

AVR_1

임베디드스쿨2기

Lv1과정

2020. 05. 15

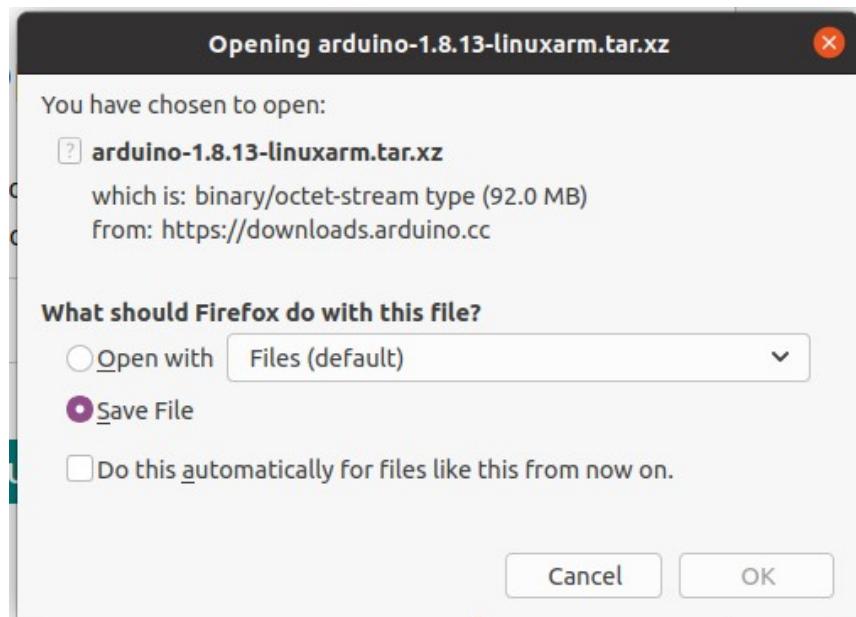
박태인

1. Ubuntu Linux 20.04에서 Arduino 기본 설정하기

1) 먼저 아래 사이트에서 아두이노를 다운 받는다.

<https://www.arduino.cc/en/donate/>

Just Download 눌러서 다운



Support the Arduino IDE

Since the release 1.x release in March 2015, the Arduino IDE has been downloaded **51,395,885** times — impressive! Help its development with a donation.

\$3 \$5 \$10 \$25 \$50 Other

JUST DOWNLOAD **CONTRIBUTE & DOWNLOAD**

1. Ubuntu Linux 20.04에서 Arduino 기본 설정하기

1) 다운 받은 파일을 압축 해제 한다.

- 먼저 적정한 위치로 이동시킨다.

- 압축을 해제한다.

아래는 터미널에서 압축 해제 명령!



→ [tar xvf arduino-1.8.13-linux64.tar.xz](#)

압축해제 후 생긴 폴더

1. Ubuntu Linux 20.04에서 Arduino 기본 설정하기

1) USB 통신을 정상적으로 동작하기 위한 dfu-util 설치

- 자동으로 설치하는 부분에 버그가 있어서 문제가 발생 할 수 있기에
수동 설치를 진행해야 한다.
- 먼저 이를 수행하기 위해 반드시 필요한 라이브러리가 있으며,
명령어는 아래와 같다.

Sudo apt-get update

Sudo apt-get install libsub-1.0-0-dev

2) 소프 포지로 가서 dfu-util을 다운로드 받는다.



<https://sourceforge.net/projects/dfu-util/files/latest/download>

1. 적당한 위치로 이동시킨다.
2. 압축을 해제한다.

→ **tar zxvf dfu-util-0.10.tar.gz**

1. Ubuntu Linux 20.04에서 Arduino 기본 설정하기

Dfu-util 설치를 시작한다.

1. cd dfu-util

→ dfu-util 압축 해제 한 곳으로 이동

2. ./configure

→ 실행

3. make

→ 에러 발생시 sudo 등의 명령어가 추가 되어야 할 수 있음.

4. make install

→ 위와 마찬가지

1. Ubuntu Linux 20.04에서 Arduino 기본 설정하기

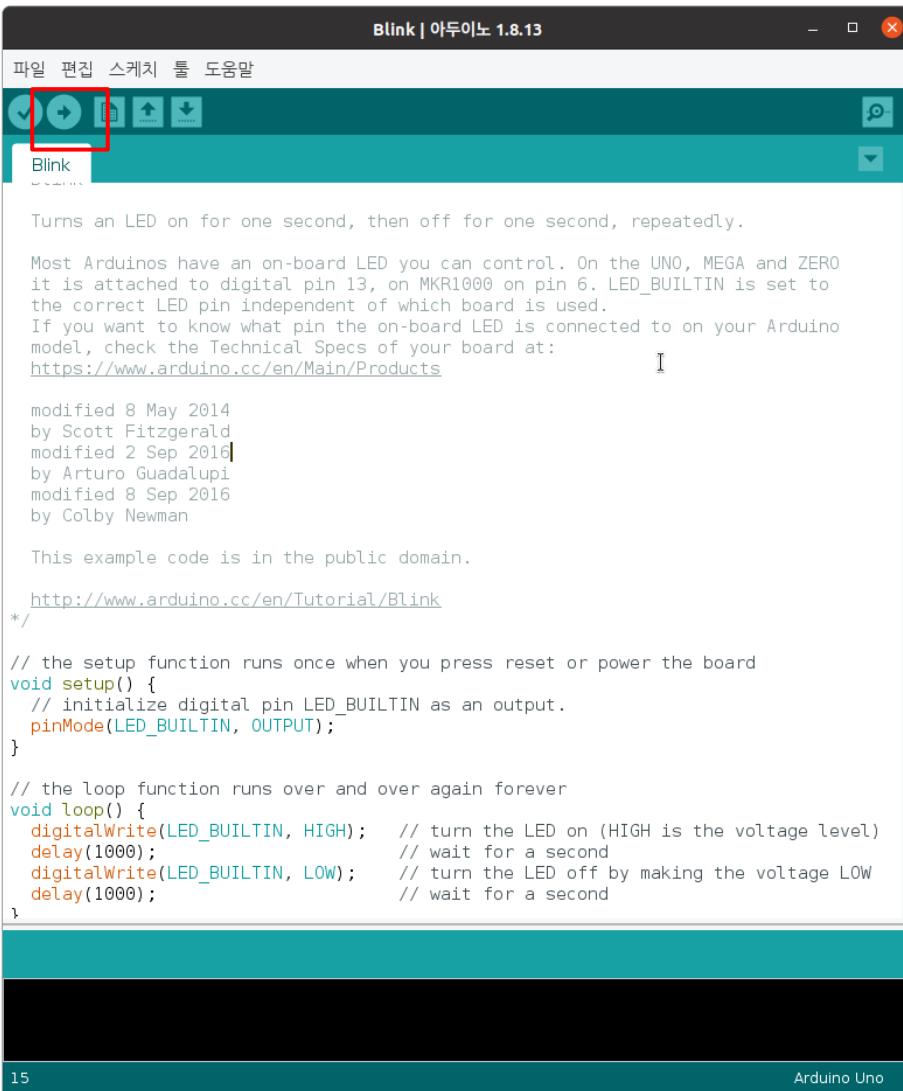
Arduino 프로그램을 연다.

1. Tool → Board → Board Manager 를 연다.
2. uno를 검색하고 Installed 가 보이지 않는다면 Install을 클릭하여 설치 한다.



1. Ubuntu Linux 20.04에서 Arduino 기본 설정하기

구동 테스트를 하기 위해 아래 코드를 넣는다.



```
Blink | 아두이노 1.8.13
파일 편집 스케치 툴 도움말
Blink
Turns an LED on for one second, then off for one second, repeatedly.

Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to the correct LED pin independent of which board is used.
If you want to know what pin the on-board LED is connected to on your Arduino model, check the Technical Specs of your board at:
https://www.arduino.cc/en/Main/Products

modified 8 May 2014
by Scott Fitzgerald
modified 2 Sep 2016
by Arturo Guadalupi
modified 8 Sep 2016
by Colby Newman

This example code is in the public domain.

http://www.arduino.cc/en/Tutorial/Blink
*/
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on (HIGH is the voltage level)
  delay(1000);                      // wait for a second
  digitalWrite(LED_BUILTIN, LOW);     // turn the LED off by making the voltage LOW
  delay(1000);                      // wait for a second
}
15
Arduino Uno
```

코드

```
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on (HIGH is the voltage level)
  delay(1000);                      // wait for a second
  digitalWrite(LED_BUILTIN, LOW);     // turn the LED off by making the voltage LOW
  delay(1000);                      // wait for a second
}
```

코드 입력 후 업로드 버튼
누르면 되나 현재 STK500 에러 발생 중..
(그래서 이후엔 다른 방법 사용 해볼 것임)



```
digitalWrite(LED_BUILTIN, HIGH);    // turn the LED on (HIGH is the voltage level)
delay(1000);                      // wait for a second
digitalWrite(LED_BUILTIN, LOW);     // turn the LED off by making the voltage LOW
delay(1000);                      // wait for a second
}

보드에 업로딩중에 문제 발생. 다음을 참고하세요. http://www.arduino.cc/en/Guide/Troubleshoot 오류 메시지 복사
avrduude: ser_open(): can't open device "/dev/ttyS4": Permission denied
보드에 업로딩중에 문제 발생. 다음을 참고하세요. http://www.arduino.cc/en/Guide/Troubleshoot
15
Arduino Uno on /dev/ttyS4
```

1. Ubuntu Linux 20.04에서 Arduino 기본 설정하기

Codeblock 프로그램 사용 후 ISP로 컴파일

- 우선 아래와 같이 Linux에 AVR 코딩을 위한 기본 설정을 합니다.

1. sudo apt-get update
2. sudo apt-get install codeblocks

- Linux 용 시리얼 프로그래머 설치

1. sudo add-apt-repository ppa:pmjdebruijn/avrdude-release
2. sudo apt-get update
3. sudo apt-get install libc6-dev-i386
4. sudo apt-get install binutils-avr gcc-avr avr-libc avrdude libc6-dev-i386

- 설치확인(codeblocks, avrdude)

1. Ubuntu Linux 20.04에서 Arduino 기본 설정하기

- 기본 예제 돌려 보기

1. 적당한 위치로 이동한다.
2. mkdir -p 적당한위치/proj/avr
3. Codeblock에서 File -> New -> Project -> AVR Project
4. Next
5. Project title: blink

Folder to create project in: 위치(위에서 잡은 위치로 지정하는 것이 좋음)

