AVR

임베디드스쿨2기

Lv1과정

2021. 06. 18

박태인

Servo Motor Control

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

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

#include <avr/interrupt.h>

// 특정 포트의 비트 설정 매크로
#define sbi(PORTx, BITx)        (PORTx |= (1 << BITx))  // set bit
// 특정 포트의 비트 클리어 매크로
#define cbi(PORTx, BITx)        (PORTx &= ~(1 << BITx)) // clear bit

void fast_pwm_mode_for_servo(void)
{
    cbi(SREG, 7);

    // non-inverted PWM
    sbi(TCCR1A, COM1A1);
    // PRESCALLER = 8, MODE 14 (FAST PWM), 16 MHz / 8 = 2 MHz, 2000 Khz -> 2000000 / 50 = 40000
    sbi(TCCR1A, COM1B1);
    sbi(TCCR1A, WGM11);
```

Set bit #define sbi

Clear bit #define cbi

SREG 상태 레지스터의 7 global interrupt clear

Servo_Motor_Control(pwm_mode_servo_init)

```
15  void fast_pwm_mode_for_servo(void)
16  {
17      cbi(SREG, 7);
18
19      // non-inverted PWM
20      sbi(TCCR1A, COM1A1);
21      // PRESCALLER = 8, MODE 14 (FAST PWM), 16 MHz / 8 = 2 MHz, 2000 Khz -> 2000000 / 50 = 40000
22      sbi(TCCR1A, COM1B1);
23      sbi(TCCR1A, WGM11);
        sbi(TCCR1B, WGM13);
        sbi(TCCR1B, WGM12);
        sbi(TCCR1B, CS11);

        // 20 ms -> 50 Hz
        // RC 모터의 특징이 50 Hz를 필요로 한다.
        ICR1 = 39999;

        DDRB = 0x02;

        sbi(SREG, 7);
    }
```

TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

2000000 hz를 50hz화 하면 40000 번

Table 15-2. Compare Output Mode, non-PWM

COM1A1/COM1B1	COM1A0/COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	Toggle OC1A/OC1B on compare match.
1	0	Clear OC1A/OC1B on compare match (set output to low level).
1	1	Set OC1A/OC1B on compare match (set output to high level).

Table 15-5. Waveform Generation Mode Bit Description(1)

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, phase correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, phase correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, phase correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, phase and frequency correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, phase and frequency correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, phase correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, phase correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Table 15-6. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (no prescaling)
0	1	0	$clk_{I/O}$/8 (from prescaler)
0	1	1	$clk_{I/O}$/64 (from prescaler)

Servo_Motor_Control(main)

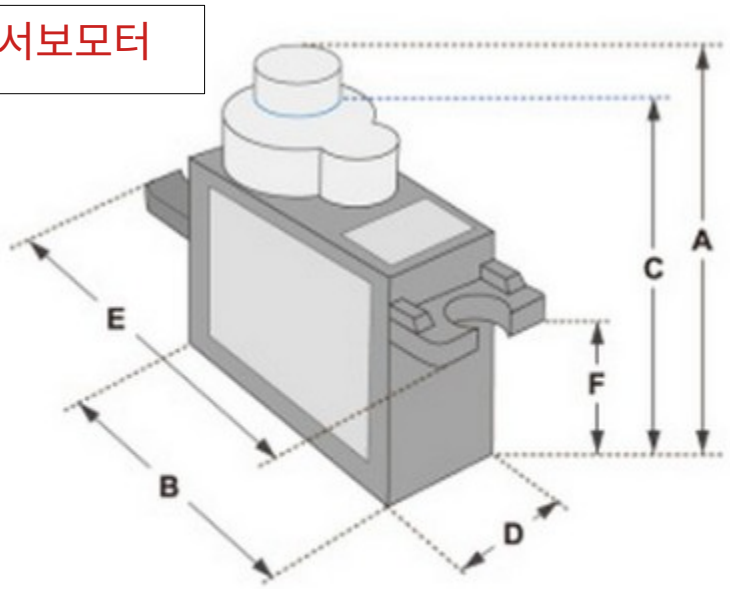
```c
int main(void)
{
    fast_pwm_mode_for_servo();

    // Insert code

    while(1)
    {
        // -90도
        //OCR1A = 2000;          // 1 ms
        OCR1A = 1000;          // 마이너스 히스테리시스 적용해서 -1000 해본 것.
        _delay_ms(1000);

        // 0 도
        OCR1A = 3000;
        _delay_ms(1000);

        // 90 도
        //OCR1A = 4000;
        OCR1A = 5000;     // 플러스 히스테리시스 적용해서 +1000 해본 것.
        _delay_ms(1000);

        // 0 도
        OCR1A = 3000;
        _delay_ms(1000);
    }
    ;

