실험 5. PWM 모터 속도제어

전자공학과 21611646 유준상

Ⅰ 실험 목적

➔ 위상정정 PWM을 이용한 DC 모터 속도 제어

[1] Up/Down 스위치에 대한 디바운스 기능을 타이머/카운터0 오버플로로 사용하여 만듦

[2] 위상정정 PWM 모드

UP/Down 스위치로 OCR0 값을 조절

OC0 핀에서 출력되는 PWM 파형을 발생

DC 모터 속도제어에 활용

Ⅱ 실험 도구 및 소자

➔ AVR Studio 4, 브레드 보드, ATmega128 보드, DC 어댑터, PWR B/D, 와이어 스트리퍼, 7-segment LED, 저항 330Ω 8개, 저항 470Ω 3개 피복 단선 0.6mm, JTAG 다운로더, PNP 트랜지스터 2개, NPN 트랜지스터 1개, 소형 기어드 DC 모터

Ⅲ 소스 코드

1. 고속 PWM을 이용한 DC 모터 속도제어

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

#define  F_CPU    16000000UL                          // CPU Clock 16MHz
#define  PRESCALE                  256L
```

```
#define  PULSE_PER_OVERFLOW    256L
#define  MS_OVERFLOW_CYCLE     ((double)(PULSE_PER_OVERFLOW * PRESCALE) ₩
                               /(double)((double)F_CPU/1000.0))

#define  OC0       PB4
#define  UP        PB7
#define  DOWN   PB6

#define  NUM_REQ                  2
#define  REQ_UP            0
#define  REQ_DOWN          1
#define  DEBOUNCE_CYCLE          50                              // Delay 값

volatile  unsigned          long      timer0;  // Overflow마다 1씩 증가될 변수
volatile  unsigned          int               number;
unsigned char led[] = {0x48, 0x7D, 0xC4, 0x64, 0x71, 0x62, 0x43, 0x7C, 0x40, 0x70};

volatile  unsigned          long      req[NUM_REQ] = {0, 0};
double   ms_ov_cycle;
// Timer/Counter0 Interrupt service routine
SIGNAL(SIG_OVERFLOW0) // 타이머/카운터0 인터럽트 서비스 루틴
{         int         i;
          timer0++;                                                  // Overflow마다 1
씩 증가

          // 오버플로 횟수가 짝수일 때 10자리, 홀수일 때 1자리 디스플레이
```

```
        PORTC = (timer0 % 2 == 0) ? led[(number % 100) / 10] : led[number%10];
        PORTD = (PORTD |0xC0) & ~(1<<((timer0 % 2 == 0) ? PD7 : PD6));

        for(i=0; i<NUM_REQ; i++)
                if( req[i] > 0)                                 // REQ 요청이 있을 때만
                req[i]--;                                       // 시간 지연 경과 응답
}

// ms_interval초 시간 지연을 위한 오버플로 횟수 계산 함수
unsigned long    ms_req_timer0(unsigned long ms_interval)
{       return ( ms_interval <= 0) ? 0 : ₩
                        + (unsigned long)(ms_interval / ms_ov_cycle);
                        // interval/cycle(=클럭 횟수) 반환
}

int     main(void)
{
        DDRC = 0xFF;                                            // 출력 지정
        DDRD |= 1<<PD7 | 1<<PD6;                        // 두 자리 7-segment LED를 켜기 위한 출력(자리수 선택)
        DDRB |= 1<<OC0;                                         // OC0=PB4(4) 출력
        DDRB &= ~(1<<UP | 1<<DOWN);                     // UP, DOWN 스위치 위치를 입력 방향으로
        PORTB |= 1<<UP | 1<<DOWN;                       // UP, DOWN 스위치 내부 풀업 저항
```

```
	TCCR0 = 1<<WGM00 | 1<<WGM01;                    // Fast PWM mode
	TCCR0 |= 1<<CS02 | 1<<CS01;          // 프리스케일러 CS02:00=(1,1,0) 256분주
	TCCR0 |= 1<<COM01;                              // 상승 중 OCR0와 일치하면 Clear, 하강 중 일치하면 Set

	TIMSK |= 1<<TOIE0;                              // TIMER/COUNTER0 INTERRUPT ENABLE

	timer0 = 0;
	sei();

	ms_ov_cycle      = MS_OVERFLOW_CYCLE;
	OCR0 = 0;
	number = OCR0*100/256;                  // OCR0 값과 number값을 일치

	while(1)                                        // 무한 loop req[REQ_UP]==0 은 DEBOUNCE_CYCLE 시간경과 요청 완료 검사
{
	  if( (req[REQ_UP]==0) && !(PINB & (1<<UP)))
		{ //req[UP]응답, UP 스위치 눌림 검사
		req[REQ_UP] = ms_req_timer0(DEBOUNCE_CYCLE); // delay 시간 재설정
		OCR0 = (OCR0 == 255) ? 255 : OCR0 + 1; // 최댓값은 255, UP 스위치가 눌리면 OCR0 증가
	   }

