 Project - Configuration Option 선택



2. General에서 아래와 같이 세팅

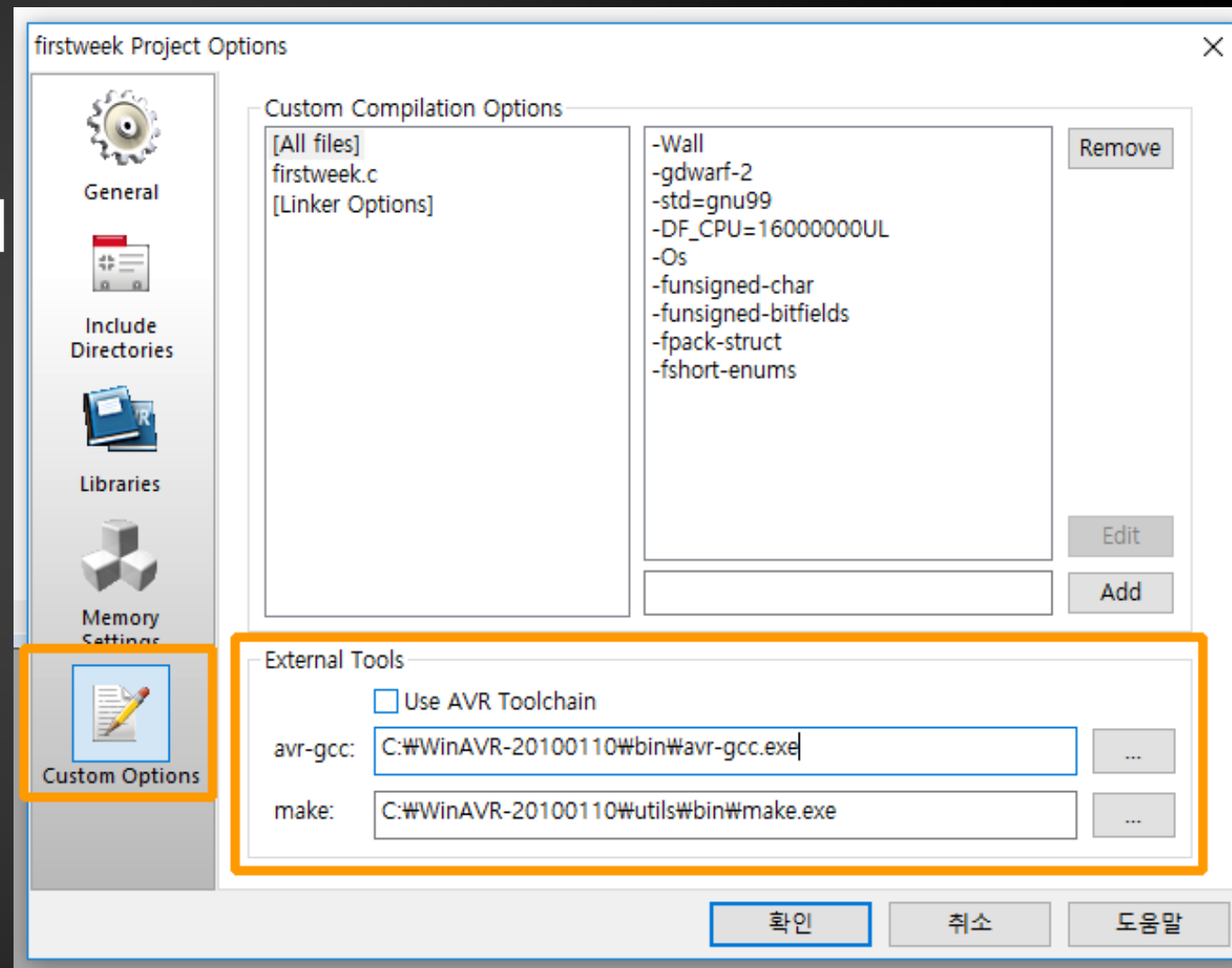
- Optimization은 일부 예제에서 변경해야 하는 경우가 있습니다.