    return 0;
}
```

Servo_motor_pwm_init

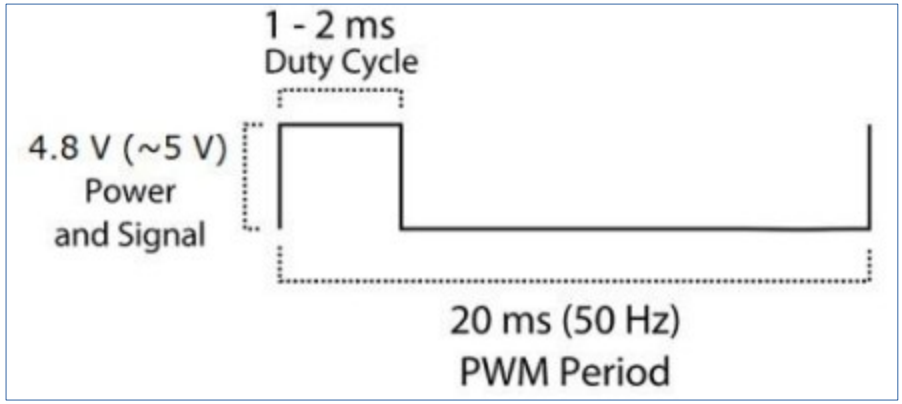
ㄴ 히스테리시스 현상 간단 설명(다음페이지)

50hz는 한 주기당 20ms
40000 이 20ms
- 1ms는 OCR1A가 2000
- 1.5ms는 OCR1A 3000
- 2ms는 OCR1A 4000

SG90 서보모터



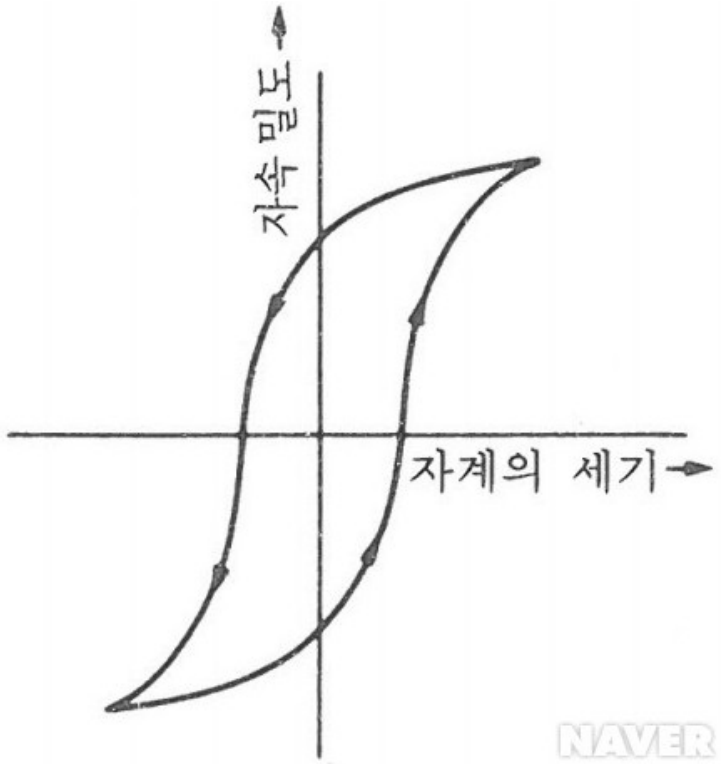
Position "0" (1.5 ms pulse) is middle, "90" (~2ms pulse) is middle, is all the way to the right, "-90" (~1ms pulse) is all the way to the left.



히스테리시스 Hysteresis : 이력현상은

어떤 물리량이 그 때의 물리조건만으로는 일의적으로 결정되지 않고, 그 이전에 그 물질이 경과해 온 상태의 변화과정에 의존하는 현상입니다. 이 히스테리시스 현상이 뚜렷하게 일어나는 것으로는 강자성체의 자기이력현상과 탄성체의 탄성이력현상이 있습니다. 이 때의 상태변화와 상태변화를 일으키는 물리량 사이의 그래프는 히스테리시스 곡선을 이룹니다.

이를 쉽게 이야기 하면
자성을 가진 금속에 자석을 가져대면 자서을 띠게 되고 자석을 떼어내면 자성이 사라지는데 이를 몇회 반복하면 자석을 금속에서 떼어낸 후에도 자성을 가지게 되는 것 즉 자성에 대한 이력이 남게 되는 것을 히스테리시스 현상이라고 합니다.



자동차 히스테리시스 Hysteresis : 이력현상은

차량의 히스테리시스현상은 주행 속도와 스로틀 밸브의 개도에 따라 정해지는 변속점의 부근에서 빈번한 변속이 일어나 주행충격이 발생하거나 운전감이 떨어 지게 되는데 이를 방지하기 위해 변속점에 있어 스로틀밸브의 개도와 차량의 주행 상태에 따라 변속점의 시프트 업이 시프트 다운보다 높게 설정되도록하여 승차감을 향상 시키고 이를 히스테리시스 (Hysteresis : 이력현상) 이라고 합니다.

알기쉽게 예를 들어 고속도로의 약간의 오르막상태로 가정하고 주행을 할때 4속으로 가다가 킥다운되어 3속으로 떨어지고 금방 또 4속으로 .... 이런경험들 다 있을것입니다.. 어정쩡한 조건이 되어지면 기어의 단수가 올라갔다가 내려 갔다를 반복해서 생기는 구간을 히스테리시스 구간이라하고 가능한 속도를 높이던지 아니면 홀드 버턴을 눌러서 그구간을 벗어나는게 좋겠지요, 히스테리시스가 반복되면 변속기에 충격이 가해져 않좋은 영향을 미치기 때문에 인위적으로 탈피하는 것이 가장 좋겠습니다.

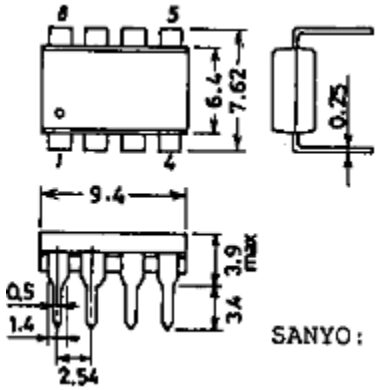
DC Motor Control

Mnolithic Digital IC

No.1261E

LB1630

Low-Saturation Bidirectional Motor Driver for Low-Voltage Applications

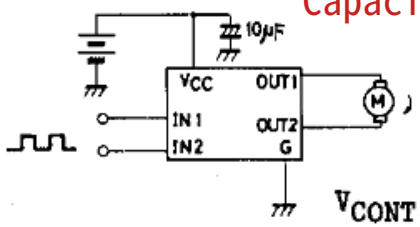


SANYO: DIP8

Truth Table

IN1	IN2	OUT1	OUT2	MOTOR
H	L	H	L	Forward
L	H	L	H	Reverse
H	H	off	off	Standby
L	L	off	off	Standby

Sample Application Circuit



Capacitor(10uF)

DC Motor

PWM Input

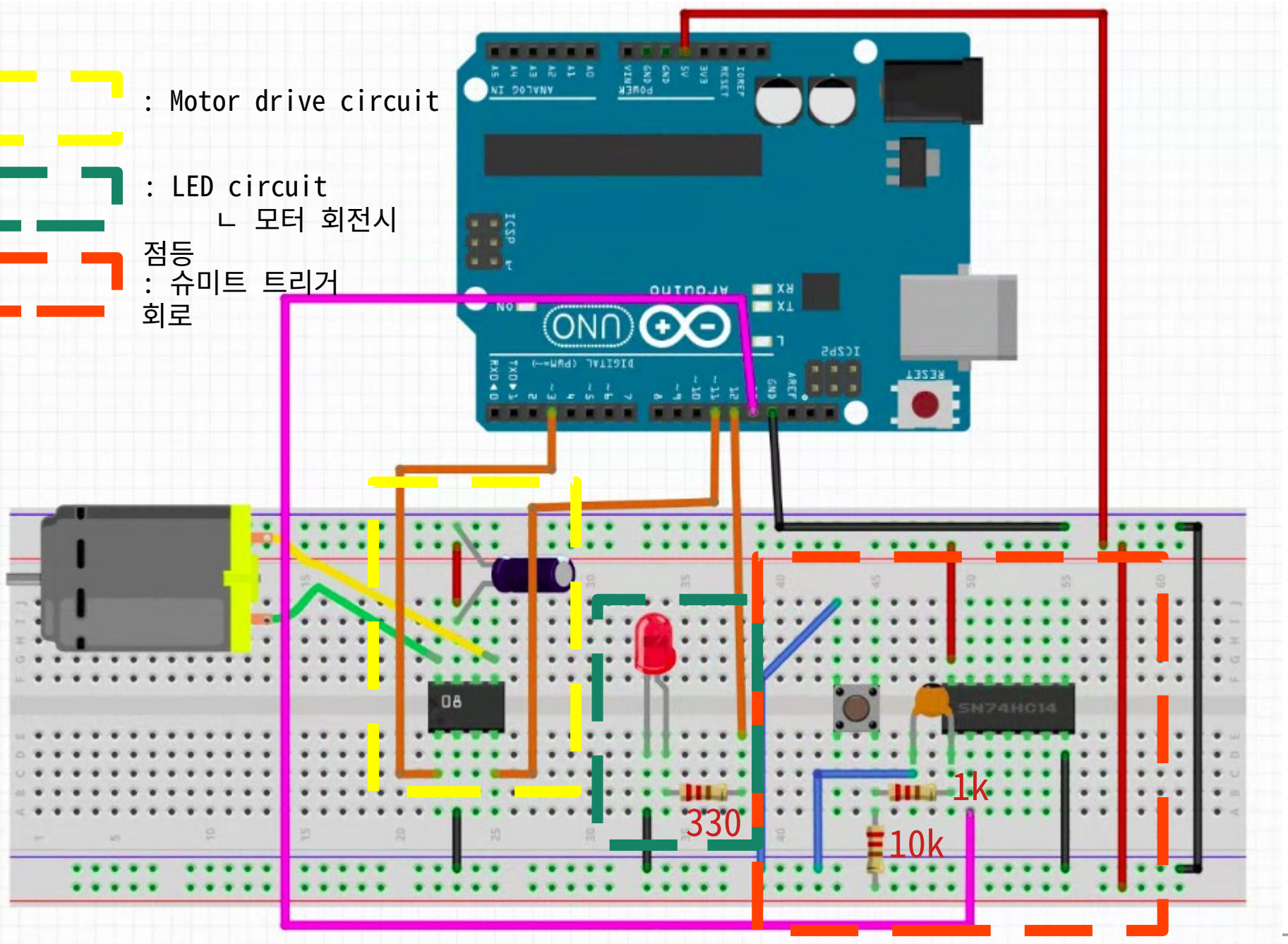
V_{CONT} pin: Open

Motor 드라이버로 사용 된 LB1630 IC 이다.
Table을 보면 IN1,2에 넣는 값에 따라 모터의 정회전, 역회전을 만들어 낸다.

: Motor drive circuit

: LED circuit
ㄴ 모터 회전시 점등

: 슈미트 트리거 회로



LB1630_DC_Motor_Control(Header, PWM_init)

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

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

#define IN1     PD0

#include <math.h>

void init_dc_motor(void)
{
    DDRB = 0xFF;
    DDRD = 0xFF;

    // Timer/Counter
    // WGM22, WGM21, WGM20 --> Fast PWM, OCRA
    TCCR2A |= (1 << WGM21) | (1 << WGM20);
    // COM2A1, COM2B1
    // Clear OC2B on compare math, set OC2B at BOTTOM,
    // (non-inverting mode). 0 으로 떨어지기 때문에 non. 반대로 1로 올라가는 경우가 inverting이 되겠다.
    // 둘 다 매칭이 되면 클리어.
    TCCR2A |= (1 << COM2A1) | (1 << COM2B1);
    TCCR2B |= (1 << WGM22);
    // 64 분주 => 16000000 / 64 = 250000
    TCCR2B |= (1 << CS22);
}
```

Interrupt 및 delay
헤더 선언

IN1은 PORTD의 0번핀

Math 헤더 선언

FAST PWM 설정
ㄴ TCCR2A,B 의 WGM 비트와 COM2A1, COM2B1 비트는
servo 모터 설명시와 같음.

분주비만 한번더 상기 시켜 본다.
16000000 / 64 = 250 kHz

Table 17-9. 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)

LB1630_DC_Motor_Control(interrupt 동작, 슈미트 트리거 설정)

```
ISR(PCINT0_vect)    // 인터럽트 서비스 루틴 ISR
{
    PORTB ^= 0x10; // toggle, PB0을
}

int main(void)
{
    int count = 0;
    double speed, t;
    unsigned char velocity;

    init_dc_motor();
//-----------------------------------슈미트 트리거 부분.
    // PB0 pin 초기화
    // PB0을 입력으로 설정
    DDRB &= ~(1 << DDB5);

    //pull-up 설정
    PORTB |= (1 << PORTB4);

    // PCIE0을 통해 PCMSK0 스캔
    PCICR |= (1 << PCIE0);
    // PCINT0을 상태 변경에 따른 인터럽트 트리거로 활용
    PCMSK0 |= (1 << PCINT5);
//-----------------------------------------------
    // 인터럽트 활성화
    sei();
```

인터럽트 서비스루틴 동작
ㄴ 인터럽트 동작 발생시 마다 PORTB 동작 반전.

PORTB의 5번핀 입력으로 설정.

PORTB의 4번핀 Pull-up

PCICR – Pin Change Interrupt Control Register

Bit	7	6	5	4	3	2	1	0	
(0x68)	–	–	–	–	–	PCIE2	PCIE1	PCIE0	PCICR
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.

PCMSK0 – Pin Change Mask Register 0

Bit	7	6	5	4	3	2	1	0	
(0x6B)	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
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 – 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.

PCINT0 상태 변경에 따른 인터럽트 트리거로 활용

LB1630_DC_Motor_Control(degree → radian, 속력, 속도, 정/역 방향 제어)

```
while(1)
{
    // count * PI / 180; ---> 1 degree를 구한 것, count가 1이면 1도
    // 호도법. 1 rad = 56.x degree
    // 360 degree = 2 PI rad
    // 1 degree = 2 * PI / 360 rad
    //          = PI / 180 rad
    // 호도법(각도와 라디안의 관계)
    // 이유 : math 라이브러리의 sin, cos, tan 등은 radian 기반으로 동작하기 때문.
    t = (double)count / 180.0 * 3.141592; // 각도를 rad 으로 변환
    // 255 * (0~360 degree)
    // sin의 특성 : -1 ~ +1
    // -255 ~ 255, avr이 8bit 인데, atmega328p가 16bit 까지 support 해줘서 255 씀. 그냥 8bit 였으면 127
    speed = 255 * sin(t);
    // speed 는 벡터
    velocity = (unsigned int)fabs(speed); // fabs는 소수점 절대 값
    // velocity는 스칼라(속력)

    // 양방향
    // OCA와 OCB를 통해 정방향, 역방향을 제어함.
    if(speed > 0)
    {
            PORTD |= 0x08;  // PD3, OC2B
            PORTB &= ~0x08; // PB3, OC2A
            OCR2A = 255 - velocity;
    }
```

Math 라이브러리의 sin, cos, tan 등은 radian 기법으로 동작하기에 각도 값을 구하기 위해 호도법을 활용해서 rad 변환 한다.
 ㄴ(각도 count → rad 변환 값 t)

각도 * PI / 180 = rad 값

360 degree = 2 PI rad
1 degree = 2 PI / 360 rad
= PI / 180 rad

Speed = 255 * sin(t)
 ㄴ atmega328 이 16bit 계산 까지 support 해주어서
-255 ~ 255 이기에 sin 값이 +1 ~ -1특징을 가지므로 255의 숫자만 적용하면 된다.
(8bit 였으면 127)
 ㄴ speed 는 속도 벡터(+,-)

Velocity = fabs(speed), fabs는 소수점 절대 값.
 ㄴ velocity는 스칼라(속력, +절대값)

참고) 원래는 속력을 speed
속도를 velocity 로 나타낸다.