project filename: blink.cbp

Resulting filename: ~~~

Next

6. Uncheck: Create "Debug" configuration

Next

7. Please choose a processor for this project: atmega328p

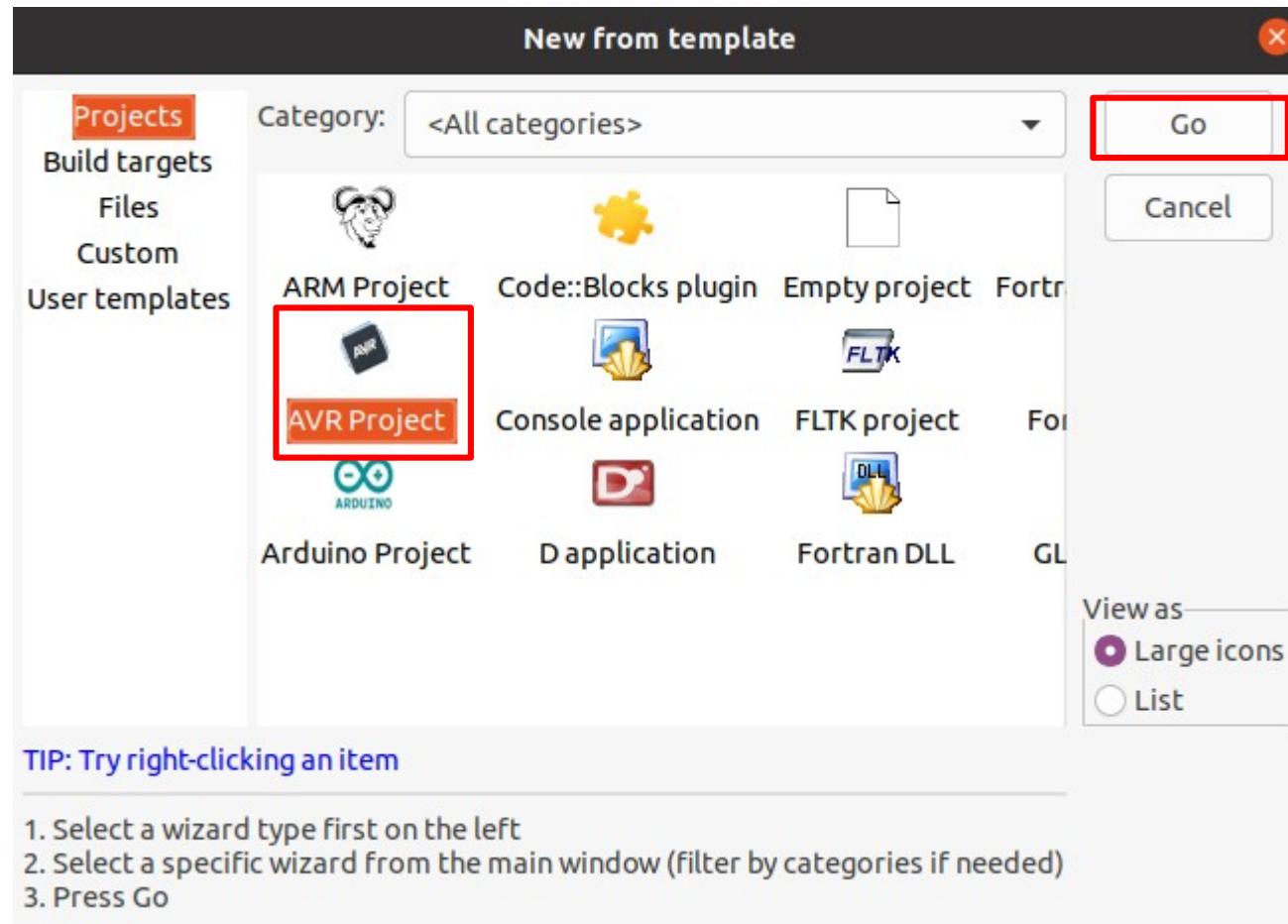
Uncheck: Create Fuse, Lock, Signature ~~~

8. Finish

↳ 위의 새 프로젝트 생성 과정은 다음 페이지부터 스크린샷 더 참조.

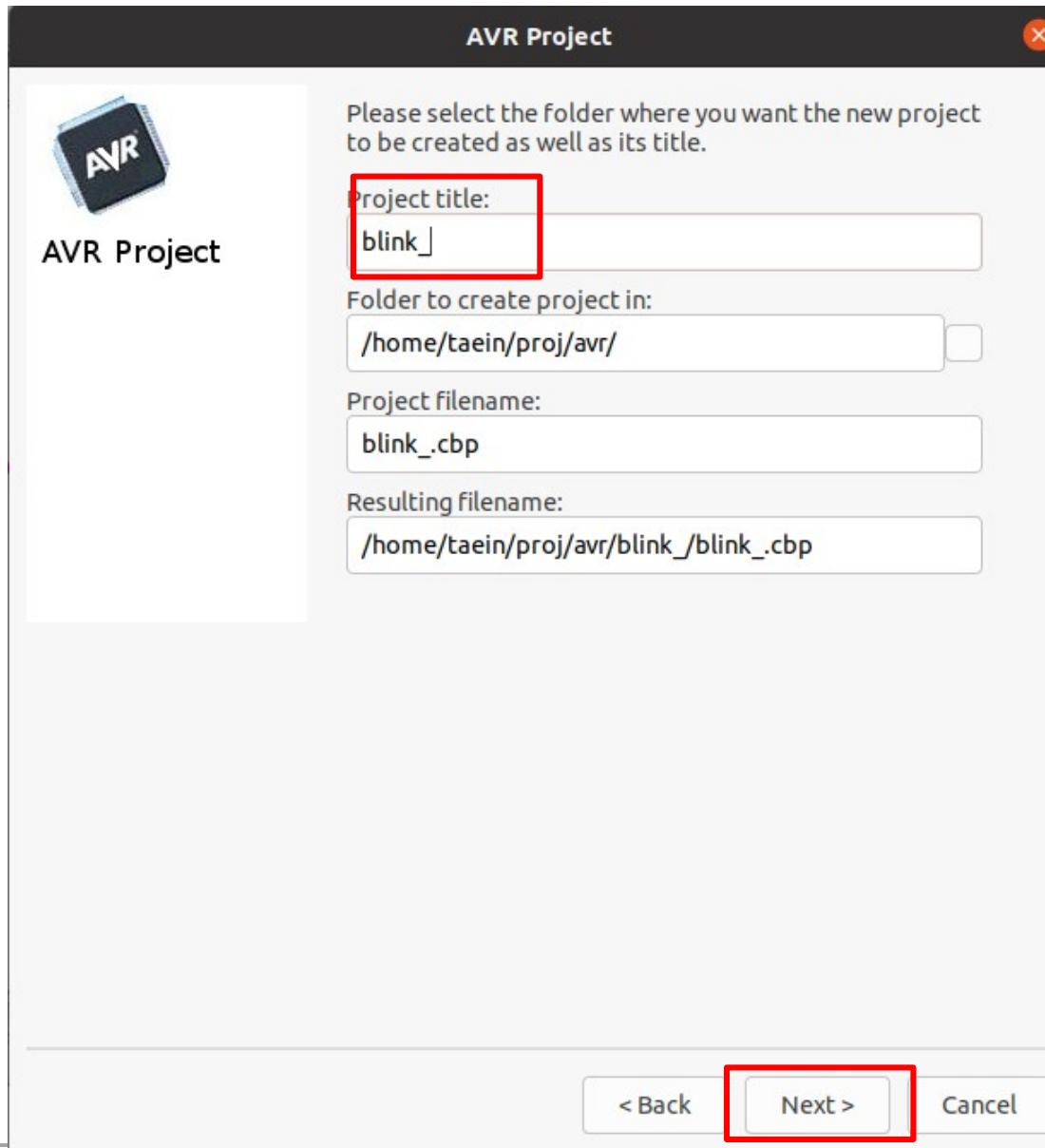
1. Ubuntu Linux 20.04에서 Arduino 기본 설정하기

Codeblock 새 프로젝트 생성 과정.



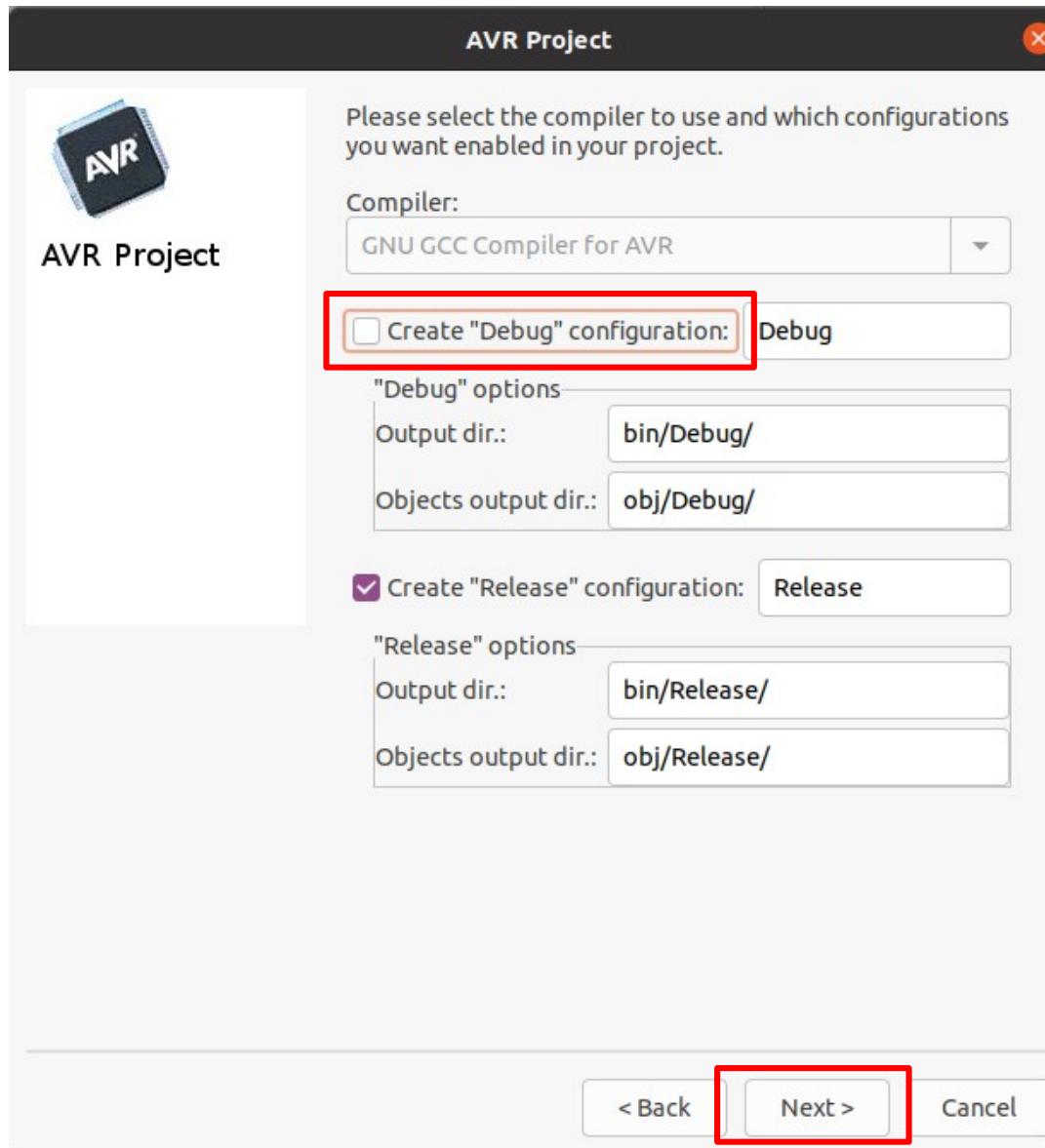
1. Ubuntu Linux 20.04에서 Arduino 기본 설정하기

Codeblock 새 프로젝트 생성 과정.