3. 프로젝트 생성

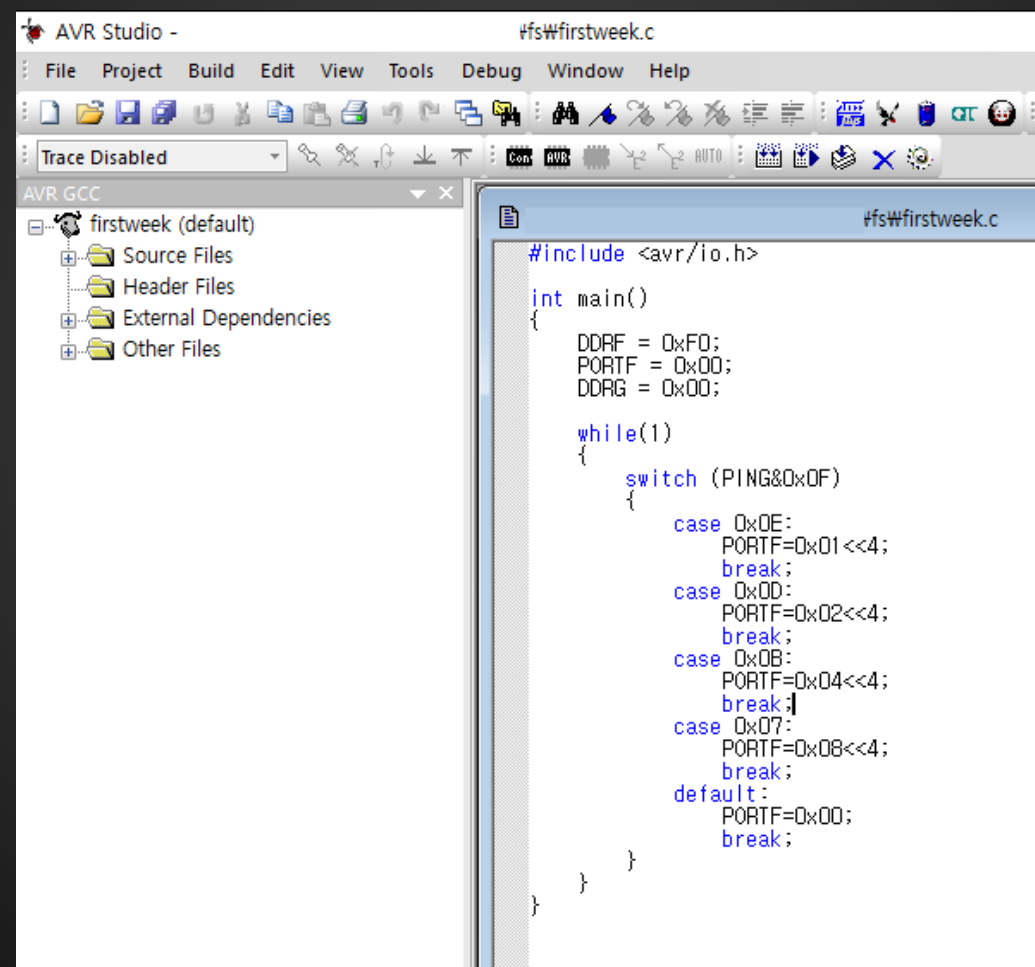
7. 프로젝트 설정

3. 하단의 Custom Option 클릭
4. 설치한 WinAVR-20100110 내의 avr-gcc, make파일로 tools 설정함.
 - 우측 사진을 참고한다.
 - Use AVR Toolchain 설정을 해제해야한다.
5. 확인 클릭하여 설정 종료



4. 프로그램 작성

➤ 편집기 창에 소스코드를 입력합니다. (예제 1 참고)



The screenshot shows the AVR Studio IDE with the file #fs#firstweek.c open in the editor. The code defines a main function that initializes DDRF, PORTF, and DDRA registers. It then enters a while loop that checks the state of PINB register bit 0 and sets PORTF bits 0 through 3 accordingly.