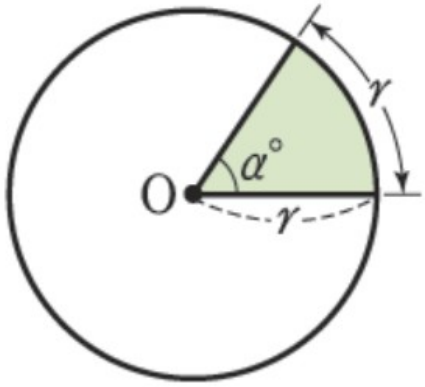
추가 정보
- 호도법?

■ 호도법(弧度法)

弧 활 호 (곡선이나 원 둘레의 일부분)

度 법도 도 (자 또는 도구)

[circular method]



반지름의 길이가 r인 원에서
길이가 r인 호에 대한 중심각의 크기를
α°라 하면 호의 길이는
중심각의 크기에 정비례하므로

$$r : 2\pi r = \alpha^\circ : 360^\circ$$

$$\therefore \alpha^\circ = \frac{180^\circ}{\pi}$$

따라서
중심각의 크기 α°는
반지름의 길이 r에 관계없이 일정하다.

이 일정한 각의 크기 α°를
1호도 또는
1라디안(radian)이라 하고,

이것을 단위로 하여
각의 크기를 나타내는 방법을
호도법이라 한다.

$$1\text{라디안} = \frac{180^\circ}{\pi}, \quad \pi\text{라디안} = 180^\circ, \quad 1^\circ = \frac{\pi}{180}\text{라디안}$$

추가 정보

- 속력? 속도?
  ㄴ 흔히들 속력은 스칼라 값, 속도는 벡터 값이라고 한다. 그럼 이 말이 무슨 말인가?

벡터와 스칼라

물리적 현상을 양적으로 표현하는 방법에는 벡터 (vectors)와 스칼라 (scalars)가 있다.

스칼라	벡터
크기만을 나타내는 물리량	크기와 방향을 나타내는 물리량
길이, 넓이, 이동거리, 부피, 시간, 온도, 질량, 무게, 속력, 에너지 등	변위, 속도, 가속도, 힘, 운동량, 전기장, 자기장, 각 운동량 등

자 그러면 여기서 속력과 속도를 보기 전에, 이동거리와 변위가 무엇인지 보고 갑시다.

이동거리는 '물체가 이동한 총 거리'
변위는 ' 물체의 위치 변화량'

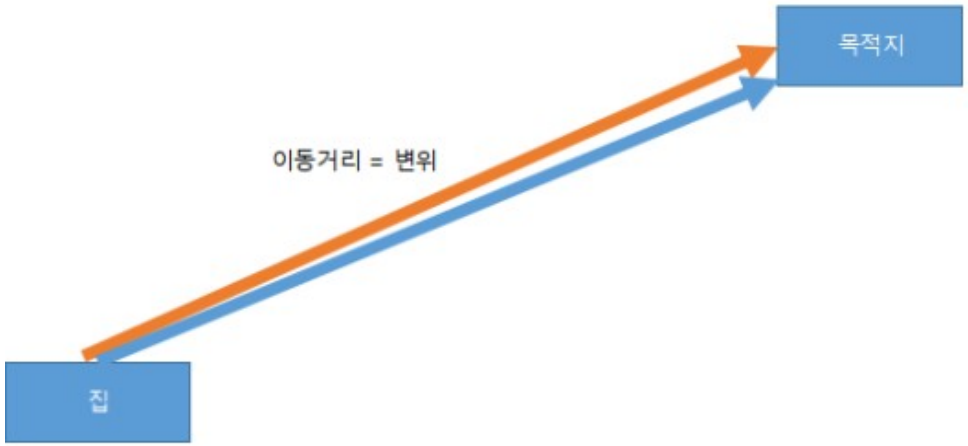
비슷해 보이지만 차이가 있습니다.

예를 들어, 한 사람이 집에서 목적지로 가려고 합니다.

오른쪽과 같이 곧바로 가는 경우는,

이동거리와 변위가 같은 것이죠.

하지만 무조건 저렇게 가는 것은 아닙니다.
(뒤 page 계속)



LB1630_DC_Motor_Control(degree → radian, 속력, 속도, 정/역 방향 제어)

때로는 오른쪽 그림과 같이

저렇게 돌아서 갈 수도 있습니다.

이때의 이동거리는 돌아서 간 거리를 말하는 것이고,

변위는 집에서부터 목적지 까지의 가장 짧은 거리를

말하는 것 입니다.

자 그러면 이제 속력과 속도를 한번 보겠습니다.

속력은 '물체의 빠르기를 나타낸 값'

$$v = \frac{s}{t}$$ (단위 : m/s, km/h)

속력을 구하는 공식입니다.

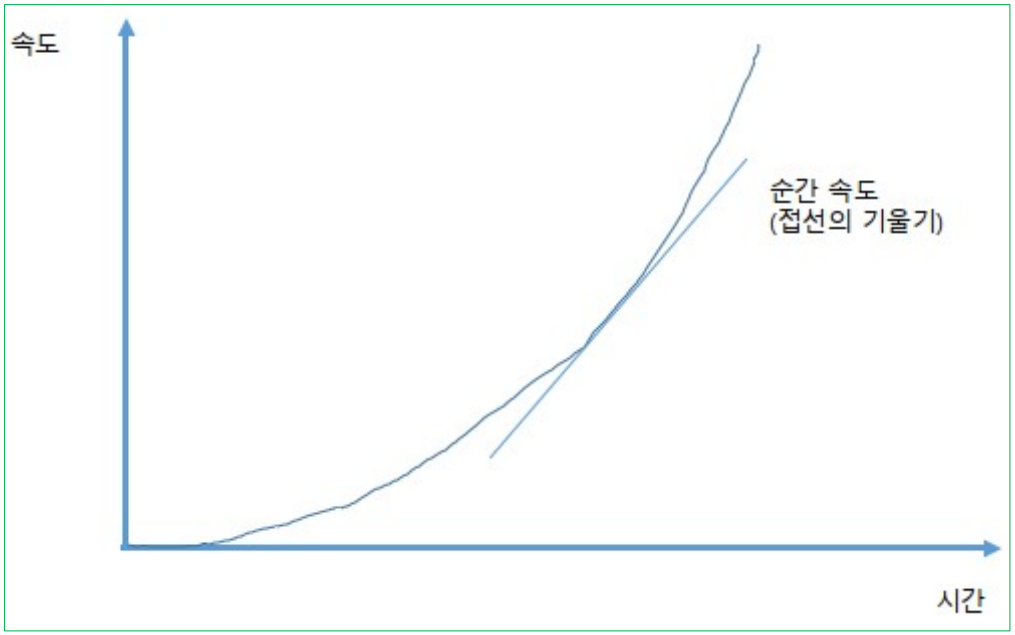
속도는 '물체의 빠르기 뿐만 아니라 운동 방향까지도 갖고 있는 값

$$\vec{v} = \frac{\vec{s}}{t}$$ (단위 : m/s, km/h)

그리고 추가적으로 순간속도, 평균속도 라는 것이 있습니다.
순간 속도는'어떠한 순간의 속도 ' 를 말합니다.

어느 한 시간의 점에서의 접선의 기울기가
순간 속도가 되는 것이죠.

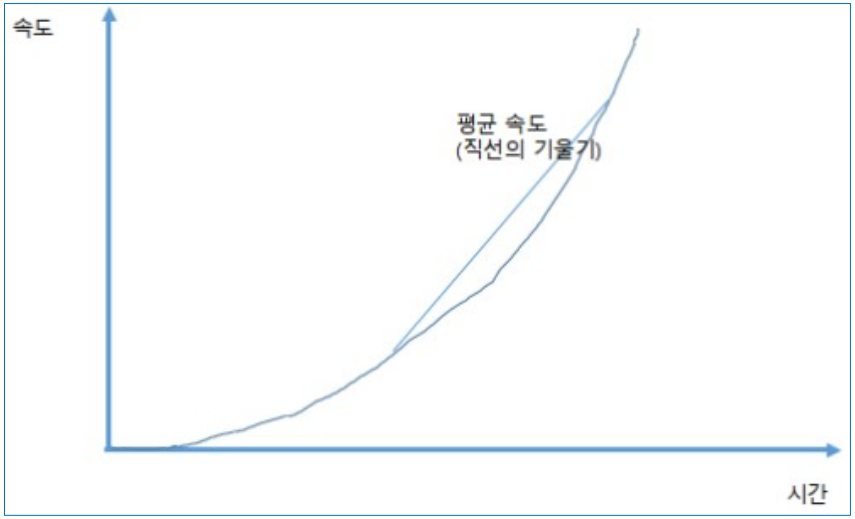
미분하라 이 소리 입니다.



평균 속도는 '어느 시간 동안의 속도의 평균을 나타낸 값 ' 입니다.