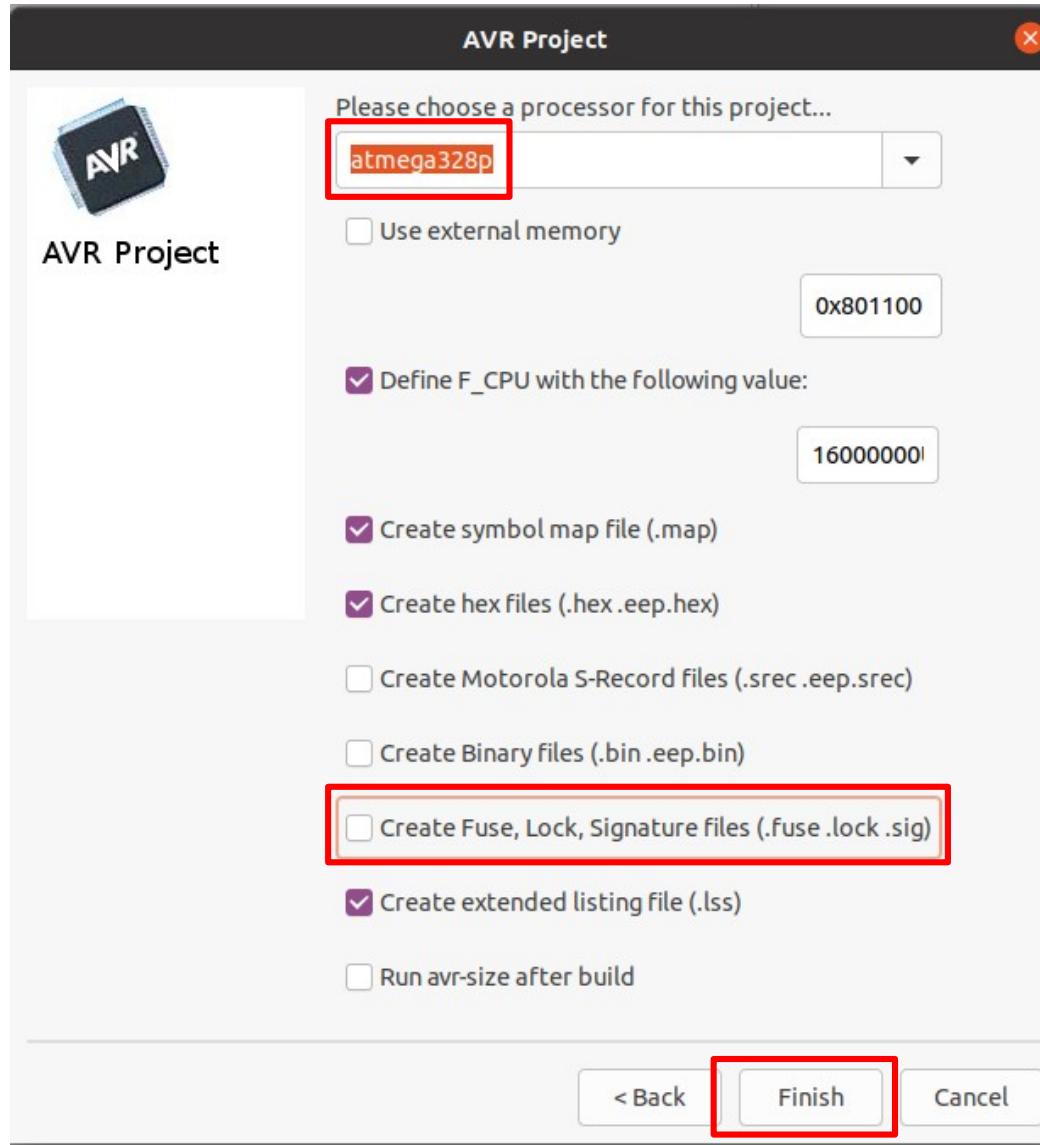
1. Ubuntu Linux 20.04에서 Arduino 기본 설정하기

Codeblock 새 프로젝트 생성 과정.



1. Ubuntu Linux 20.04에서 Arduino 기본 설정하기

Codeblock 새 프로젝트 생성 과정.



1. Ubuntu Linux 20.04에서 Arduino 기본 설정하기

예제 해보기(LED - blink)

```
main.c [blink] - Code
File Edit View Search Project Build Debug Tools Plugins Settings Help
main(void) : int
<global>
Management Projects
  Workspace
    button
      Sources
        blink
          Sources
main.c main.c
1 /*
2 */
3
4 #include <avr/io.h>
5
6 #define F_CPU 1600000L
7 #include <util/delay.h>
8
9 int main(void)
10 {
11     DDRB = 0x20;
12     // Insert code
13     while(1)
14     {
15         PORTB = 0x00;
16         _delay_ms(500);
17         PORTB = 0x20;
18         _delay_ms(500);
19     }
20
21     return 0;
22 }
```

Logs & others

Code::Blocks Search results Build log Build messages Deb

----- Build: Release in button (compiler: GNU GCC Compiler for AVR) -----
Target is up to date.
Nothing to be done (all items are up-to-date).

코드를 작성 후 톱니모양 아이콘
클릭하면 컴파일을 하고 코드에
에러가 있는지 알 수 있다.
(hex 파일 생성)

- 코드 간략 풀이
-> DDRB = 0x20;
0010 0000
PORTB5를 렉으로 설정.

PORTB = 0x00;
PORTB = 0x20;
-> PORTB5 출력 On/off
-> 사이에 delay 500ms

1. Ubuntu Linux 20.04에서 Arduino 기본 설정하기

예제 해보기(LED - blink) - 컴파일, hex파일

The screenshot shows the Code::Blocks IDE interface. On the left, the project tree shows a workspace named 'Workspace' with a single project 'blink' containing a source file 'main.c'. The main window displays the code for 'main.c' and a terminal window.

Code Editor Content (main.c):

```
1  /*
2   */
3
4  #include <avr/io.h>
5
6  #define F_CPU 1600000L
7  #include <util/delay.h>
8
9  int main(void)
10 {
11     DDRB = 0x20;
12     // Insert code
13     while(1)
14     {
15         PORTB = 0x00;
16         _delay_ms(500);
17         PORTB = 0x20;
18         _delay_ms(500);
19     }
20
21     return 0;
22 }
```

Terminal Output:

```
taein@taein-Lenovo-ideapad-700-15ISK: ~/proj/avr/blink/bin/Release
taein@taein-Lenovo-ideapad-700-15ISK: ~/proj/avr/blink$ cd bin
bash: cd: bin: No such file or directory
taein@taein-Lenovo-ideapad-700-15ISK: ~/proj/avr/blink$ cd bin
taein@taein-Lenovo-ideapad-700-15ISK: ~/proj/avr/blink/bin$ cd Release/
taein@taein-Lenovo-ideapad-700-15ISK: ~/proj/avr/blink/bin/Release$ ls
blink.eep  blink.elf  blink.hex  blink.lss  blink.map
taein@taein-Lenovo-ideapad-700-15ISK: ~/proj/avr/blink/bin/Release$ sudo avrdude -c avrisp2 -p m328p -U flash:w:blink.hex
[sudo] password for taein:
avrdude: AVR device initialized and ready to accept instructions

Reading | ##### | 100% 0.00s

avrdude: Device signature = 0x1e950f (probably m328p)
avrdude: NOTE: "flash" memory has been specified, an erase cycle will be performed
          To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file "blink.hex"
avrdude: input file blink.hex auto detected as Intel Hex
avrdude: writing flash (178 bytes):

Writing | ##### | 100% 0.08s

avrdude: 178 bytes of flash written
avrdude: verifying flash memory against blink.hex:
```

proj/avr/blink 프로젝트를 앞서 생성 했고

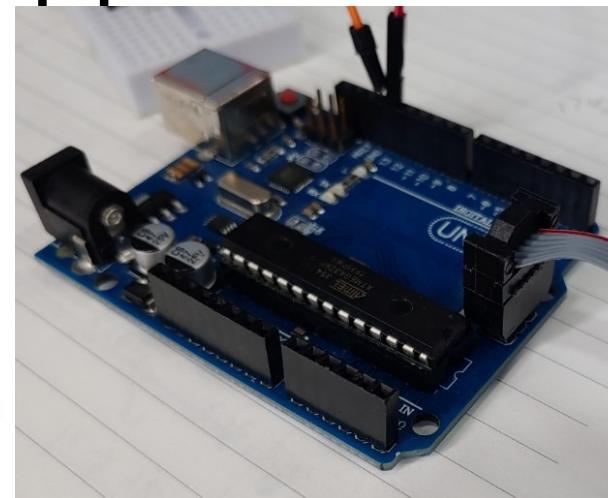
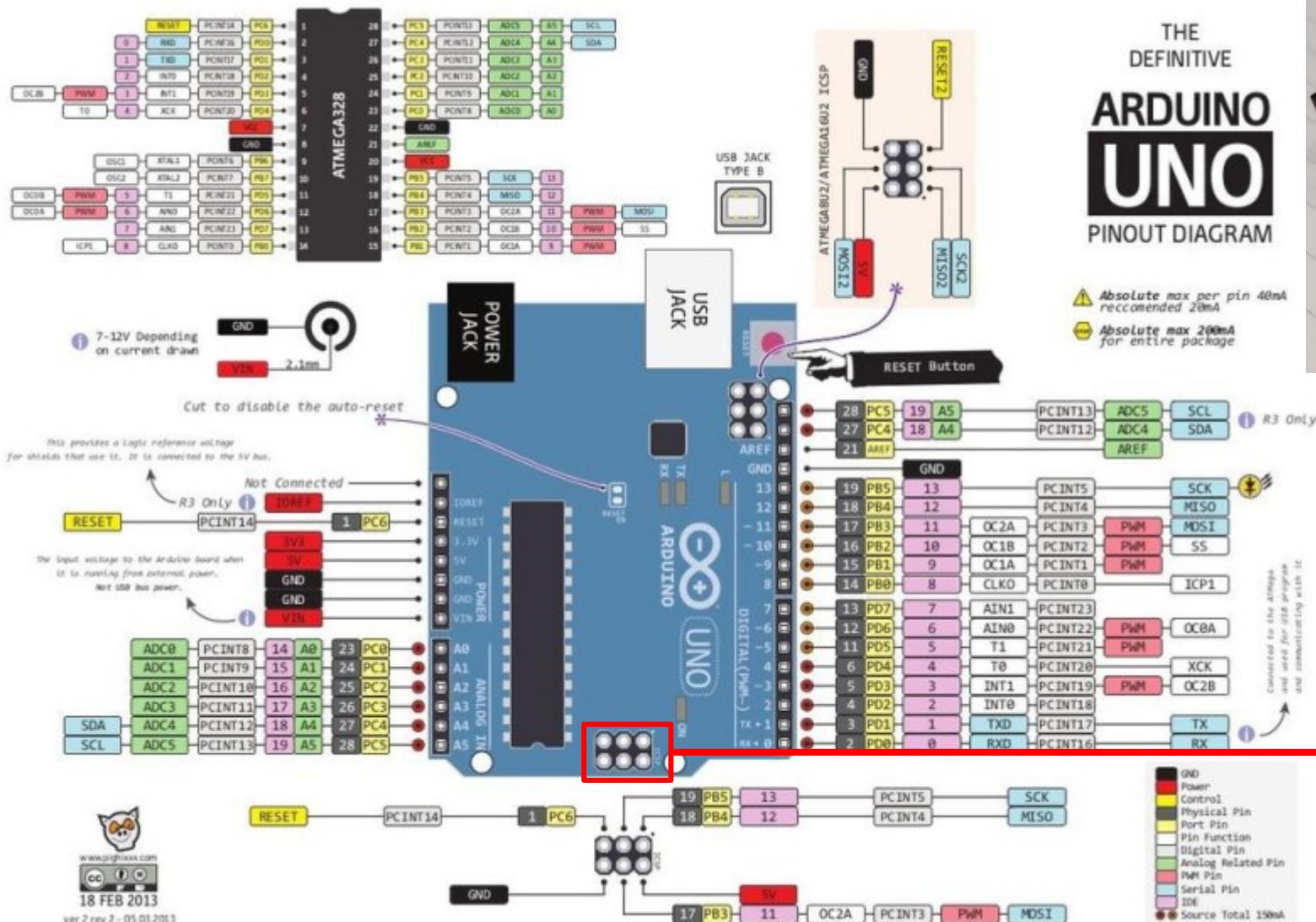
그 프로젝트를 컴파일 하면 bin/Release 안에 hex 파일이 생성 됩니다.(hex 파일을 아두이노에 입력하면 프로그램이 삽입 된다.)