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

int main()
{
    DDRF = 0xFF;
    PORTF = 0x00;
    DDRA = 0x00;

    while(1)
    {
        switch (PINB & 0x01)
        {
            case 0x01:
                PORTF = 0x01 << 4;
                break;
            case 0x00:
                PORTF = 0x02 << 4;
                break;
            case 0x02:
                PORTF = 0x04 << 4;
                break;
            case 0x03:
                PORTF = 0x08 << 4;
                break;
            default:
                PORTF = 0x00;
                break;
        }
    }
}
```

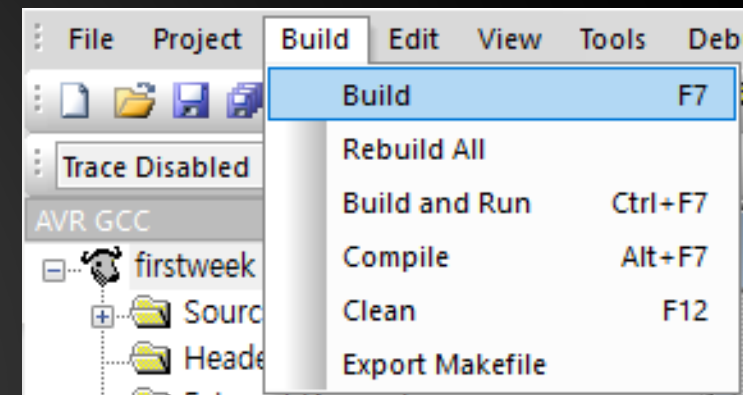
5. 프로그램 빌드

1. 상단의 Build - Build를 클릭합니다.

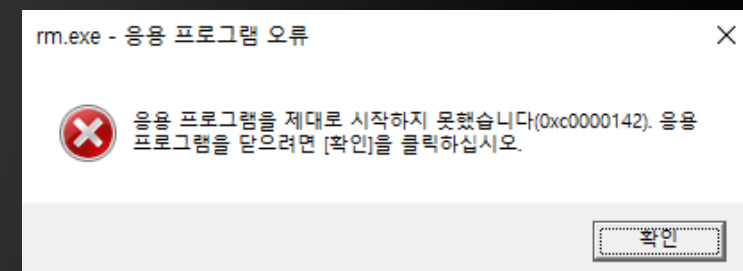
➤ 단축키 F7

➤ 소스코드 수정이 발생하는 경우, 반드시 저장하고 빌드합니다.

➤ 편집기에 보이는 코드가 아니라, 파일에 저장된 코드를 빌드합니다.



2. (PC가 Windows Vista 이상인 경우) 빌드 과정에서 오류가 발생하는 것을 확인합니다.



```
0 [main] sh 23156 sync_with_child: child 25224(0x1F4) died before initialization with status code 0xC0000142
3442 [main] sh 23156 sync_with_child: *** child state waiting for longjmp
/usr/bin/sh: fork: Resource temporarily unavailable
avr-gcc -mmcu=atmega128 -Wall -gdwarf-2 -std=gnu99 -DF_CPU=16000000UL -Os -funsigned-char -funsigned-bitfields
../firstweek.c:30: fatal error: opening dependency file dep/firstweek.o.d: No such file or directory
compilation terminated.
make: *** [firstweek.o] Error 1
Build failed with 1 errors and 0 warnings...
```


5. 프로그램 빌드

3. WinAVR-20100110 파일 패치

- 앞 Slide와 동일한 오류가 발생하는 경우만 진행
 - 빌드 과정에서 오브젝트 파일의 의존성 문제가 발생하는 경우
- 다른 오류가 생긴다면 조교에게 질문하세요.

1. msys-1.0-vista64.zip의 압축을 푼다

➤ msys-1.0.dll을 확인한다.

2. C:\WinAVR-20100110\utils\bin 안의 msys-1.0.dll와 앞에서 압축을 푼 파일을 교체한다.

3. 파일을 교체하고 나서, 빌드를 다시 진행한다.

4. 빌드 결과를 확인한다.