나중속도에서 처음속도를 뺀 값을

걸린 시간으로 나누어 주면 평균 속도가 나옵니다.



LB1630_DC_Motor_Control(degree → radian, 속력, 속도, 정/역 방향 제어)

```
speed = 255 * sin(t);
// speed 는 벡터
velocity = (unsigned int)fabs(speed); // fabs는 소수점 절대 값
// velocity는 스칼라(속력)
```

그런데 코드상에서 속도 값을 구할 때 sin을 활용해서
구하는 것을 볼 수 있다.

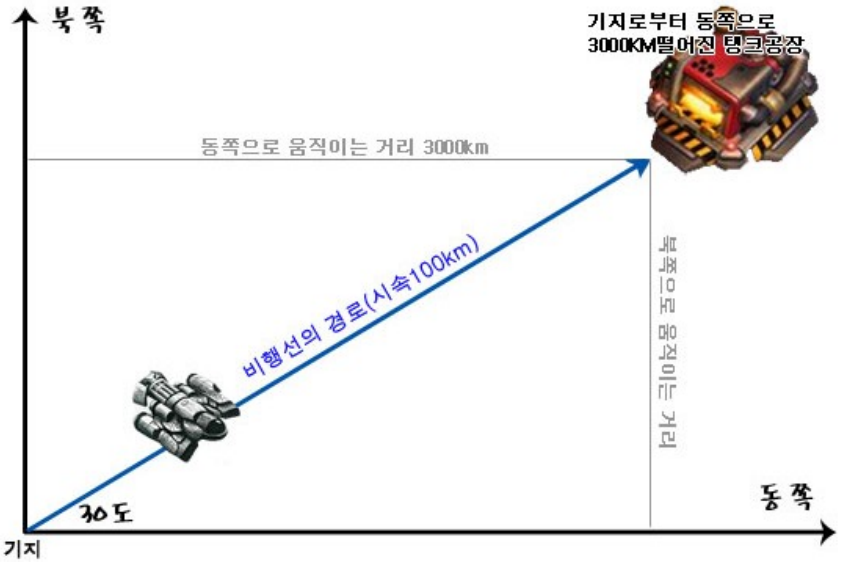
이건 뭐지?? 라는 궁금증으로 더 조사해 본다.

자, 다시한번 속도는 **어떠한 물체가 속력(힘)으로 움직이는 운동상태에서 방향(변위)가 추가** 된 것이다.
속도를 알기 위한 성분으로는 동서남북과 같은 또는 x축 y축과 같은 이동하는 방향에 따른 기준이 될 수 있는 속도벡터를 계산 할 수 있다.

- 속도벡터 : 운동하고 있는 물체의 각 시점에서의 순간 속도에 비례하는 길이를 갖고, 운동하는 방향과 나란하게 그은 벡터

가령 북쪽으로 시속 10키로로 향하고 동쪽으로 시속 10키로로 이동하는 물체가 있다고 할 때, 북쪽으로 이동하는 성분과
동쪽으로 이동하는 성분의 크기를 가지고 북동쪽으로 이동하는 선을 속도벡터라고 하고 시간과 거리를 삼각법으로 구할 수 있다.

문제) 기지에 드랍쉽이 출발하여 공장에 탱크들을 태우러 가야한다. 탱크공장은 기지에서 동쪽으로 3000KM 떨어져 있고
지도상의 각도는 북쪽으로 30도에 있을 때 비행선이 공장으로 시속 100 km씩 직선운동을 한다면 몇 시간 후에 도착 할까?



기지로 부터 공장까지의 직선 각도는 30도 공장의 위치는 x=3000
따라서 밑변 값과 각도를 알고 있으므로
Cos에 관한 식을 세우면 cos(30) = 시간당 동쪽 벡터/빗변(시간당 이동거리)
비율 이므로 현재 이동할 총 거리는 몰라도 시속 100 km를 대입해서

한시간 동안의 이동거리를 정리 하면

동쪽으로 1시간 동안 이동하는 거리 = cos(30) * 100km 이고
값은 약 86.6km가 나온다.

따라서 동쪽 총 거리 3000 / 시간당 움직이는 동쪽 거리 86.6 을 하면

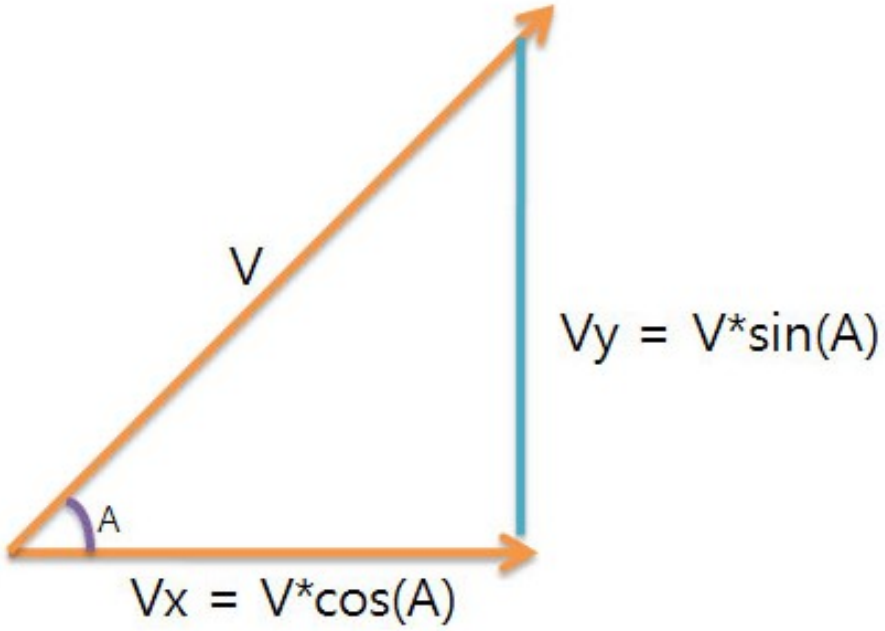
34.64 시간 후에나 도착 할 것 같다.

앞선 내용의 속도 성분을 정리하여 보면 아래와 같다고 할 수 있다.

• 속도성분

일단 두 방향을 정하고 각각 x 방향, y 방향이라고 한다. (예를 들어 이전 경우 처럼 x는 동쪽, y는 북쪽으로 정하고 두 방향은 반드시 직각 이어야 한다) 그리고 v가 아무 방향의 속도라고 할 때 x 성분과 y 성분은 각각 다음의 공식으로 계산 할 수 있다.

Vx = V * cos(각도)
Vy = V * sin(각도)



[속도성분 계산식]

LB1630_DC_Motor_Control(degree → radian, 속력, 속도, 정/역 방향 제어)

자, 이제 다시 코드로 넘어와 본다.

```
        // 양방향
        // OCA와 OCB를 통해 정방향, 역방향을 제어함.
        if(speed > 0)
        {
                PORTD |= 0x08;  // PD3, OC2B
                PORTB &= ~0x08; // PB3, OC2A
                OCR2A = 255 - velocity;
        }
        else
        {
            PORTD &= ~0x08;
            PORTB |= 0x08;
            OCR2A = velocity;
        }

        count++; // sin 웨이브를 만듬. 실제로는 계단처럼 올라가는 것을 볼 수 있음.
        // 속도의 변화율이 가속도 이므로
        // 딜레이가 크면 가속도가 작고 작으면 가속도가 크다.
        // dv / dt
        // (나중 속도 - 이전 속도) / 이동시간 = 평균 속도, 평균속도가 올라간다는건 가속도가 빠르다는 소리.
        // (나중 속도 - 이전 속도) / 시간(짧다고 가정) = 근사 가속도.
        // (현재 샘플 - 이전 샘플) / 시간(샘플링 타임, 샘플링 비트 수).
        // 이와 같이 디지털로 미적분을 수행하는 기법을 수치해석이라 한다.(미분은 아니지만 미분 같은)
        _delay_ms(20); // 느리게 해주면 sin 웨이브가 느리게 올라 가겠지. 가속도의 의미로 활용도 가능 하겠지. 속도를 미분 하면 가속도.
    }
    ;

    return 0;
}
```

Speed 값 (속도 값이 0 이상 일 때)

PORTD 의 3을 High
PORTB 의 3을 Low

주고 반대의 경우는

반대의 값을 넣어 주어 정방향/역방향 제어를 구현 했다.

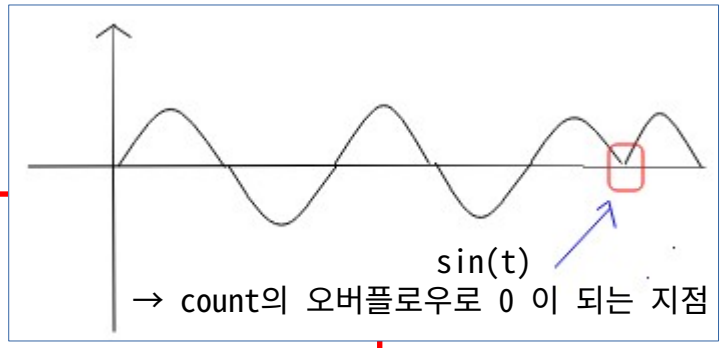
OCR2A 에는 속력(스칼라) 값을 넣어주어
모터 PWM을 제어 하여 모터의 속도를 제어 하였다.