Hex 파일 명령어

→ sudo avrdude -c avrisp2 -p m328p -U flash:w:blink.hex
isp 버전 IC종류 flash 파일

1. Ubuntu Linux 20.04에서 Arduino 기본 설정하기

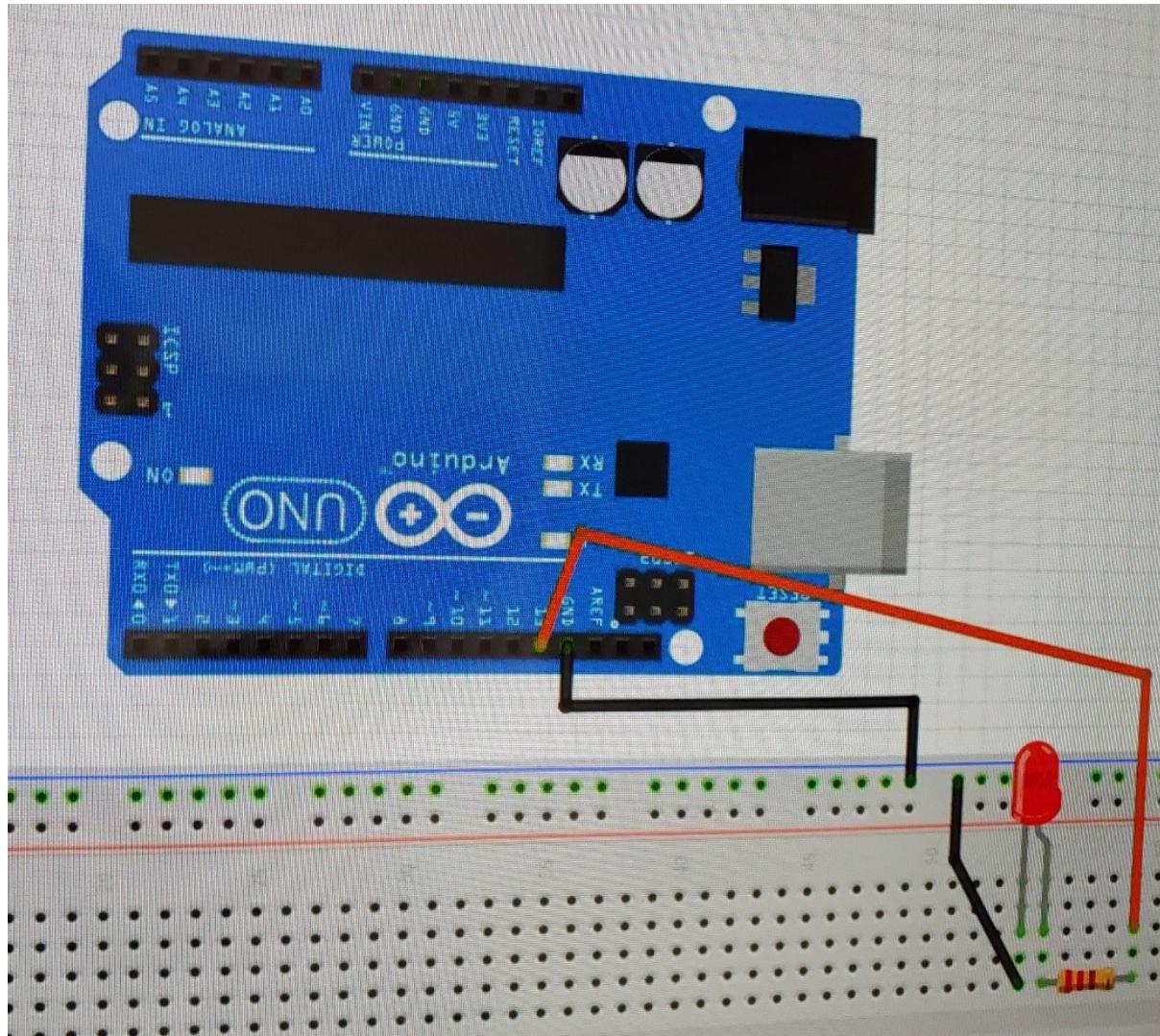
예제 해보기(LED - blink) - isp arduino와 연결 하기 ISP 연결 모습



ISP 핀 맵 참고 후 연결

1. Ubuntu Linux 20.04에서 Arduino 기본 설정하기

예제 해보기(LED - blink) - 회로 구성



ISP 핀 맵 참고 후 연결

-> LED의 +를

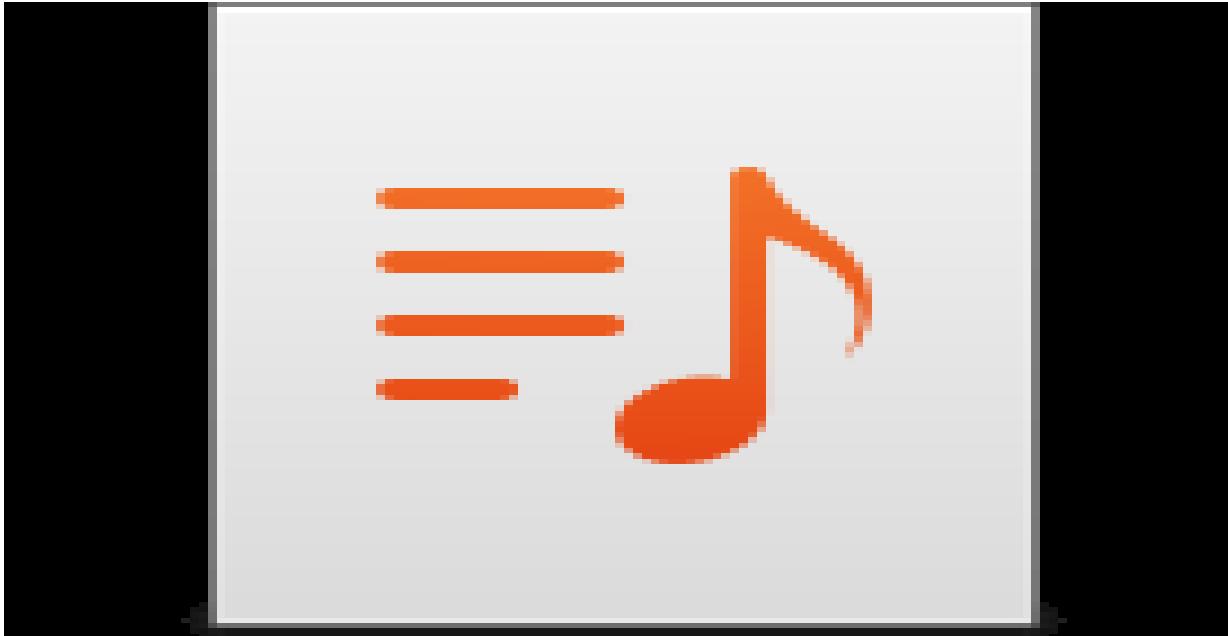
출력인 13번핀 (POORT5)에 연결

LED – 를 GND에 연결

LED는 330 옴 정도의 보호 저항 연결

1. Ubuntu Linux 20.04에서 Arduino 기본 설정하기

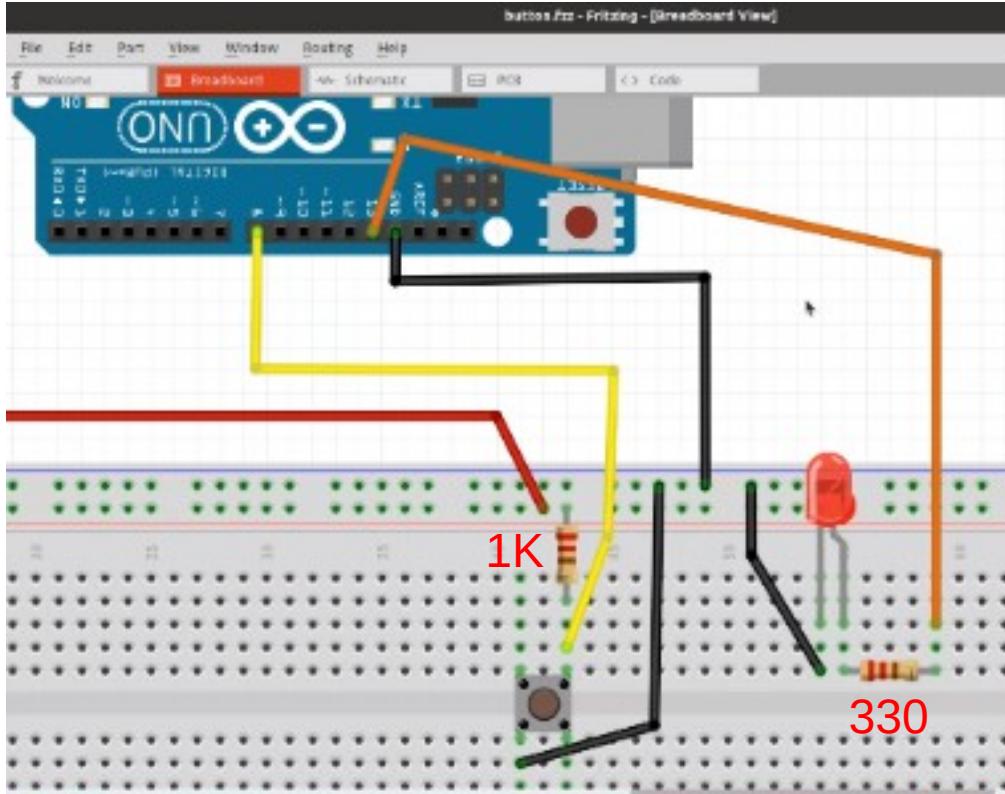
예제 해보기(LED - blink) - 작동 영상 및 오실로스코프 파형



500ms로 LED 점멸 시마다
파형이 구형파처럼 동작.

2. Ubuntu Linux 20.04에서 Arduino 기본 예제

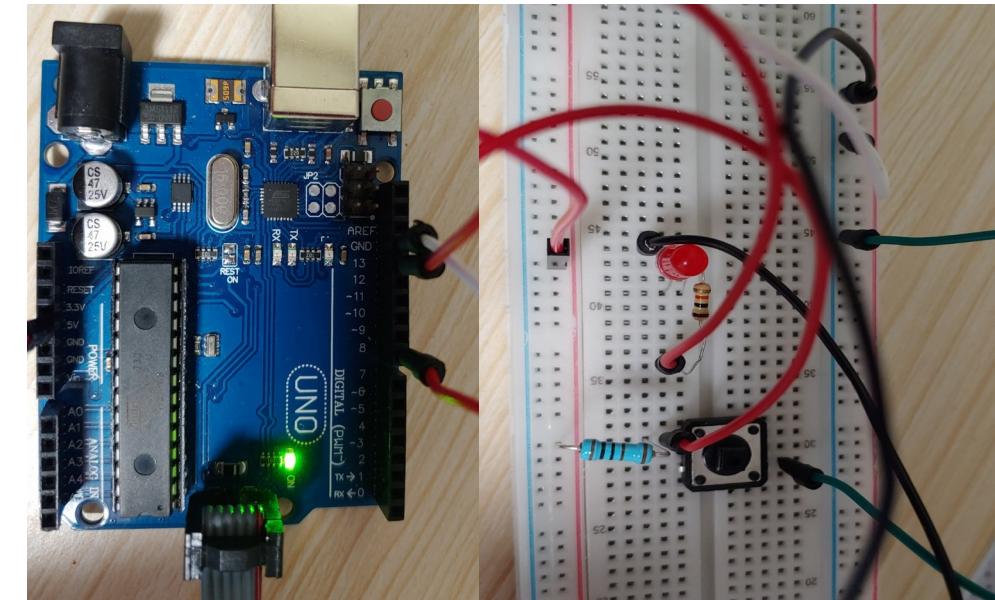
예제 해보기(button) - 회로 구성



앞의 blink 예제에서

입력을 받기 위한 button을 추가로 구성.

8번핀(PB0)을 입력 핀(버튼)으로 구성
13번핀(PB5)를 출력 핀(LED)으로 구성



연결 사진

2. Ubuntu Linux 20.04에서 Arduino 기본 예제

예제 해보기(button) - 코드 작성

The screenshot shows the Arduino IDE interface. The menu bar includes File, Edit, View, Search, Project, Build, Debug, Tools, Plugins, Settings, and Help. The toolbar has icons for file operations like Open, Save, and Print, along with build and run tools. The left sidebar shows the 'Management' section with 'Projects' selected, displaying 'button' and 'blink' projects, each with a 'Sources' folder containing a 'main.c' file. The main workspace shows the 'main.c' file content:

```
#define F_CPU 16000000L
#include <util/delay.h>

int main(void)
{
    // Insert code
    DDRB = 0x20;

    while(1)
    {
        if((PINB & 0x01) == 0x00)
        {
            PORTB = 0x20;
            _delay_ms(500);
        }
        else
        {
            PORTB = 0x00;
            _delay_ms(500);
        }
    }

    return 0;
}
```

주요 코드 설명

- DDRB = 0x20;
PB5 출력, 나머지 입력
- if((PINB & 0x01) == 0x00)
-> PB0에 입력이 low 일 때, if 문 진입.
- 동작 버튼을 눌렀을 때 LED ON
뗐을 때 LED OFF

2. Ubuntu Linux 20.04에서 Arduino 기본 예제

예제 해보기(button) - 동작 영상 및 파형



버튼을 눌렀을 때 LED ON 하고,
버튼을 뗐을 때 LED OFF 하는 동작 자체는

잘 되긴 하나

버튼을 동작 할 때 스위칭 노이즈가
심한 관계로 버튼 동작이 굼뜨는 경향이 있다.

이를 해결 하기 위한 S/W 적(인터럽트)
H/W 방법을 사용해 보도록 한다.