	  if( (req[REQ_DOWN]==0) && !(PINB & (1<<DOWN)))
		{ //req[DOWN]응답,DOWN 스위치 검사
```

```
                    req[REQ_DOWN] = ms_req_timer0(DEBOUNCE_CYCLE); // delay 시간 재설정
                     OCR0 = (OCR0 == 0) ? 0 : OCR0 - 1; // 최솟값은 0, DOWN 스위치가 눌리면
OCR0 감소
            }

              number = OCR0 * 100/256;
          }
          return 0;
}
```

2. 위상정정 PWM을 이용한 DC 모터 속도 제어

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

#define  F_CPU    16000000UL                                    // CPU Clock 16MHz
#define  PRESCALE                  256L
#define  PULSE_PER_OVERFLOW    510L
#define  MS_OVERFLOW_CYCLE     ((double)(PULSE_PER_OVERFLOW * PRESCALE) ₩
                                /(double)((double)F_CPU/1000.0))

#define  OC0       PB4
#define  UP        PB7
#define  DOWN   PB6

#define  NUM_REQ                   2
#define  REQ_UP            0
```

```
#define  REQ_DOWN       1
#define  DEBOUNCE_CYCLE          50                                      // Delay 값

volatile  unsigned          long      timer0;  // Overflow마다 1씩 증가될 변수
volatile  unsigned          int                 number;
unsigned char led[] = {0x48, 0x7D, 0xC4, 0x64, 0x71, 0x62, 0x43, 0x7C, 0x40, 0x70};

volatile  unsigned          long      req[NUM_REQ] = {0, 0};
double  ms_ov_cycle;
// Timer/Counter0 Interrupt service routine
SIGNAL(SIG_OVERFLOW0) // 타이머/카운터0 인터럽트 서비스 루틴
{         int       i;
          timer0++;                                                       //  Overflow마다  1
씩 증가

          // 오버플로 횟수가 짝수일 때 10자리, 홀수일 때 1자리 디스플레이
          PORTC = (timer0 % 2 == 0) ? led[(number % 100) / 10] : led[number%10];
          PORTD = (PORTD |0xC0) & ~(1<<((timer0 % 2 == 0) ? PD7 : PD6));

          for(i=0; i<NUM_REQ; i++)
                    if( req[i] > 0)                                       // REQ 요청이 있을 때만
                    req[i]--;                                             // 시간 지연 경과 응답
}

// ms_interval초 시간 지연을 위한 오버플로 횟수 계산 함수
unsigned long     ms_req_timer0(unsigned long ms_interval)
```

```
{       return ( ms_interval <= 0) ? 0 : ₩
                        + (unsigned long)(ms_interval / ms_ov_cycle);
                        // interval/cycle(=클럭 횟수) 반환
}

int     main(void)
{
        DDRC = 0xFF;                                    // 출력 지정
        DDRD |= 1<<PD7 | 1<<PD6;                        // 두 자리 7-segment LED를 켜기 위한 출력(자리수 선택)
        DDRB |= 1<<OC0;                                 // OC0=PB4(4) 출력
        DDRB &= ~(1<<UP | 1<<DOWN);                     // UP, DOWN 스위치 위치를 입력 방향으로
        PORTB |= 1<<UP | 1<<DOWN;                       // UP, DOWN 스위치 내부 풀업 저항

        TCCR0 = 1<<WGM00;                               // Phase Correct PWM mode
        TCCR0 |= 1<<CS02 | 1<<CS01;                     // 프리스케일러 CS02:00=(1,1,0) 256분주
        TCCR0 |= 1<<COM01;                              // 상승 중 OCR0와 일치하면 Clear, 하강 중 일치하면 Set

        TIMSK |= 1<<TOIE0;                              // TIMER/COUNTER0 INTERRUPT ENABLE

        timer0 = 0;
        sei();
```