LB1630_DC_Motor_Control(degree → radian, 속력, 속도, 정/역 방향 제어)

```
        // 양방향
        // OCA와 OCB를 통해 정방향, 역방향을 제어함.
        if(speed > 0)
        {
                PORTD |= 0x08;  // PD3, OC2B
                PORTB &= ~0x08; // PB3, OC2A
                OCR2A = 255 - velocity;
        }
        else
        {
            PORTD &= ~0x08;
            PORTB |= 0x08;
            OCR2A = velocity;
        }

        count++; // sin 웨이브를 만듬. 실제로는 계단처럼 올라가는 것을 볼 수 있음.
        // 속도의 변화율이 가속도 이므로
        // 딜레이가 크면 가속도가 작고 작으면 가속도가 크다.
        // dv / dt
        // (나중 속도 - 이전 속도) / 이동시간 = 평균 속도, 평균속도가 올라간다는건 가속도가 빠르다는 소리.
        // (나중 속도 - 이전 속도) / 시간(짧다고 가정) = 근사 가속도.
        // (현재 샘플 - 이전 샘플) / 시간(샘플링 타임, 샘플링 비트 수).
        // 이와 같이 디지털로 미적분을 수행하는 기법을 수치해석이라 한다.(미분은 아니지만 미분 같은)
        _delay_ms(20); // 느리게 해주면 sin 웨이브가 느리게 올라 가겠지. 가속도의 의미로 활용도 가능 하겠지. 속도를 미분 하면 가속도.
    }
    ;

    return 0;
}
```

Count는 위 코드상에서 보면 0~360 degre를 의미하는 값이 되고,
Rad 값으로 변환 후 속도 값을 구할 때 삼각법을 이용해서 결과 값이 결국 sin 웨이브를 만들게 된다.

이 단계를 delay를 활용해 텀을 줄 수 있고
이 텀이 짧을 수록 속도 및 속력은 상승하게 되는 것이다.



선생님 꿀팁)
이 때문에 스피드가 x에서 x+1, x-1이 아닌 매끄럽지 못하게 0이 되는 구간이 발생합니다.

질문

```
// PCIE0을 통해 PCMSK0 스캔
PCICR |= (1 << PCIE0);
// PCINT0을 상태 변경에 따른 인터럽트 트리거로 활용
PCMSK0 |= (1 << PCINT5);
```

1번 질문)

슈미트 트리거 코드에서 PCICR 에서 PCIE0를 1 설정하면
PCINT0..7 이 동작 선택 되고
PCMSK0에서 PCINT5를 1 설정 하면
PCINT5를 상태 변경에 따른 인터럽트 트리거로 활용 하는게 아닌가요??

```
ISR(PCINT0_vect)    // 인터럽트 서비스 루틴 ISR
{
    PORTB ^= 0x10; // toggle, PB0을
}
```

2번 질문)

분주비 64로 인해 250 khz가 주기 인데, 그렇다면
인터럽트 동작은 1/주기 = 0.004 ms 마다 인터럽트 동작이 발생하는 건가요?

```
if(speed > 0)
{
        PORTD |= 0x08;  // PD3, OC2B
        PORTB &= ~0x08; // PB3, OC2A
        OCR2A = 255 - velocity;
}
else
{
    PORTD &= ~0x08;
    PORTB |= 0x08;
    OCR2A = velocity;
}
```

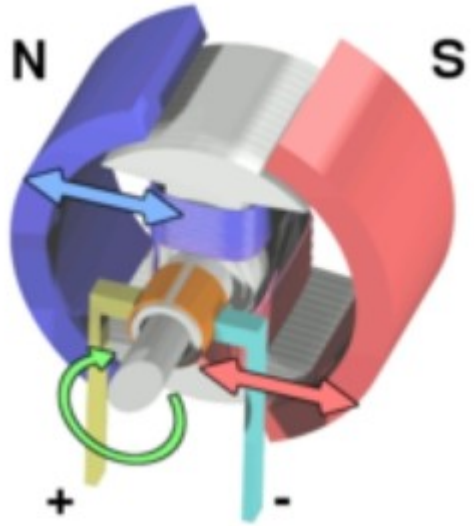
3번 질문)

If문에서 OCR2A 값을 처음에 255 에서 속력 값을 뺀 이유가 있나요?
그냥 정/역 방향에 따라 속도를 다르게 하기 위함 일까요??

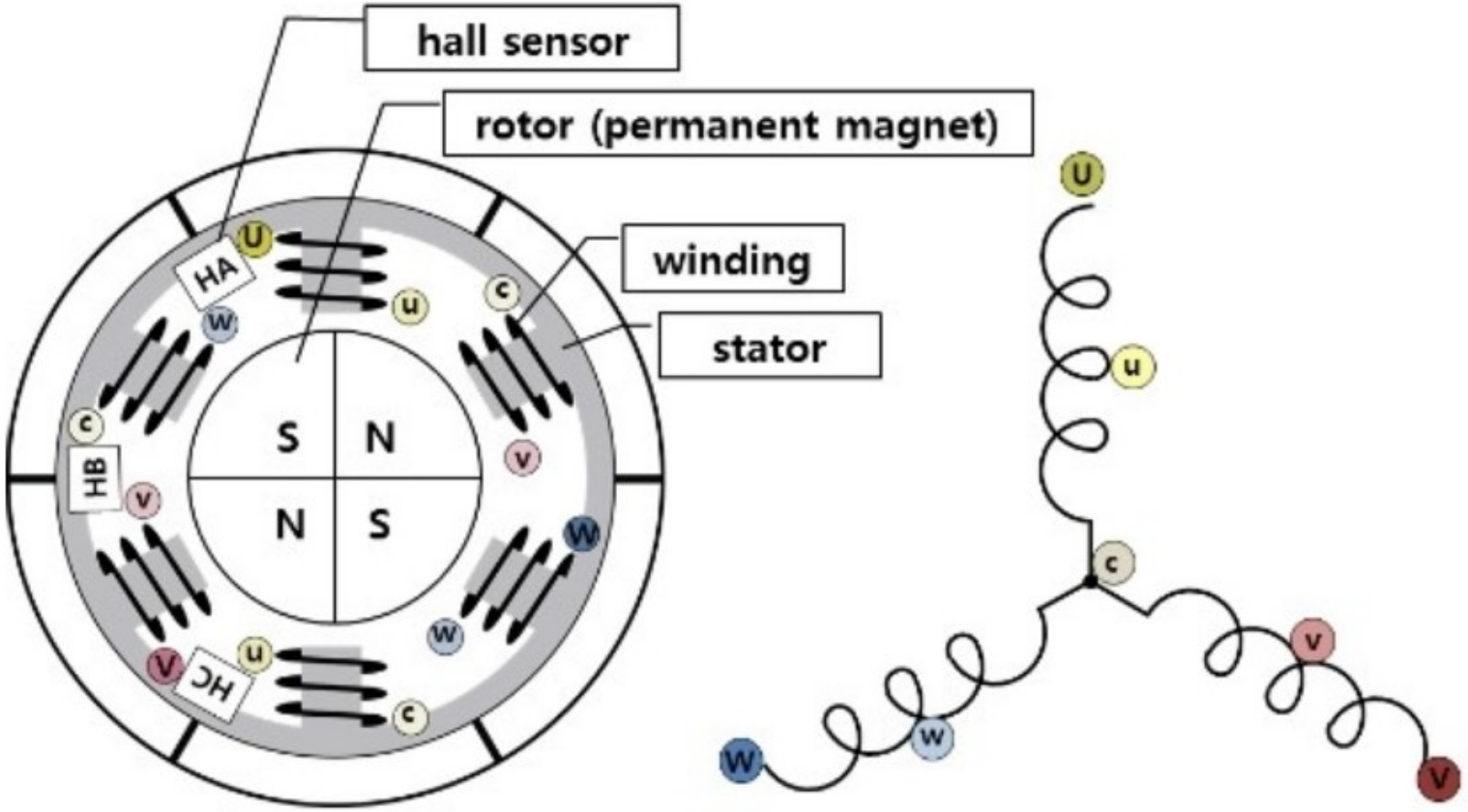
BLDC Motor 제어기

BLDC 모터의 구조

ㄴ DC 모터는 브러쉬를 통한 기계적인 접촉 구조이냐 아니냐에 따라 Brushed DC 모터와 Brushed less DC 모터로 구분 합니다.



Brushed DC 모터는 2개의 전선으로만 구성되어 있으며 전동기를 구동 시키기 위한 드라이버의 설계 및 제어가 용이하다는 장점이 있지만, Brush의 접촉을 통해서 회전에 따라 전기자 전류의 극성이 바뀌게 되므로, 기계적 소음과 전기적 잡음이 심하며 내구성이 떨어집니다.