채터링 노이즈

2. Ubuntu Linux 20.04에서 Arduino 기본 예제

예제 해보기(button_interrupt) - 코드 작성

The screenshot shows the Code::Blocks IDE interface with the following details:

- File Menu:** FILE, EDIT, VIEW, SEARCH, PROJECT, BUILD, DEBUG, TOOLS, PLUGINS, SETTINGS, HELP.
- Toolbar:** Includes icons for file operations like Open, Save, Print, and various build and debug tools.
- Project Explorer (Projects View):** Shows the workspace with three projects: Workspace, button (with Sources and main.c), blink (with Sources), and button_interrupt (with Sources). The main.c file under button_interrupt is currently selected.
- Code Editor:** Displays the C code for the main.c file. The code handles a button interrupt on pin PB5, toggling an LED on pin PB4. It includes comments in Korean explaining the logic.
- Logs & others:** Shows the build status: "Build: Release in button (compiler: GNU GCC Compiler for AVR)". It indicates that the target is up-to-date and nothing needs to be done.

```
/*  
 *  
 */  
#include <avr/io.h>  
//인터럽트 활용은 이한 헤더  
#include <avr/interrupt.h>  
  
// 인터럽트란 무엇일까?  
// 가장 중요한 최우선 순위의 작업으로 하던 모든 작업을 중단하고 이것을 최우선적으로 처리해야 한다.  
  
// 인터럽트 케이스 : 물건  
// 부모님이 방문 열고 나옴 : 인터럽트 발생!  
// 걸리면 즉기 때문에 빨리 모니터를 끄로 자는 척을 한다.(부모님이 방에 돌아 가실 때 까지) : 인터럽트 핸들러의 동작  
// 부모님이 다시 돌아가시면 인터럽트 처리의 일련의 절차가 마무리 된다.  
  
int main(void)  
{  
    DDRB &= ~(1 << DDB5); // 해당 비트를 무조건 입력으로, PB5를 입력으로(스위치)  
    PORTB |= (1 << PORTB4); // 비트가 있던 없던 새로운 정보를 덮어씀, 입력 PB4를 출력으로(LED)  
  
    // PCIE0 활성화, 인터럽트 상태 변경에 따른 트리거 PCINT0  
    PCICR |= (1 << PCIE0);  
    PCMSK0 |= (1 << PCINT5);  
  
    //인터럽트 활성화  
    sei();  
  
    while(1) // 이것 때문에 계속 버튼 눌러도 안끝나고 쭉쭉 된다.  
;  
  
    return 0;  
}  
  
// 1111 0000  
// 0001 0000      0x10 xor  
//-----  
// 1110 0000  
// 0001 0000      xor  
//-----  
// 1111 0000      // 원래대로 돌아옴  
// 인터럽트 서비스 루틴  
ISR(PCINT0_vect)  
{  
    //toggle( 껌다.. 켰다)  
    PORTB ^= 0x10;  
}
```

2. Ubuntu Linux 20.04에서 Arduino 기본 예제

예제 해보기(button_interrupt) - 코드 작성

버튼의 채터링 영향을 덜하게 하기 위해 인터럽트 방법을 사용한다고 하는데.. 인터럽트가 뭘까?

- 정의 : 가장 중요한 최우선 순위의 작업이므로 하던 모든 작업을 중단하고 이것을 최우선적으로 처리해야 한다.
- **인터럽트 케이스** : 몰컴
 - > 부모님이 방문 열고 나옴 : **인터럽트 발생 !**
 - > 걸리면 죽기 때문에 빨리 모니터를 끄고 자는 척을 한다.(부모님이 방에 돌아가실 때 까지)
: **인터럽트 핸들러의 동작**
 - > 부모님이 다시 돌아가시면 **인터럽트 처리의 일련의 절차가 마무리** 된다.

2. Ubuntu Linux 20.04에서 Arduino 기본 예제

예제 해보기(button_interrupt) - 코드 해석

```
DDRB &= ~(1 << DDB5); // 해당 비트를 무조건 입력으로, PB5를 입력으로(스위치)  
// 해당 비트에 0의 값을 & 연산 해줌으로써 입력(0)  
PORTB |= (1 << PORTB4); // 비트가 있던 없던 새로운 정보를 덮어 씀, PB4를  출력으로(LED)  
// 해당 비트에 1의 값을 | 연산 해줌으로써 반대 값이 계속해서 출력.
```

```
// PCIE0 활성화, 인터럽트 상태 변경에 따른 트리거 PCINT0_vect  
PCICR |= (1 << PCIE0);  
PCMSK0 |= (1 << PCINT5);  
-> 인터럽트 레지스터는 좀 더 알아 보자.
```

PCICR – Pin Change Interrupt Control Register

Bit	7	6	5	4	3	2	1	0	PCICR
(0x68)	-	-	-	-	-	PCIE2	PCIE1	PCIE0	
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Bit 0 - PCIE0: Pin Change Interrupt Enable 0

When the PCIE0 bit is set (one) and the I-bit in the status register (SREG) is set (one), pin change interrupt 0 is enabled. Any change on any enabled PCINT7..0 pin will cause an interrupt. The corresponding interrupt of pin change interrupt request is executed from the PCI0 interrupt vector. PCINT7..0 pins are enabled individually by the PCMSK0 register.

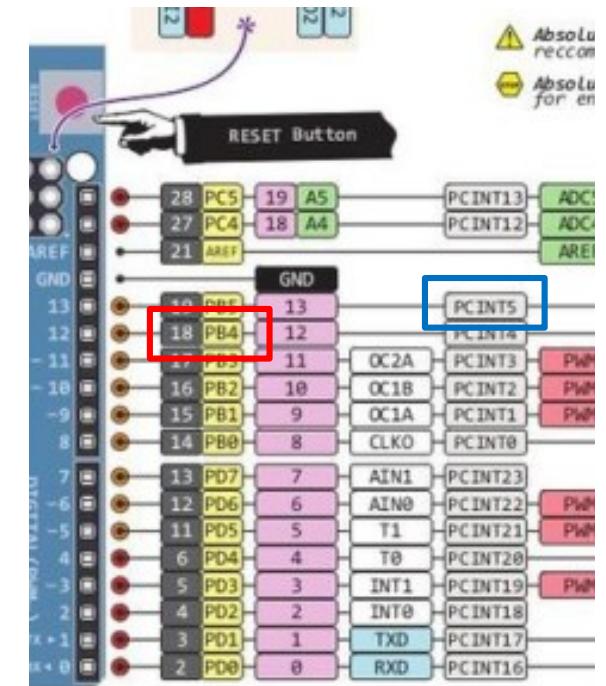
-> 위에서 보는 것 처럼 PCICR은 인터럽트 사용을 enable 하는 역할을 하며 PCIE0을 ON 하였기에 인터럽트0이 동작 된다.
그리고, PCINT7..0들을 활성화 할 수 있게 되는데 이것은 PCMSK0 레지스터에 의해 개별적으로 활성화 된다.

PCMSK0 – Pin Change Mask Register 0

Bit	7	6	5	4	3	2	1	0	PCMSK0
(0x6B)	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

- Bit 7..0 – PCINT7..0: Pin Change Enable Mask 7..0

Each PCINT7..0 bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT7..0 is set and the PCIE0 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin. If PCINT7..0 is cleared, pin change interrupt on the corresponding I/O pin is disabled.



2. Ubuntu Linux 20.04에서 Arduino 기본 예제

예제 해보기(button_interrupt) - 코드 해석

Sei(); // 인터럽트 동작 시작하는 명령어이다.

// 앞서 PCICR 레지스터 설명에서
아래 interrupt 서비스 루틴 확인.

```
ISR(PCINT0_vect)
{
    // toggle (껐다.. 켰다)
    PORTB ^= 0x10;
}
```

- Bit 0 - PCIE0: Pin Change Interrupt Enable 0

When the PCIE0 bit is set (one) and the I-bit in the status register (SREG) is set (one), pin change interrupt 0 is enabled. Any change on any enabled PCINT7..0 pin will cause an interrupt. The corresponding interrupt of pin change interrupt request is executed from the PCI0 interrupt vector. PCINT7..0 pins are enabled individually by the PCMSK0 register.

여기서 PORTB에 0x10을 XOR 계산하면 LED가 토플 되는 이유는 아래와 같다.

1111 0000
0001 0000 0x10 XOR

1110 0000
0001 0000 0x10 XOR

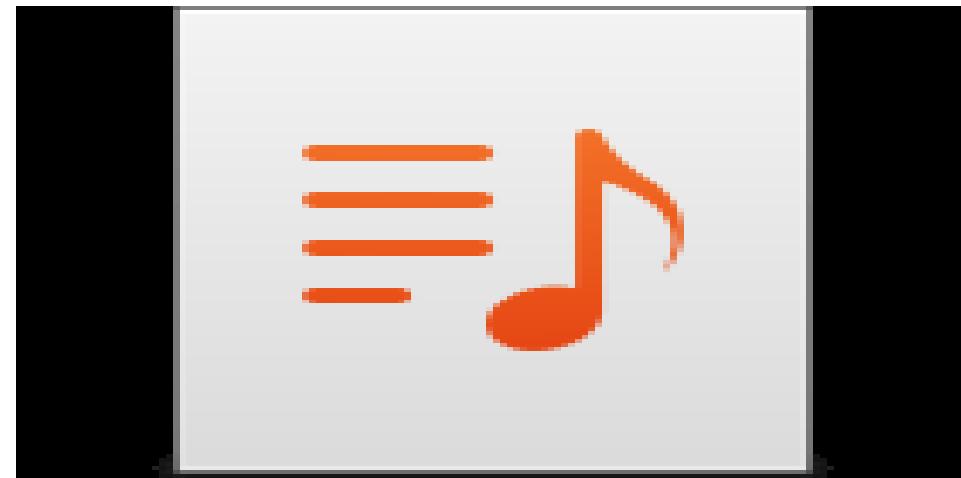
1111 0000 원래대로 돌아옴.

2. Ubuntu Linux 20.04에서 Arduino 기본 예제

예제 해보기(**button_interrupt**) - 동작 및 파형 영상



버튼 영상

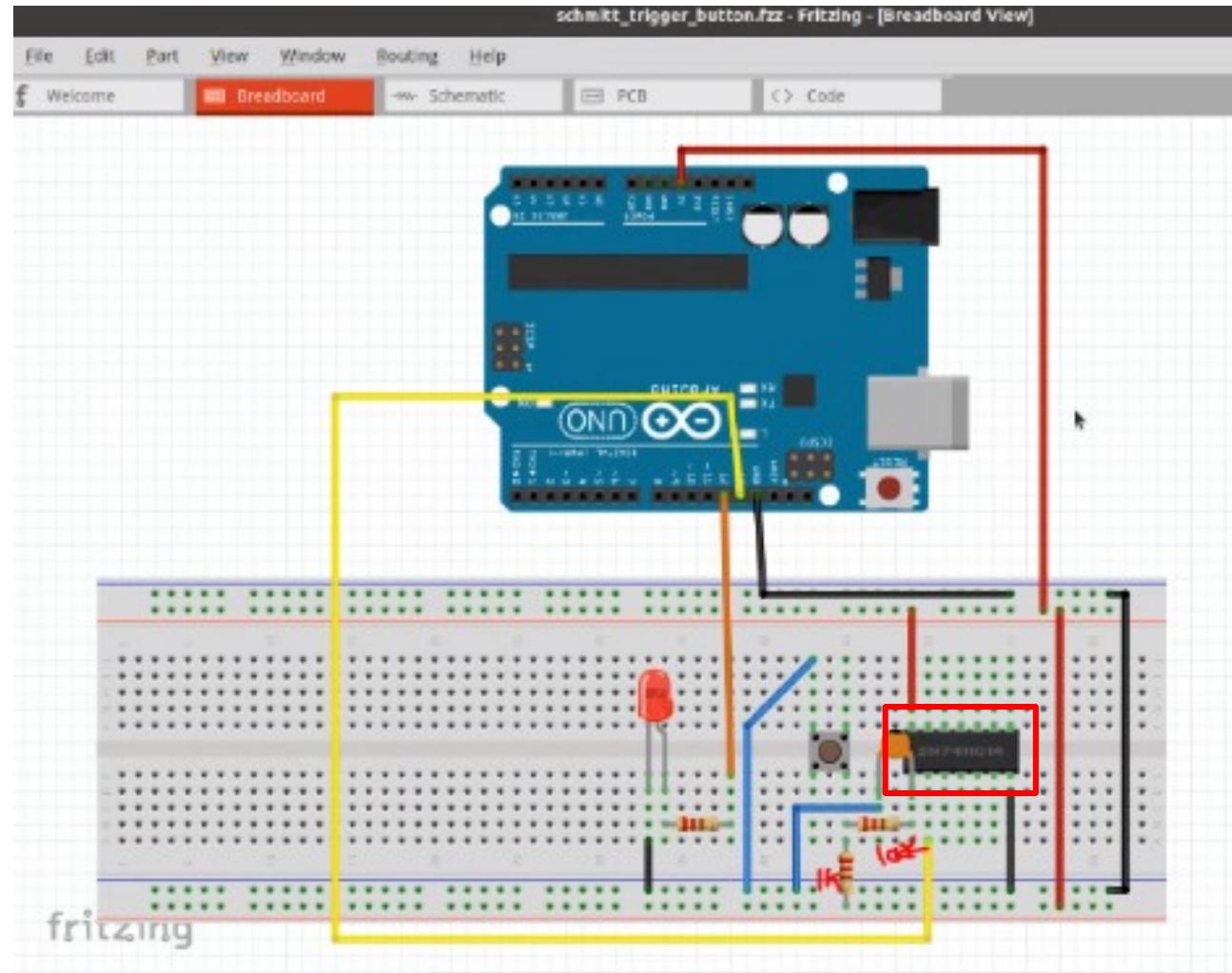


파형 영상

I/O에만 의지하지 않고 interrupt를 사용했을 때 반응속도가 더 빠른 듯 하나 보는 바와 같이 동작이 제멋대로임을 알 수 있다.