```
Program:      284 bytes (0.2% Full)
(.text + .data + .bootloader)
```

```
Data:          0 bytes (0.0% Full)
(.data + .bss + .noinit)
```

```
Build succeeded with 0 Warnings...
```

6. 프로그램 다운로드 및 실행

1. AVR 개발보드와 PC를 연결한다.

1. 배부한 USB 케이블을 이용해 연결한다.
2. 개발보드의 전원 스위치, ISP 스위치 상태를 반드시 확인한다.

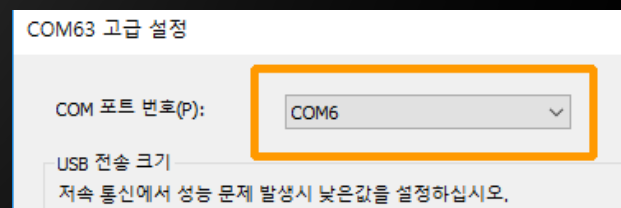
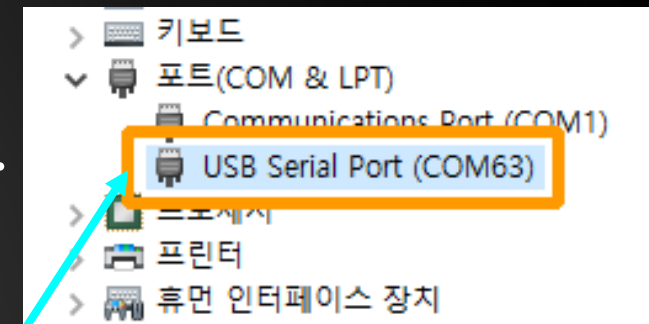
2. 장치관리자에서 연결 상태를 확인한다.

➤ 안 뜨는 경우, 2 ~ 3분 정도 기다립니다.

1. 연결한 AVR보드가 몇 번 COM 포트를 할당 받았는지 확인한다.
2. 만일 COM포트 번호가 10을 넘어 가는 경우, 번호를 다시 할당 해준다.

➤ 10을 넘어가지 않는 경우 하지 않아도 된다.

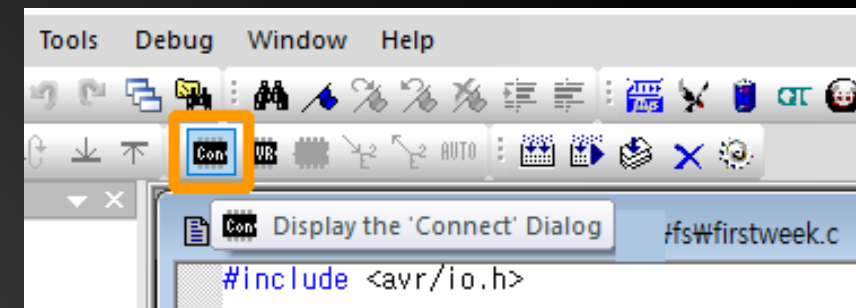
1. 장치관리자에서 해당 디바이스 오른쪽 클릭 - 설정 - 포트설정 - 고급 클릭