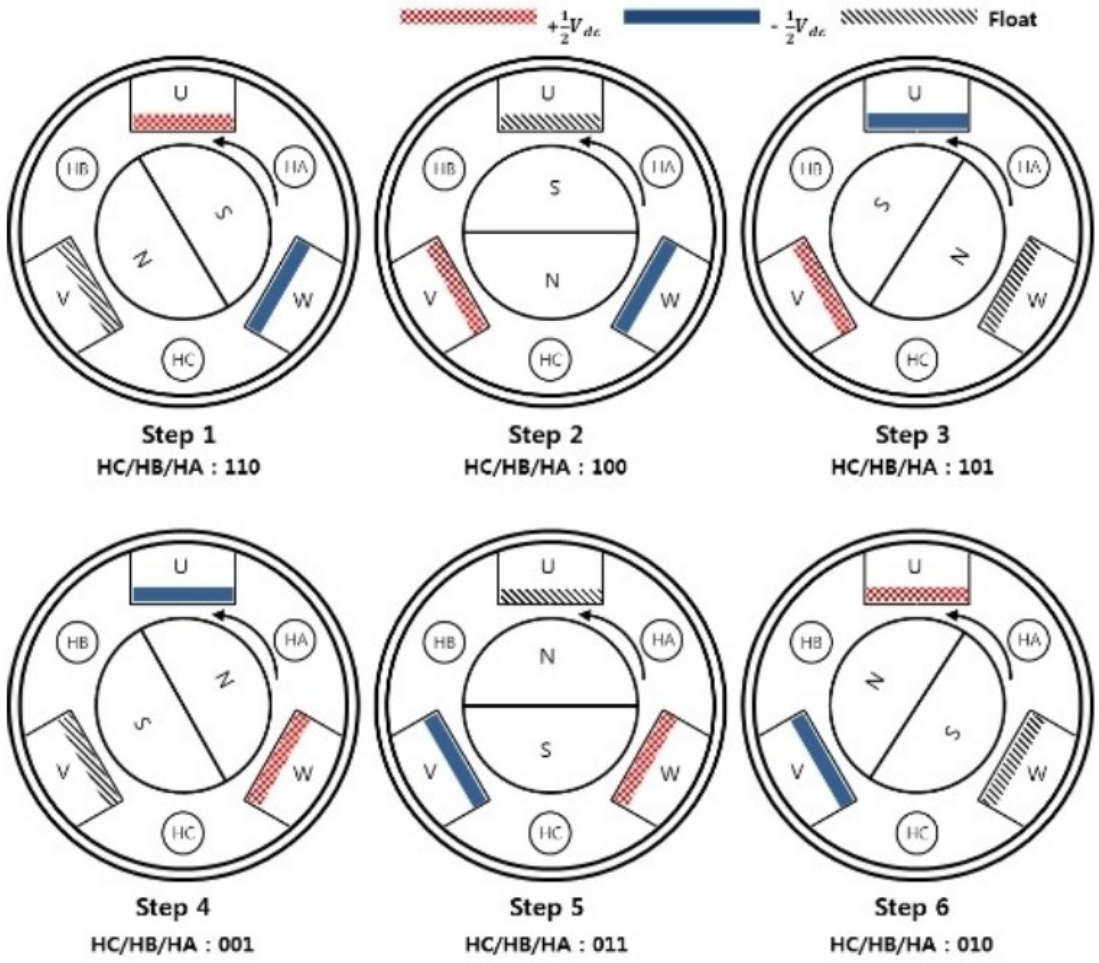


반면, BLDC 모터는 Brush가 제거된 형태 입니다. 위 그림에서 보이듯 BLDC 모터는 가운데 영구자석으로 되어 있는 회전자, 고정자의 이빨에 권선이 감겨 있는 형태 입니다. 또한 BLDC 모터의 영구자석(회전자)을 회전 시키기 위해서는 영구자석의 위치 및 극성에 따라 회전자에서 정확한 시점과 정확한 방향으로 자속을 발생시켜야 합니다. 따라서 영구자석의 위치를 검출하기 위해 홀 센서 3개가 필요 합니다.

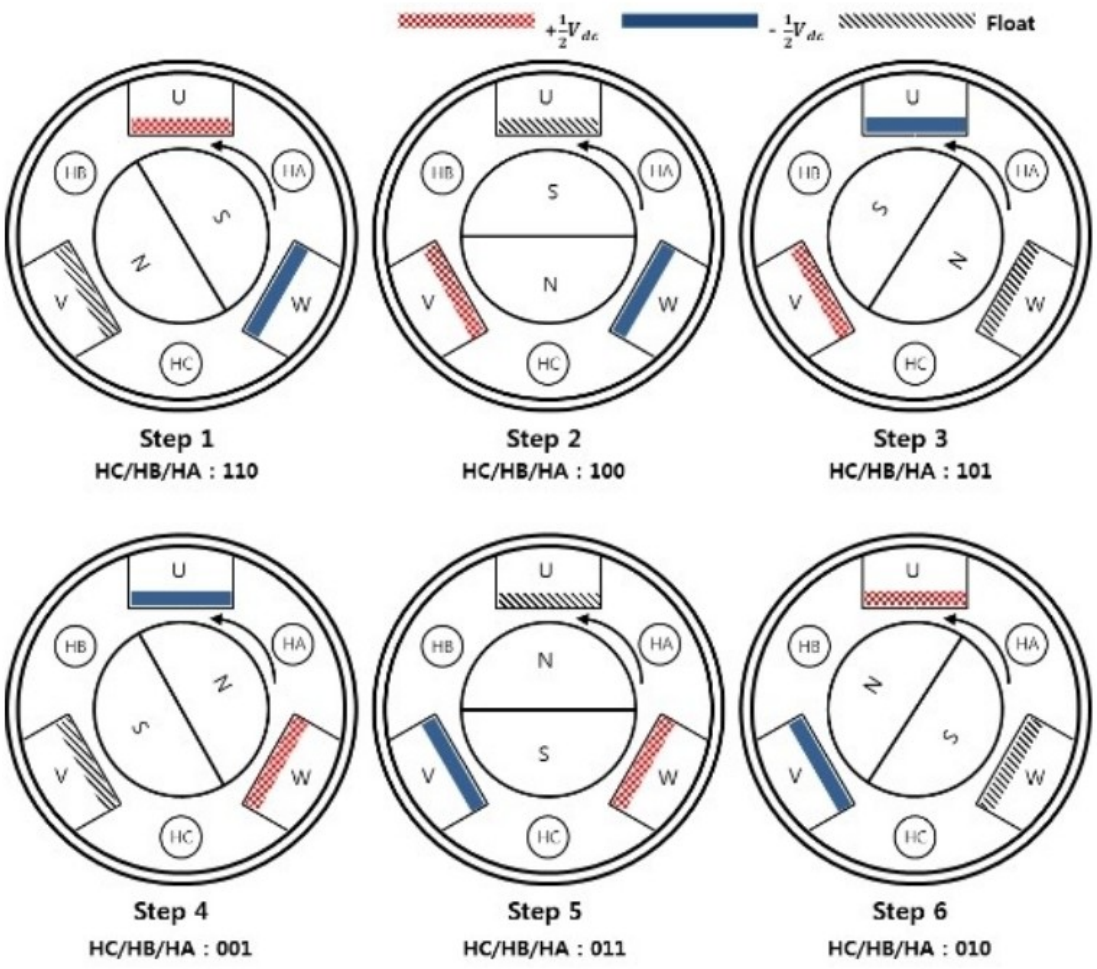
그리고 Brush DC 모터는 전류를 스위칭 하는 전선이 2개인 것과 다르게, BLDC 모터는 3상을 스위칭 하여 제어하기 때문에 3개의 전선이 필요 합니다.

BLDC 모터의 구동 원리

ㄴ BLDC 전동기를 구동 시키기 위해서는 회전자(영구자석)가 회전하도록 영구자석의 위치에 따라 고정자의 권선에 전류를 흘려서 자속을 발생시킬 권선을 순시적으로 바꾸어 주어야 합니다.



3상 2극 (설명 다음 페이지)



오른쪽 그림은 3상 2극의 BLDC.

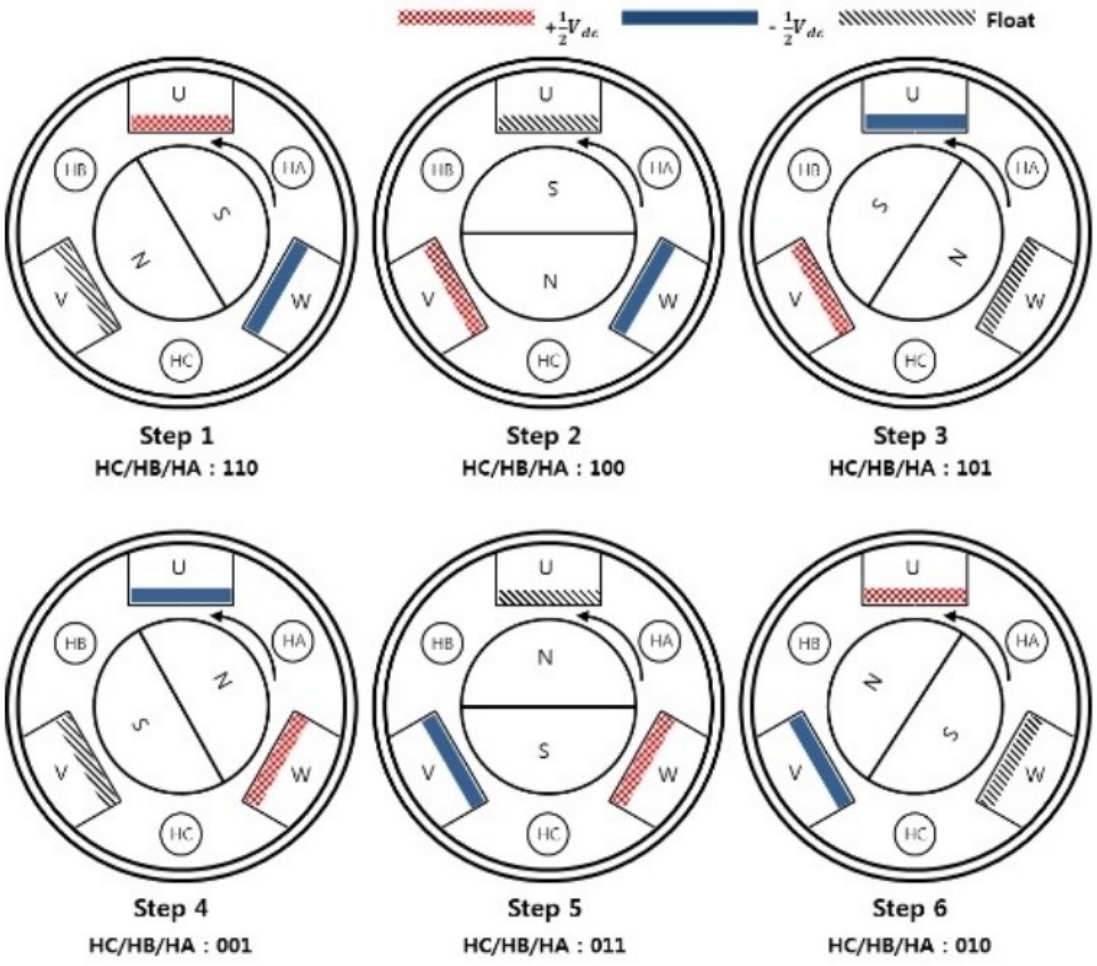
가운데 영구자석이 회전하고 고정자에는 3개의 고정자 상이 존재 합니다. 이 때 각각의 상을 U, V, W 상 이라고 합니다. 그리고 3개의 홀 센서는 HA, HB, HC 혹은 HU, HV, HW 이라 합니다.