2. Ubuntu Linux 20.04에서 Arduino 기본 예제

예제 해보기(button_interrupt) - 74HC14 IC 적용(채터링 방지)



74HC14를 적용해서 H/W 적으로도 채터링을 제거 한다.

2. Ubuntu Linux 20.04에서 Arduino 기본 예제

예제 해보기(button_interrupt) - 74HC14 IC 적용(채터링 방지)

74HC14 IC 구조를 살펴 보자.

MM74HC14

Hex Inverting Schmitt Trigger

General Description

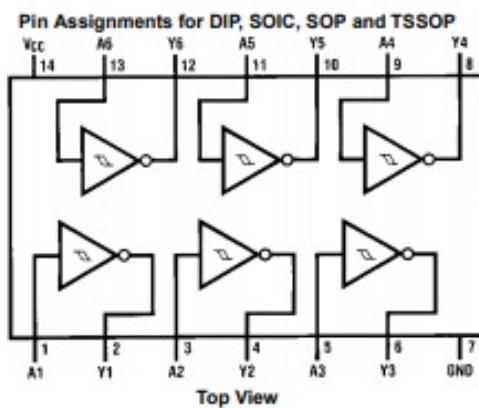
The MM74HC14 utilizes advanced silicon-gate CMOS technology to achieve the low power dissipation and high noise immunity of standard CMOS, as well as the capability to drive 10 LS-TTL loads.

The 74HC logic family is functionally and pinout compatible with the standard 74LS logic family. All inputs are protected from damage due to static discharge by internal diode clamps to V_{CC} and ground.

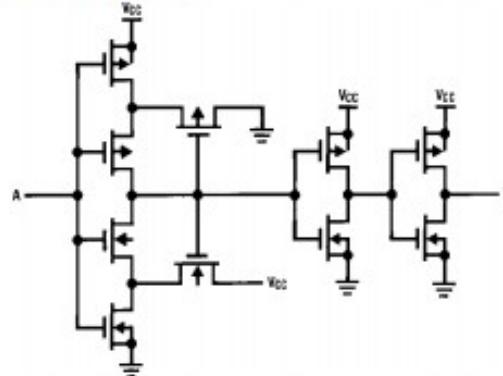
Features

- Typical propagation delay: 13 ns
- Wide power supply range: 2–6V
- Low quiescent current: 20 μ A maximum (74HC Series)
- Low input current: 1 μ A maximum
- Fanout of 10 LS-TTL loads
- Typical hysteresis voltage: 0.9V at $V_{CC} = 4.5V$

Connection Diagram



Logic Diagram



보자하니 슈미트트리거가 6 개가 구성이 되어 있는 IC 이다.

그럼 슈미트 트리거가 무엇인가?

2. Ubuntu Linux 20.04에서 Arduino 기본 예제

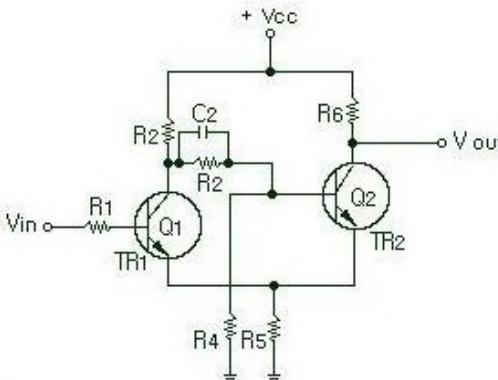
예제 해보기(button_interrupt) - 74HC14 IC 적용(채터링 방지)

슈미트 트리거(Schmitt trigger circuit)

2개의 논리 상태 중에서 어느 한 상태로 안정 되는 회로이다. (쌍안정 멀티 바이브레이터의 변형된 형태라 할 수 있다.)

이 회로는 입력 전압이 어떤 정해진 값 이상으로 높아지면 출력 파형이 상승하고

어떤 정해진 값 이하로 낮아지면 출력파형이 하강하는 동작을 한다.

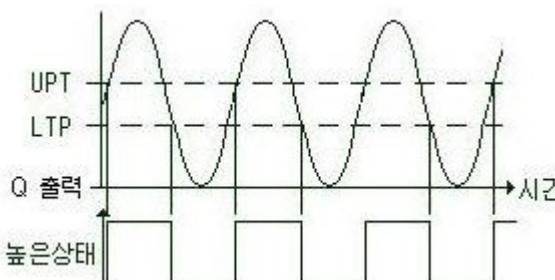


슈미트 트리거 내부회로

구형파가 아닌 입력 파형을 걸어 주면 아래 그림처럼 전환 레벨에 해당되는 펄스 폭의 직사각형 파를 얻을 수 있다.

이 회로의 트리거 신호는 서서히 변하는 전압(사인파)이다. **슈미트 트리거 회로는 입력 전압 값에 따라 민감하게 동작하는 회로로써 2개의 서로 다른 트리거 전압 값에서 출력 상태가 변환된다.**

즉, 아래 그림과 같이 낮은 트리거 전압 값(LTP : Low trigger point)과 높은 트리거 전압(UTP : upper trigger point)에서 동작한다. 그러므로, 입력파형은 서서히 변화되는 사인파이고 출력은 높고 낮은 2개의 논리 상태를 형성하는 구형파이다.



포기하면 얻는 건 아무것도 없다.

2. Ubuntu Linux 20.04에서 Arduino 기본 예제

예제 해보기(button_interrupt) - 74HC14 IC 적용(채터링 방지)

슈미트 트리거(Schmitt trigger circuit)

슈미트 트리거 회로는 히스테리시스 현상이 나타나며, 직접회로(IC)는 SN7414가 있다.

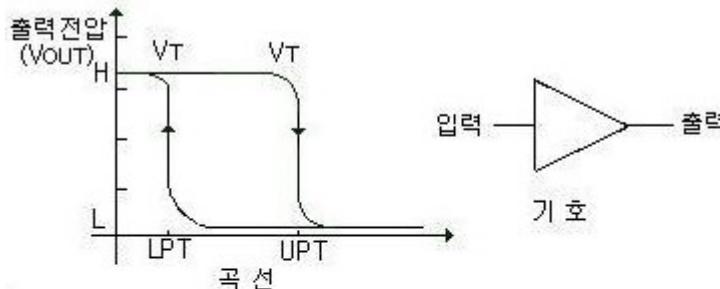
히스테리시스 전압이란 일정한 상태에서 일정한 값으로 정의되는 전압이 아니라 이전의 전압 상태 변화에 따라 값이 변하는 전압을 말합니다.

입력전압 V_i 에 대해 출력전압 V_o 가 결정될 때 입력전압값이 커지면서 결정되는 출력전압값과
입력전압 값이 작아지면서 결정되는 출력전압값이 다를 때 이러한 전압 특성을 히스테리시스 특성을 가진다고 한다.

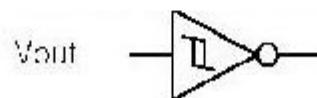
이러한 히스테리시스 특성은 주로 입력전압값에 대한 어떤 임계값에 대하여 출력전압이 high 혹은 low로 결정 될 때
임계값 근처의 입력 값에 대하여 출력전압값이 흔들리는 것을 막을 수 있다.

이 때, 히스테리시스 특성을 이용하면 어떤 값 이상에서 high 가 되고 나면, 특정 값 이하로 떨어지기 전에는 high 값이 유지 된다.
반대로 어떤 값 이하에서 low가 되고 나면, 특정 값 이상으로 올라가기 전에는 low 값을 유지하게 됩니다.
즉, 임계값 근처의 작은 변화에 의해 출력전압 값이 변하는 것을 막고자 할 때 이러한 특성을 이용 한다.

슈미트 트리거 회로는 일반적으로 NOT회로와 같이 출력은 입력과 반대이고, 단안정 멀티바이브레이터와 같이 동작하도록 변경 가능 하다.



일반적인 NOT(7404)



슈미트 트리거(7414)의 기호의 차이

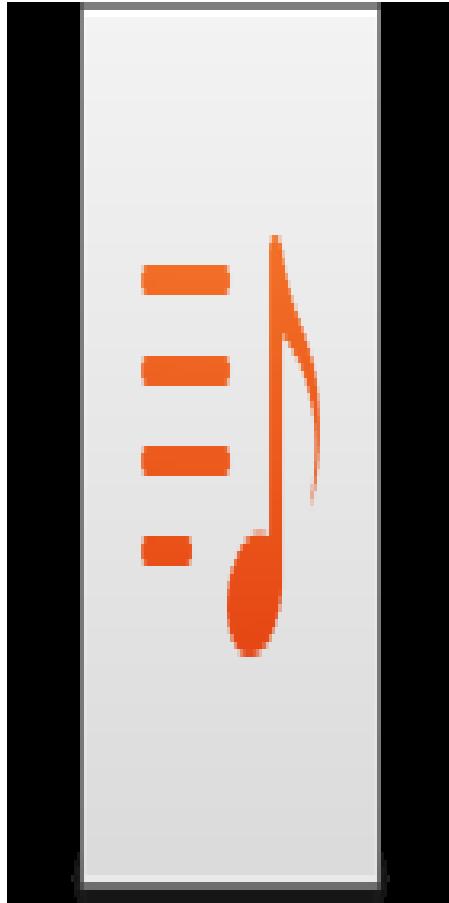
슈미트 트리거는 1에서 0으로 또는 0에서 1로 신호가 변할 때,
잡음에 의해서 1인지 0인지를 판별 할 수 없을 때를 대비하여
추가하는 회로이다. 즉, 변하는 시점이 1~0 사이에 두개가 존재한다.

만약 5V 회로가 있다면, 0 ~ 0.8V는 0으로 인식하고,
2.5V ~ 5V는 1로 인식을 한다.

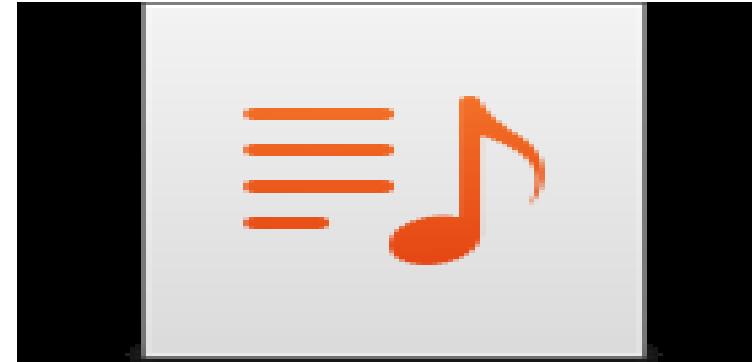
따라서, 중간에 남는 부분(0.8V ~ 2.5V)은 두 개의 값이 존재하게 되는데,
1에서 0으로 변할 때는 0.8V에서 0으로 인식.
0에서 1로 변할 때는 2.5 V에서 1로 인식하는 것이다.