2. 설정창에서 **현재 연결된 다른 디바이스와 겹치지 않는 번호를 선택하여** 재할당(사용중 이라고 떠있어도, 현재 연결되어 있지 않으면 상관 없음. 9 이하로 선택.)



6. 프로그램 다운로드 및 실행

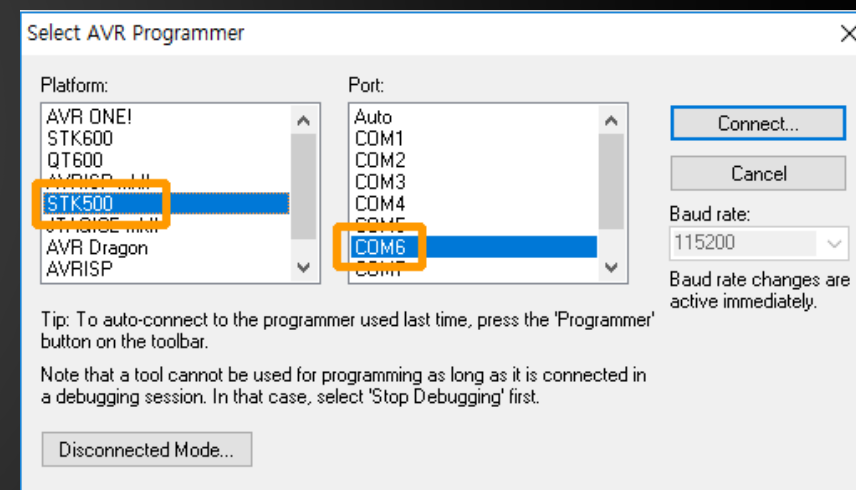
3. AVR Studio에서 Connect 버튼 클릭

➤또는 상단 Tools - Program AVR - Connect 클릭



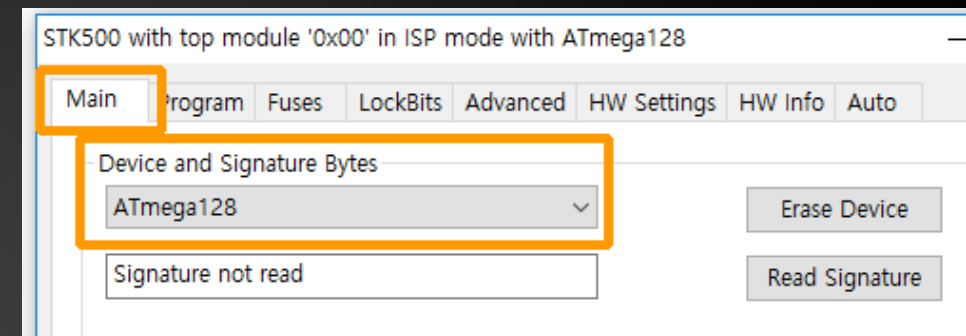
4. Platform은 STK500, Port는 앞 Slide에서 확인/설정된 포트 번호 선택

➤선택하고 나서 Connect 클릭



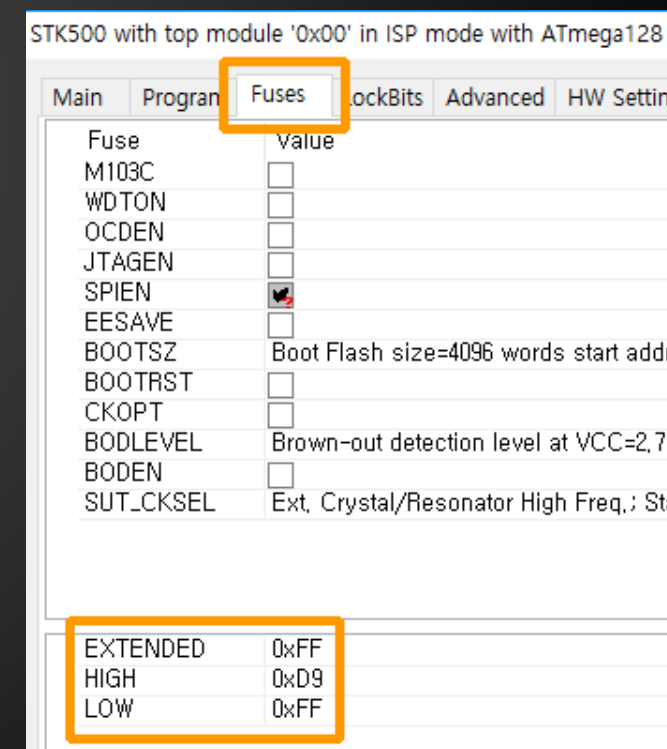
6. 프로그램 다운로드 및 실행

5. 상단의 Main을 클릭하여 ATmega128로 설정되어 있는지 확인



6. 상단의 Fuses를 클릭하여 우측 그림처럼 설정되었는지 확인

- Fuses는 연구실에서 보드 배부전에 미리 설정 완료 하였습니다.
- Fuses값은 ATmega128 칩의 기능을 설정합니다.
- 일부 값은 한번 바꾸면 수정하기 상당히 힘들어집니다. **절대 함부로 바꾸지 마세요.**



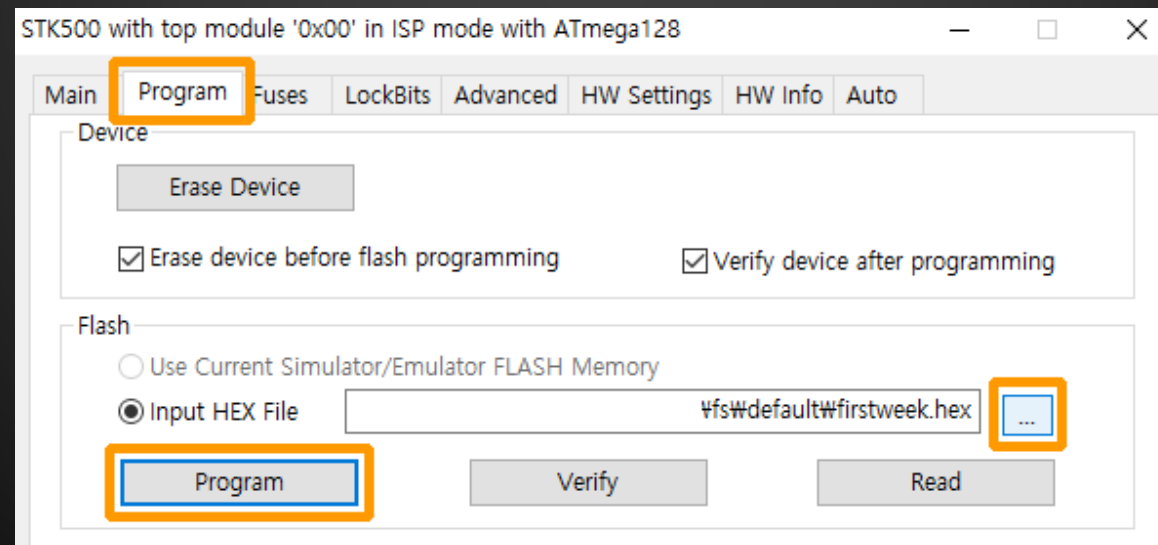
6. 프로그램 다운로드 및 실행