가정1 : 고정자의 권선에 + 방향으로 전류를 흘리면 N극이 되고, - 방향으로 전류를 흘리면 S극이 된다고 가정.
가정 2 : 홀센서가 자석의 S 극이 다가올 때는 LOW(0), N극이 다가올 때는 HIGH(1) 라고 가정한다.

첫번째 step 부터 살펴 보면 회전자가 반시계 방향으로 회전하기 위해서 고정자 U,V,W 상은 각각 어떤 자속을 발생시켜야 할지 보겠습니다. U 상은 회전자의 S극을 당겨야 하므로 N극이 될 수 있게 전류를 흘려야 하고, W 상은 회전자의 S 극을 밀어야 하므로 S극이 될 수 있게 전류를 흘려야 합니다. V상은 에매한 위치에 있으므로 Float 상태로 전류를 흘려 보내지 않으면 됩니다.

이 때, 홀 센서의 신호를 살펴 보겠습니다. 홀 센서가 자석의 S 극이 다가 올 때는 LOW (0), N극이 다가올 때는 HIGH (1) 라고 가정하면 HA 는 S극이 가까이에 있으므로 0, HB와 HC는 N극이 가까이에 있으므로 1이 됩니다. 따라서 HC/HB/HA의 신호 조합은 110 이라 할 수 있습니다.

다시 말해, 홀 센서의 조합이 110 이 나오면 U는 N극이 될 수 있게, W 상은 S극이 될 수 있게 전류를 흘려주면 회전자가 회전하게 됩니다.

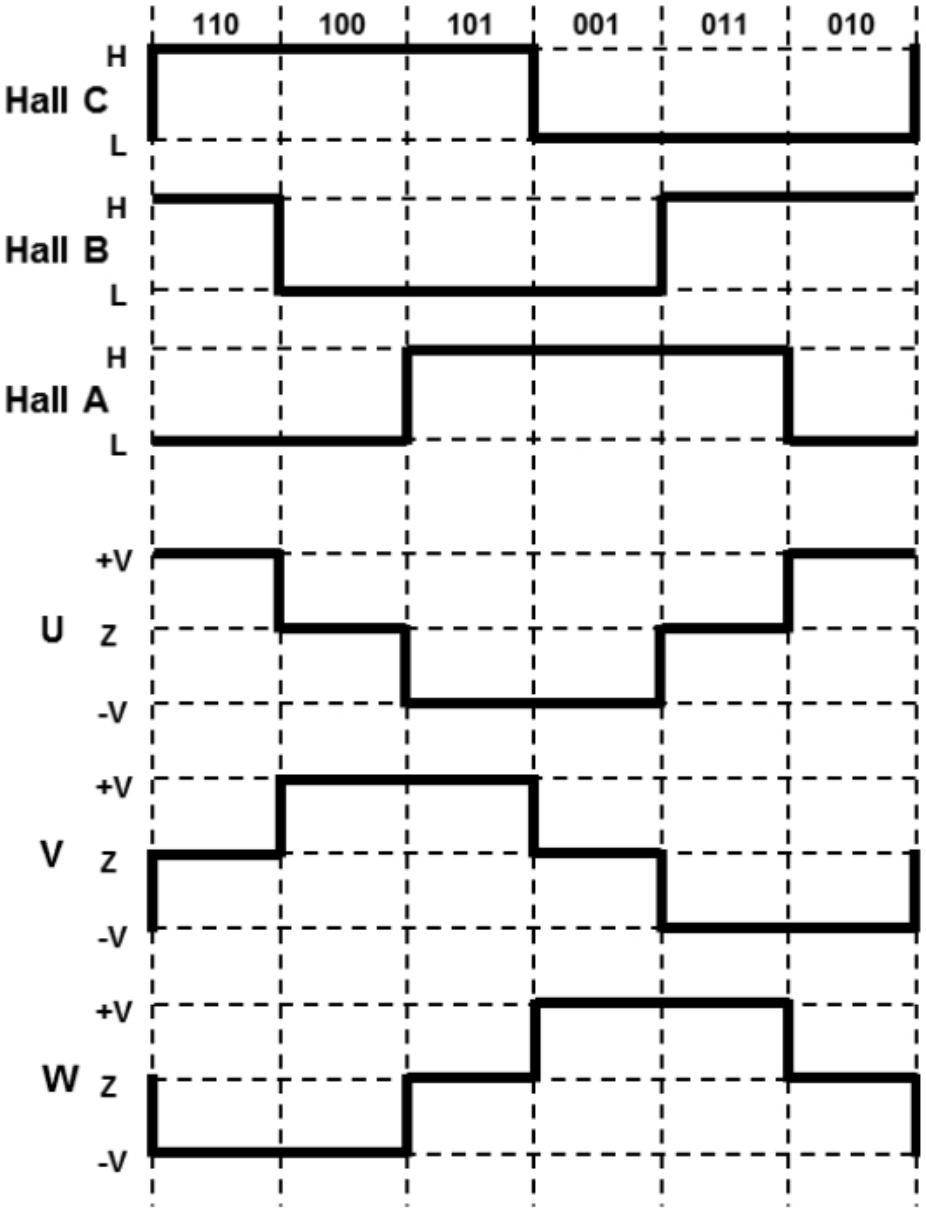


BLDC 전동기의 회전 순서

두 번째 step 역시 마찬가지 입니다.

Step2를 살펴보면 이번에는 회전자의 N극을 밀어내기 위해서는 V상이 N극, 회전자의 N극을 당겨 주기 위해서는 W 상이 S극이 되면 됩니다. 이 때 U상의 위치는 에매하므로 float 상태로 둡니다. Step 2의 홀센서 신호를 조합하면 HC/HB/HA : 100 이 됩니다.

같은 방식으로 6 step을 돌고 나면 제자리로 오게 됩니다.



BLDC 전동기의 순서 차트

앞 서서 얻어낸 홀 센서의 신호의 조합과 그에 따른 U,V,W 상의 조합을 순서 차트로 나타내면 오른쪽 그림과 같습니다.

현재의 홀 센서 신호를 알고 U,V, W 상에 어떤 전류를 보내야 할 지 알고 있으므로 연속적으로 회전이 가능 한 것 입니다.

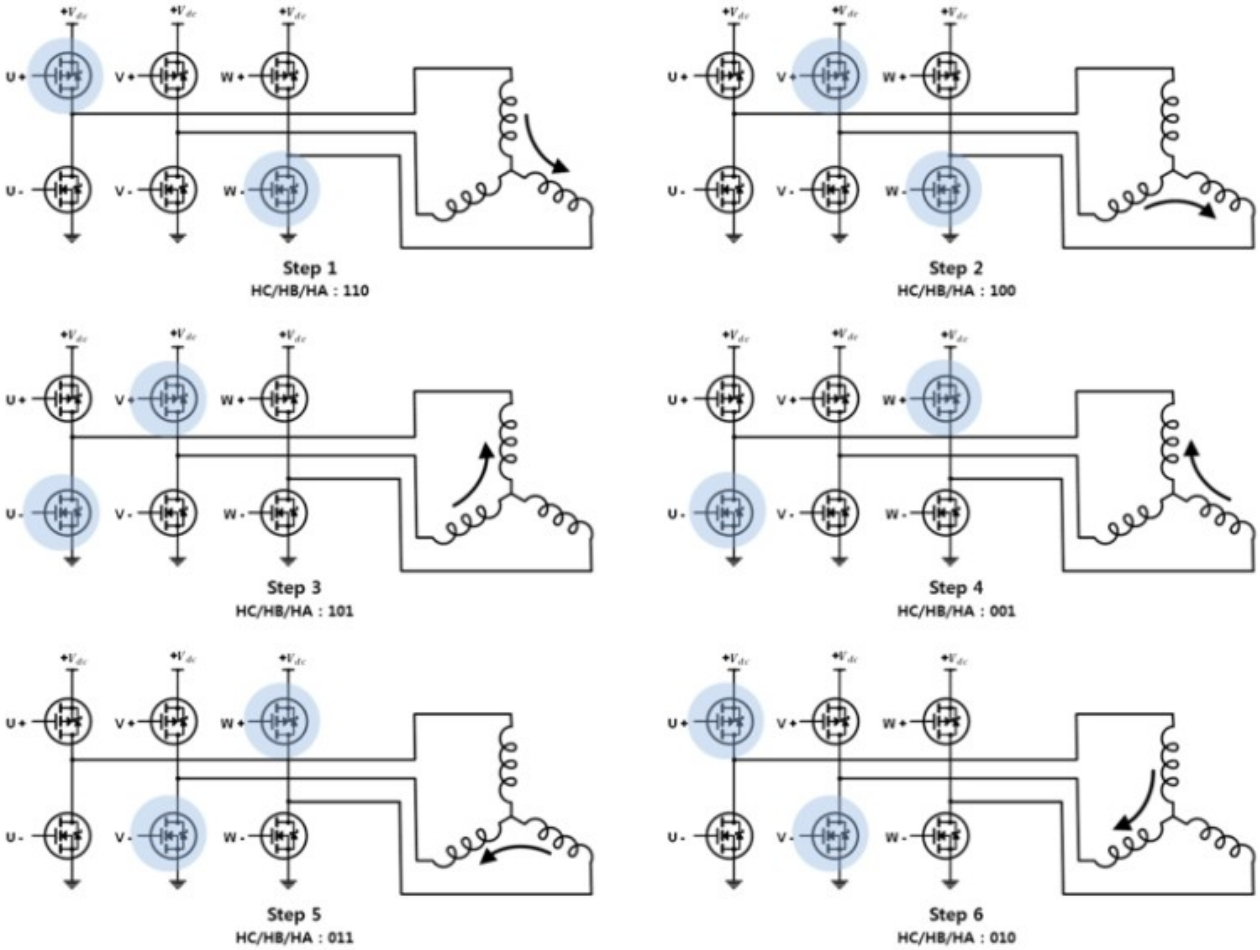
전동기의 순서 차트를 보면 다음과 같은 특징을 발견 할 수 있습니다.