2. Ubuntu Linux 20.04에서 Arduino 기본 예제

예제 해보기(button_interrupt) - 74HC14 IC 적용(영상 및 파형)



동작 영상



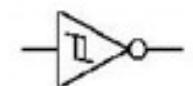
동작 파형



깔끔한 채터링

슈미트 트리거 까지 적용하여
파형이 굉장히 깔끔하고 동작도 깔끔하게
되는 것을 확인 할 수 있으며,

슈미트 트리거 IC에 입력이 GND로 연결 되어
있고 출력이 반대로 나오므로 영상과 같은 동작
되는 것을 볼 수 있다.



포기하면 얻는 건 아무것도 없다.

3. ARM 파이프라인 조사

파이프 라인 법칙??

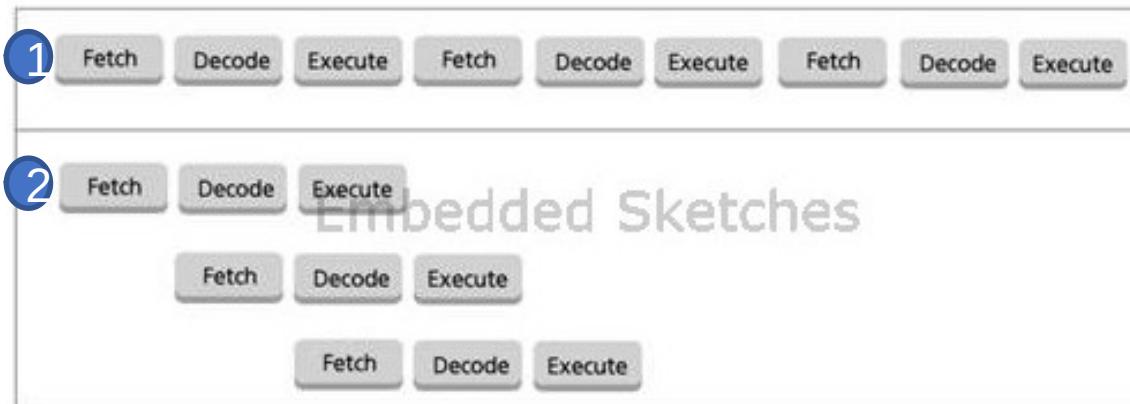
사람과 사람 관계가 깨지면 고달파 진다. 이러한 관계가 깨지지 않으려면 서로서로 이해하며 살아야 한다.
그런데 ARM core는 대화를 할 수 없으니 개발자가 알아서 관계가 깨지지 않도록 해주어야 한다.

무슨 말인가 하니.. ARM core는 메모리에 있는 코드를 읽어오고, 해석하고, 실행 합니다.
이를 **Fetch – Decode – Execute** 라고 표현한다.

개발자가 C 언어로 개발한 프로그램은 어디에 넣게 되죠? Flash 메모리죠.
Flash 메모리에 있는 데이터를 ARM core가 처리하기 위해서는 메모리에 접근해서 읽어 와야 겠죠.
그래서 Fetch 단계가 필요하죠.
메모리 접근해서 읽어온 데이터는 ARM core가 이해하는 언어인 기계어로 번역을 해야 겠죠.
요것이 바로 Decode 단계입니다.
ARM core가 이해하는 기계어로 번역 후 실행을 해 줘야 겠죠.
요기서 Execute 단계가 있는 것입니다.

자 여기서, 3단계가 각 단계별로 CPU clock을 한 cycle을 소모한다고 가정해 본다.
그렇다면 3개의 어셈블리 명령어가 수행한다고 하면 9번의 cycle 이 필요 하겠죠? ①

② 여기에서 각 단계별로 사용되는 자원이 다르니 중첩해서 사용해 보자고 생각한 것이 바로 파이프 라인(Pipe Line)입니다!!
아래 그림 처럼 파이프라인을 구성하여 3개의 명령어를 처리 하는데 5 cycle로 동일한 결과를 얻을 수 있게 되는 것이다.
그래서 성능이 훨씬 좋아지게 된다.

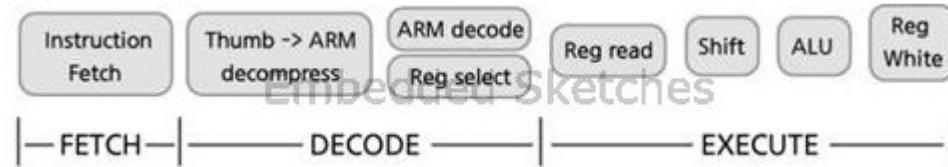


3. ARM 파이프라인 조사

가장 기본이 되는 3단계가 있고, 세부적으로 나누어 보면 ARM7 3단계, AMR9 5단계, ARM11 8단계, CortexA8은 13단계입니다.

3단계 부터 차근히 알아 보자.

3 단계 (Fetch - Decode – Execute)

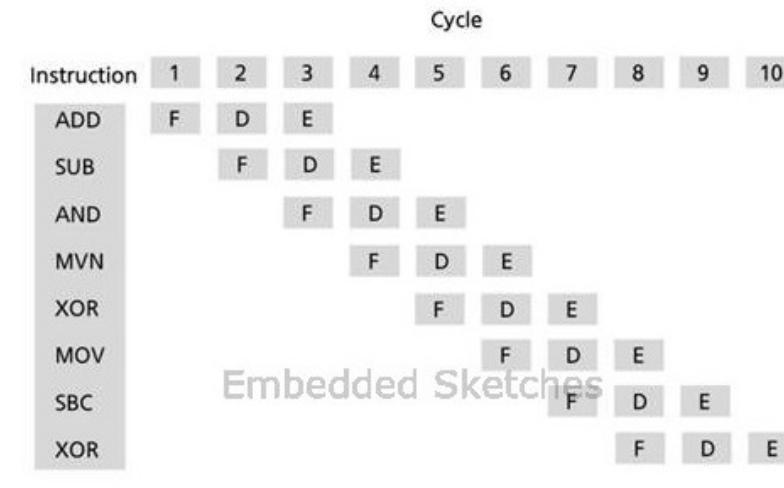


파이프라인 동작 과정을 분석해 보면, 전체 10 cycle 중 명령어가 실행(Execute)된 것은 8번 이다.

3번째 cycle에서 실행을 시작하게 된다.

하드웨어 디버거를 통해 살펴보면 메모리에 있는 코드를 읽어 와서 ①. 디코딩 ② 작업이 끝나면 ARM core의 기계어를 볼 수가 있다.
기계어로 된 ARM 명령어를 실행하면 ARM core가 동작하게 된다.

32bit ARM 명령어라면 주소 증가는 4byte 씩 된다.



A table showing detailed information for each instruction. It includes fields for address/line, code, label, and mnemonic. The table is divided into two sections: ① (fetch and decode) and ② (execute).

addr/line	code	label	mnemonic
SR:00000524	E1A01000		mov r1,r0
SR:00000528	E2812001		add r2,r1,#0x1
SR:0000052C	E1A02802		mov r2,r2,lsl #0
SR:00000530	E1A01842		mov r1,r2,asr #0
SR:00000534	E3A00000		mov r0,#0x0
SR:00000538	E3500034		cmp r0,#0x34

포기하면 얻는 건 아무것도 없다.

3. ARM 파이프라인 조사

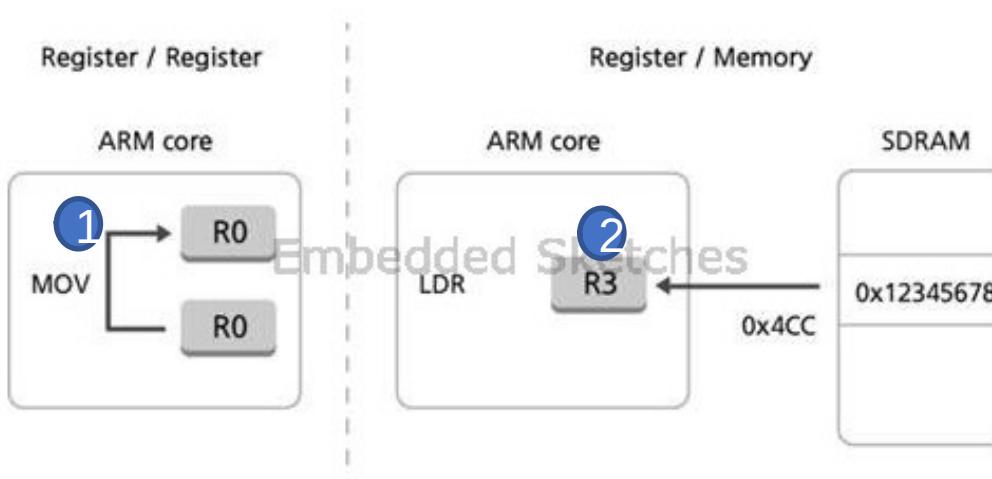
ARM 명령어 중에 LDR 명령어가 사용되면 파이프 라인의 동작단계는 조금 달라지게 된다.

LDR 명령어는 메모리에 접근하는 명령어이다.

ARM core는 레지스터/레지스터 명령어와 레지스터/메모리 명령어로 크게 두 가지 명령어군으로 구성된다.

	레지스터/레지스터 명령어	레지스터/메모리 명령어
명령어	mov r1,r0 add r2,r1,#0x1 sub r0,r1,#0 x1 ...	ldr r3,0x4CC str r3,0x4CC ...
비고	mov 명령어가 실행될 때는 단지 R1, R0 레지스터만 있으면 된다. add 명령어가 실행될 때는 단지 R2, R1만 레지스터만 있으면 된다.	ldr 명령어가 실행될 때는 R3 레지스터와 0x4CC인 메모리의 주소가 필요합니다.

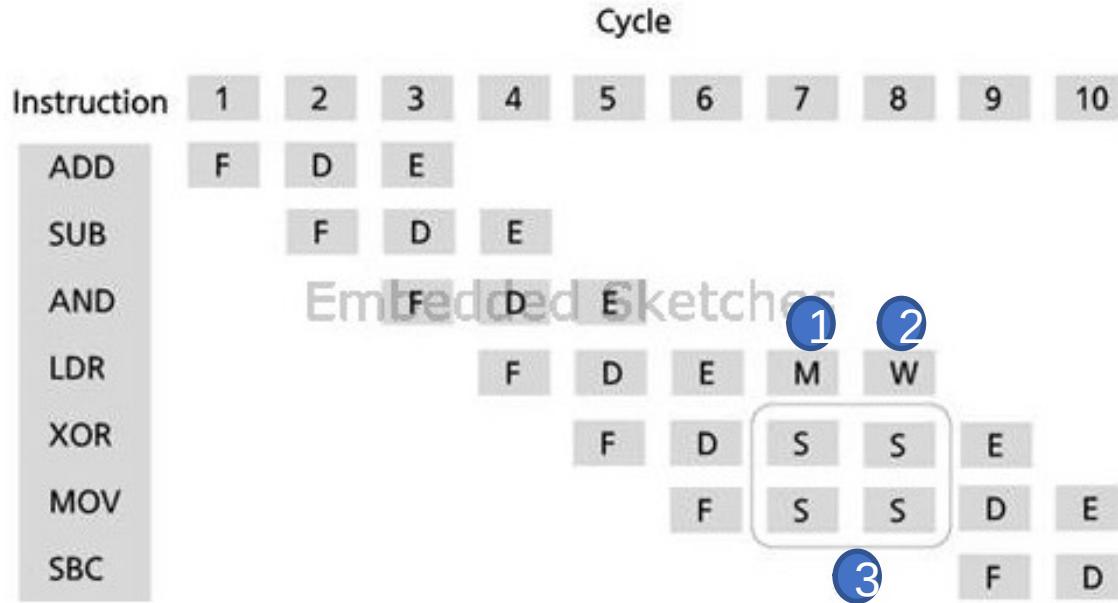
레지스터와 레지스터 명령어 ① 는 ARM Core 내에서 동작하기 때문에 실행속도가 매우 빠른 반면에 레지스터와 메모리 ② 간의 명령어는 외부 버스를 통해 접근하기 때문에 상대적으로 속도가 느리고, 더불어 파이프라인도 추가가 됩니다.