7. 상단의 Program을 클릭하여 빌드한 hex파일을 찾는다.

➤본인이 생성한 프로젝트 폴더 내의 default 폴더 안에 (프로젝트 이름).hex 파일이다.

8. Program 버튼을 클릭한다.

9. 프로그램이 AVR 개발보드에서 동작하는지 확인한다.



•예제 1의 동작

- 개발 보드의 SW 5,6,7,8 스위치를 누르면서 LED 동작을 확인한다.
- 만약 보드에서 소리가 나면 뭔가 잘못된 것입니다.
- **레포트 반영 내용 1** : switch - case문에서 왜 PING값 대신 PING&0x0F 값을 확인했는가?
 - "floating"과 연관 지어 생각해본다. 본 개발보드 Schematic 1 페이지에는 DIN4가 PG4에 연결된 것으로 작성되어 있으나, Schematic 그 어디에도 DIN4가 어디에 어떻게 연결되어 있는지 나와있지 않다. 다시 말해 PG4는...?
 - 개발할 때 항상 Schematic을 보는 습관을 기른다.
- **레포트 반영 내용 2** : switch - case문에서 label 값은 왜 0x01, 0x02, 0x04, 0x08이 아니라, 0x0E, 0x0D, 0x0B, 0x07을 사용했는가?
 - 각 16진수 숫자를 2진수로 바꾸고 나서, 버튼 회로에 Pull-Up저항이 장착된 것을 연관 지어 생각해 본다.