1. 홀 센서는 6개의 조합을 나타내기 위해 000 과 111을 제외한 조합을 사용하며, glitch 현상(정전기 처럼 아주 짧은 순간 팍하고 일어나는 오류)이 일어나지 않도록 grey code로 이루어져 있습니다.

2. 한 step에서 3개 상의 상태는 모두 다릅니다. 다시 말해, 한상은 +방향의 전류, 한상은 -방향의 전류, 한상은 float 상태가 되겠습니다.
   2개의 상만 스위칭 하는 것을 2상 여자 방식이라 부릅니다.

여기서 잠깐) '**grey code**'

grey code는 이진법 부호의 일종으로, 연속된 수가 1개의 비트만 다른 특징을 지니는 것을 말합니다.
3bit 중 하나만 다른 종류의 비트인 모양 입니다.



BLDC 전동기의 구동 과정

이번에는 권선에 + 방향으로의 전류 혹은 - 방향으로의 전류를 어떻게 제어하는지 알아 보겠습니다.

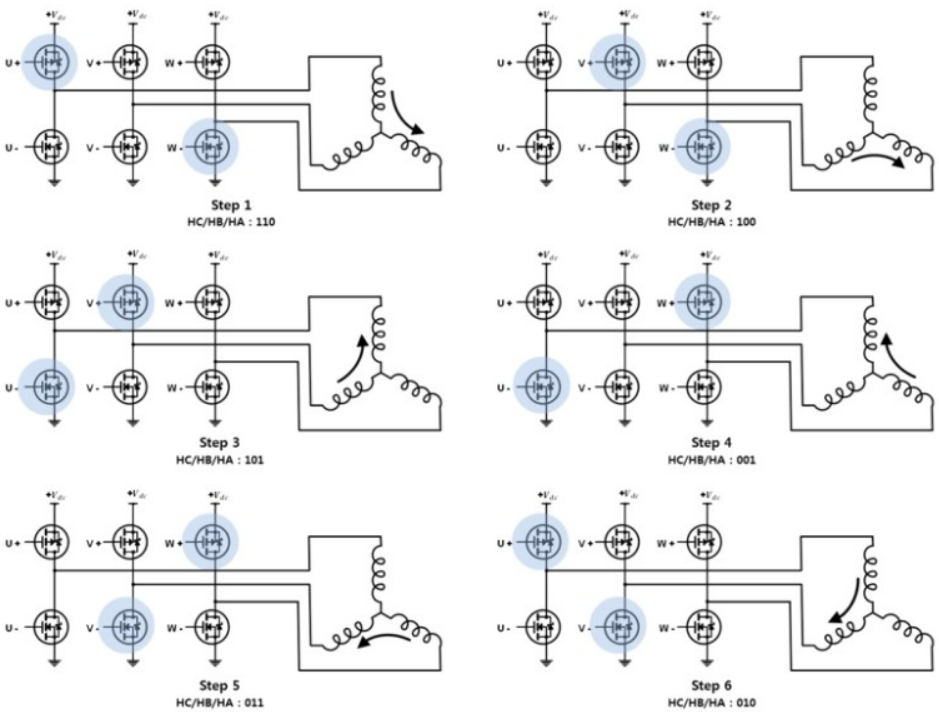
오른쪽 그림을 보면 권선 3개가 각각 U상, V상, W상에 해당하는 권선 입니다. 그리고 각 권선마다 MOSFET (스위칭 소자)가 2개씩 총 6개로 구성되어 있습니다.

위쪽에 MOSFET 3개가 각 상마다 +방향으로 전류를 흘려주게 하는 역할을 하고, 아래쪽 MOSFET 3개가 각 상마다 - 방향으로 전류를 흘려주게 합니다.

이전 그림들에서 step1의 U,V,W 상의 상태는 U 상은 +방향 V 상은 float, W상은 - 방향으로 전류가 흐르면 됩니다.
다시 말해, V상은 전류가 흐르지 않고 U 상에서 W 상으로 전류가 흐르면 되는 것 입니다.
따라서, U상의 윗단 MOSFET와 W 상의 아랫단 MOSFET를 스위칭 시켜 주면 됩니다.

Step2의 경우는 V상에서 W상으로 전류가 흘러야 하므로 V상의 윗단 MOSFET와 W상의 아랫단 MOSFET를 스위칭하여 전류를 흘려줍니다.

BLDC Motor(사전 자료 조사)



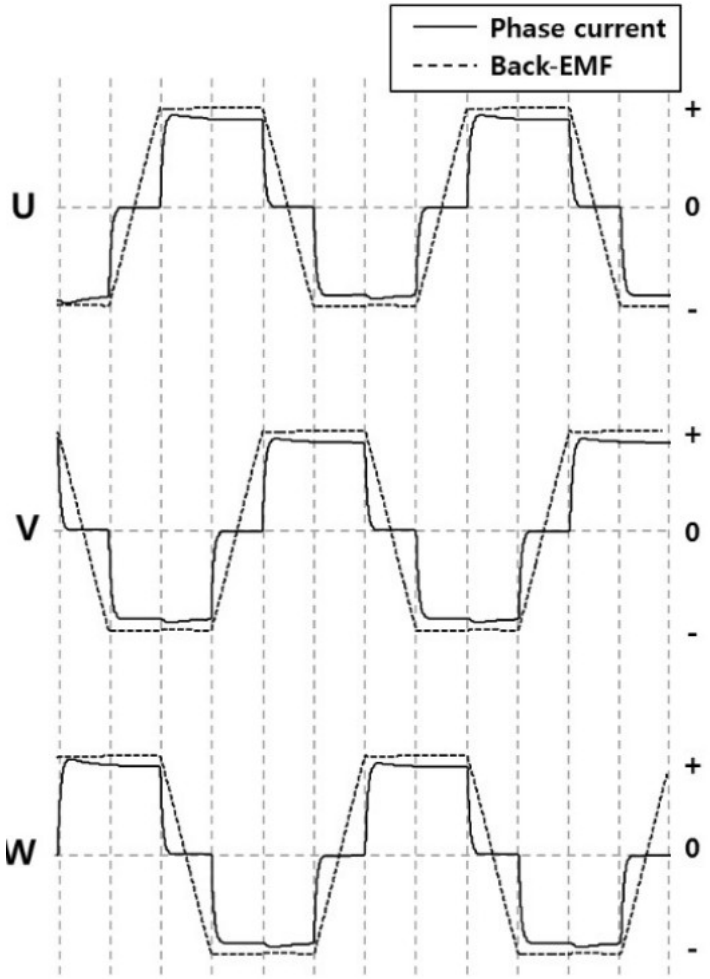
이러한 방식으로 6 step 모두 홀 센서의 조합에 따라 MOSFET를 스위칭하여 원하는 권선에 전류를 흘려주어 원하는 자속을 발생시켜 BLDC 전동기가 회전하게 됩니다.

마지막으로 홀 센서의 신호 조합과 MOSFET의 스위칭 조합을 진리표로 나타내면 아래와 같습니다.

	Hall sensor			Switching driver					
	HC	HB	HA	U+	U-	V+	V-	W+	W-
1	1	1	0	1	0	0	0	0	1
2	1	0	0	0	0	1	0	0	1
3	1	0	1	0	1	1	0	0	0
4	0	0	1	0	1	0	0	1	0
5	0	1	1	0	0	0	1	1	0
6	0	1	0	1	0	0	1	0	0

BLDC 모터의 특징

BLDC 전동기의 전류파형과 역기전력 파형에 대해 살펴 보겠습니다.



BLDC 전동기의 상전류 파형과 역기전력 파형

BLDC 전동기는 MOSFET과 같은 switching 소자에 의해 ON/OFF 되는 형태이기 때문에 float 상태와 같이 switching 소자가 OFF 인 경우는 전류가 흐르지 않게 됩니다.

따라서, 오른쪽 이미지와 같이 상전류는 준구형파(quasi-square waveform)가 나타나게 됩니다.

그리고 BLDC 전동기의 큰 특징중에 하나가 역기전력 파형이 사다리꼴(trapezoidal) 형태로 나타난다는 것 입니다.

오른쪽 그림은 각각 U,V,W 상에서 나타나는 상전류와 역기전력 파형을 나타낸 것 입니다.

사다리 꼴 형태의 역기전력을 설명하기 위해 잠깐 BLDC 전동기와 PMSM(Permanent Magnet Synchronous Motor)을 비교하고 넘어가겠습니다.

BLDC 전동기와 PMSM을 구분 할 수 있는 가장 큰 특징이 역기전력 파형을 비교하는 것 입니다.

BLDC 전동기는 역기전력이 사다리꼴 형태로 나타나는 한편 PMSM은 역기전력이 사인파 형태로 나타 납니다.

	BLDC motor	PMSM
Back-EMF	Trapezoidal	Sinusoidal
Phase current	Quasi-square waveform	Sinusoidal
Driving method	- Two-phase excitation method - Rotor position is detected by hall sensors	- Torque control in vector control - High resolution position sensor is needed
Torque ripple	Moderate	Low
Cost	Low	High

BLDC 전동기와 PMSM의 비교

BLDC Motor(사전 자료 조사)

먼저 영구자석의 착자 방향에 따라 역기전력의 형태가 달라집니다. BLDC 전동기는 방사 형태로 균일한 자속이 발생하도록 착자 도어 있는 반면 PMSM은 평생으로 착자 되어 있습니다. 이는 상전류의 파형과 연관이 있습니다. BLDC 전동기의 경우 상전류가 준구형파인데 평평한 구간을 같은 평평한 파형으로 일치 시키기 위해서 역기전력은 사다리꼴 형태가 발생하게 자 한 것이고, PMSM의 경우 상전류가 사인파이기 때문에 이와 형태를 일치 시키기 위해 사인파 형태로 착자한 것 입니다.