3. ARM 파이프라인 조사

ARM7 파이프라인 3단계 같은 경우에는 메모리에서 데이터를 읽는 Memory stage ① 와 레지스터에 쓰기를 하는 Write stage ② 가 추가가 됩니다.

하지만 ARM7의 경우 3단계 밖에 없기 때문에 이러한 단계에서는 모두 Stall ③ 됩니다.

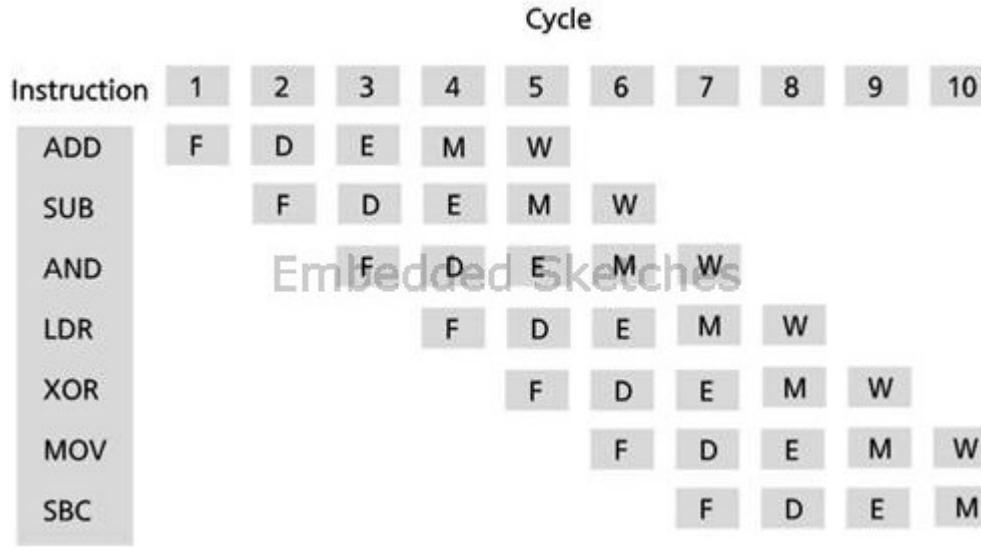


이렇게 된다면, 총 10 cycle 동안 명령어 실행(Execute)은 6번만 하게 됩니다.

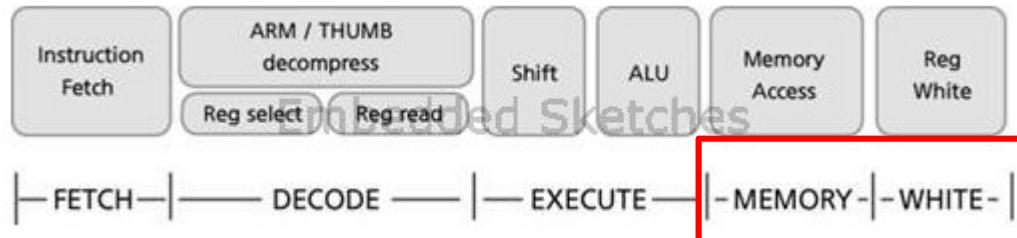
3. ARM 파이프라인 조사

ARM9 Core의 파이프라인은 5단계입니다.
5단계 파이프라인에서는 Memory와 Write 단계가 추가가 된다.

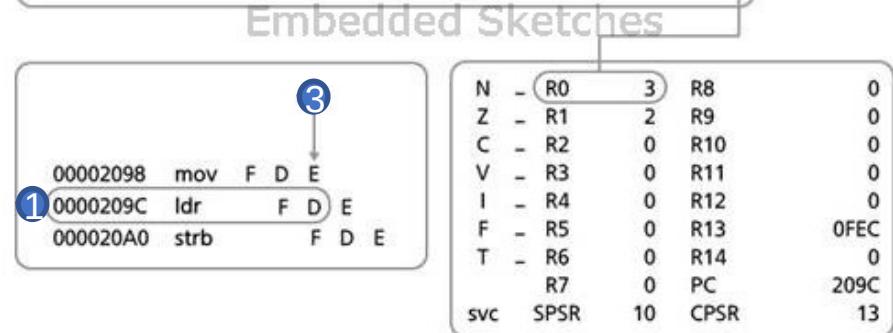
3단계의 LDR 명령어랑 비교해 보자.



Stall이 발생하지 않고 동작하는 하는 것을 알 수 있다.



addr/line	code	label	mnemonic
SR:00002090	E59F01F4		ldr r0,0x228C
SR:00002094	E5C0100C		strb r1,[r0,#0x0C]
592		vtripplearray [0][1][0] = 3;	
① SR:00002098	E3A00003		mov r0,#0x3
SR:0000209C	E59F11E8		ldr r1,0x228C
SR:000020A0	E5C10004		strb r0,[r1,#0x4]
593		vtripplearray [0][0][1] = 4;	
SR:000020A4	E3A01004		mov r1,#0x4



Q) 참고로 파이프라인 공부하는 사람들이 하드웨어 디버거를 이용해서 파이프라인을 분석 할 때 많이 헷갈린다고 하는데 이유가 무엇 일까?

하드웨어 디버거를 이용하여 0x209C ① 주소에서 Breakpoint 걸고 나서 Core를 Running 하면 0x209C에서 멈추게 됩니다.

우선 PC의 이전 주소인 0x2098 주소를 보면, 0x2098 ② 주소에 MOV 명령어가 있고, 이 명령어는 R0 레지스터에 0x3 값을 넣는 것인데, 현재 레지스터를 보니 이미 0x3이라는 값이 들어가 있다. 그렇다면 0x2098은 Execute 단계라는 결론과 함께 0x209C는 Decode ③ 단계라는 이야기가 됩니다. 그럴까요? 아닙니다.

왜냐면? 하드웨어 디버거는 여러 종류의 디버거가 있는데, 임베디드 시스템에서 가장 많이 사용하는 하드웨어 디버거가 JTAG 기반 툴입니다.

이 툴들은 Real-time성이 깨진다는 특징을 가지고 있습니다. 무슨 말인고 하니.. JTAG 기반의 툴들을 이용했을 때 실제 파이프라인을 고려해서 디버깅 해주는 것이 아니라 파이프라인을 무시하고 디버깅 합니다. 결론은 이론적으로 0x209C가 Decode가 되어야 하지만 파이프라인이 무시되고 Fetch-Decompress-Execute가 한꺼번에 실행 됩니다.

0x209C주소에 멈추어 있을 때 0x2098 주소에 있는 코드는 Execute가 끝난 상태이고, 0x209C는 다시 Fetch-Decompress-Execute 단계를 한꺼번에 실행할 준비를 하고 있는 상태입니다.

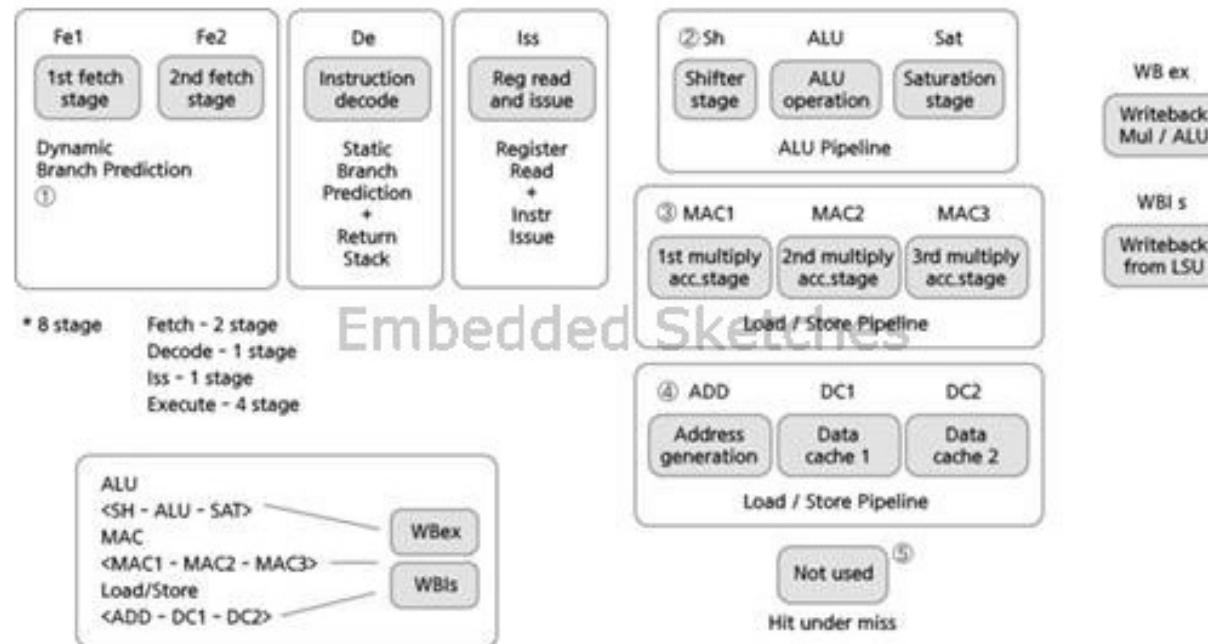
포기하면 얻는 건 아무것도 없다.

3. ARM 파이프라인 조사

ARM11 Core

이제부터는 파이프 라인이 아주 복잡해 지는데, 파이프라인은

Fetch stage 2단, Decode Stage, Issue Stage, Execute Stage 4단으로 총 8단계로 구성되어 있습니다.



Execute 단계는 4단계로 되어 있는데, 각 명령어 처리 unit 이 분리 되어 있습니다.

산술/논리 연산 처리 ALU pipeline 단계에서 처리하며, **SH-ALU-SAT 단계에서** 담당합니다.

곱셈 연산 처리는 MAC pipeline 단계이며, **MAC1-MAC2-MAC3 단계에서 담당** 합니다.

Load 명령과 Store 명령 처리 전용은 Load/Store pipeline 단계에서 처리하며, **ADD-DC1-DC2에서 담당**합니다.

ARM11은 pipeline 효율을 높이기 위한 Program Flow Prediction 기능이 있습니다.

명령의 흐름을 예측하여 pipeline의 stall 현상을 줄여 효율성을 높여 주는 역할입니다.

Dynamic branch prediction, Static branch prediction, Branch folding, Return stack 들이 그 역할들을 합니다.

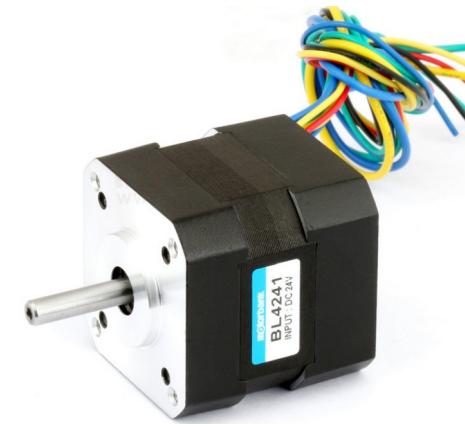
4. 프로젝트 제안서

주제 : BLDC 모터 제어기 제작

- 모터는 12 or 24 V BLDC 모터(Hall Sensor Type), Core는 Atmega128

- 기능 구현 :

1. 모터에 맞는 주파수로 모터 제어 하여 부드럽게 제어
2. 드티비를 이용한 모터 속도 제어
3. ADC를 통한 사용 전류 측정 및 연결 상태 확인
4. 역방향, 정방향 모터 회전
5. 모터 방향 전환 시 부드럽게
6. 동작 방향, 사용 전류, 드티 비, RPM 등의 정보를 LCD에 모니터링(캐릭터 LCD)
7. PI 제어 방법을 사용해 외부 환경 변화시에도 목표 RPM을 설정하면 모터가 알아서 찾도록 설계



- 제작 절차 :



- 예상 일정 :

단계	1주	2주	3주	4주	5주	6주	7주	8주
자료 조사 및 부품구매								
기본 기능 테스트								
회로 설계								
아트웍								
회로 납땜								
펌웨어								

Q. 질문

- 파이프라인 5단계 이상부터는 자료도 잘 없고, 이해하기가 어려운 부분이 있는 것 같습니다..
파이프라인 9단계 혹은 5단계 이상은 수업시간에 설명 해 주시면 감사드리겠습니다!!