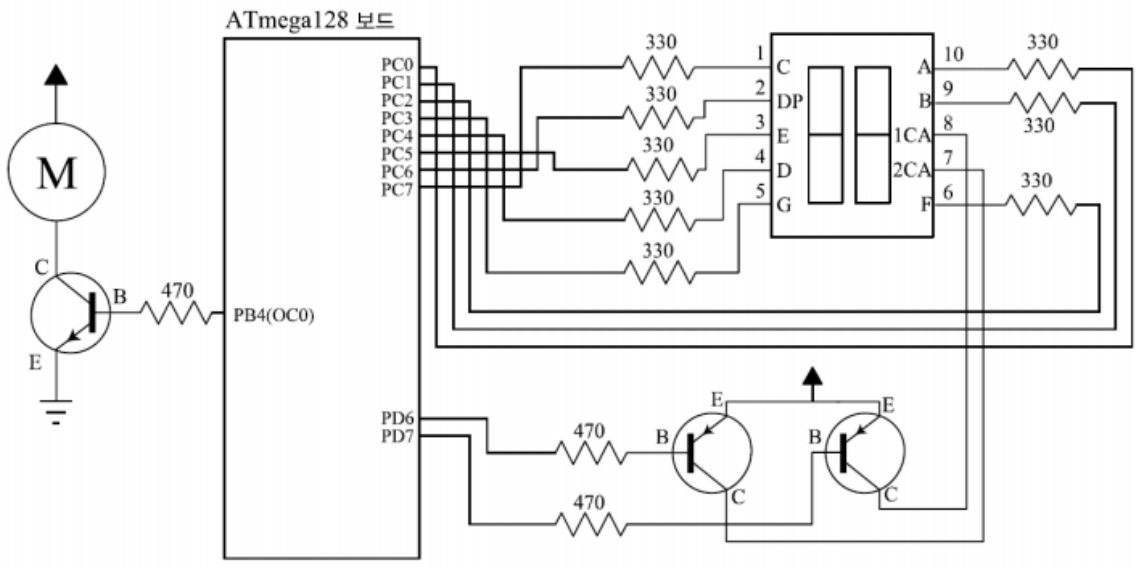
```
	ms_ov_cycle	= MS_OVERFLOW_CYCLE;
	OCR0 = 0;
	number = OCR0*100/256;				// OCR0 값과 number값을 일치

	while(1)					// 무한 loop
req[REQ_UP]==0 은 DEBOUNCE_CYCLE 시간경과 요청 완료 검사
{
	  if( (req[REQ_UP]==0) && !(PINB & (1<<UP)))
		{ //req[UP]응답, UP 스위치 눌림 검사
		req[REQ_UP] = ms_req_timer0(DEBOUNCE_CYCLE); // delay 시간 재설정
		OCR0 = (OCR0 == 255) ? 255 : OCR0 + 1; // 최댓값은 255, UP 스위치가 눌리
면 OCR0 증가
	  }

	  if( (req[REQ_DOWN]==0) && !(PINB & (1<<DOWN)))
		{ //req[DOWN]응답,DOWN 스위치 검사
		req[REQ_DOWN] = ms_req_timer0(DEBOUNCE_CYCLE); // delay 시간 재설정
		 OCR0 = (OCR0 == 0) ? 0 : OCR0 - 1; // 최솟값은 0, DOWN 스위치가 눌리면
OCR0 감소
	  }

	  number = OCR0 * 100/256;
	}
	return 0;
}
```

Ⅳ 사전 지식

➔ 실험 4의 기본 회로 구성을 사용하고 추가한다. 따라서 기본 회로 구성에 대한 설명은 하지 않고 추가한 부분에 대해서만 설명한다.

1. 고속 PWM을 이용한 DC 모터 속도제어

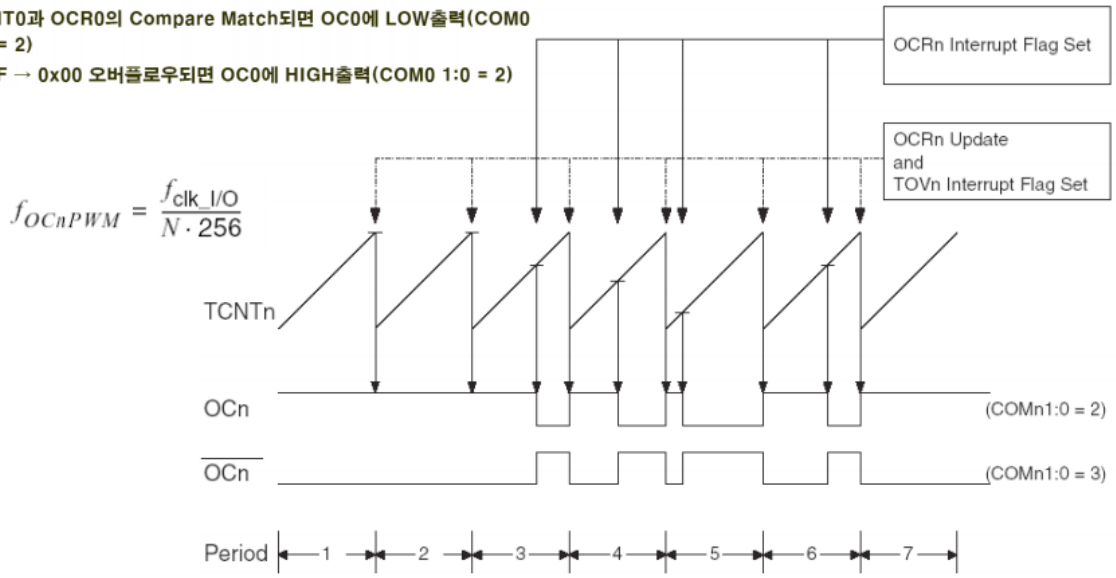
– 실험 회로



기본 회로 구성에서 PB4와 저항 470Ω과 NPN 트랜지스터, DC 모터를 연결한다. 여기서 트랜지스터는 에미터와 그라운드를 연결하는 Down side 방식으로 사용한다.

Fast PWM Mode

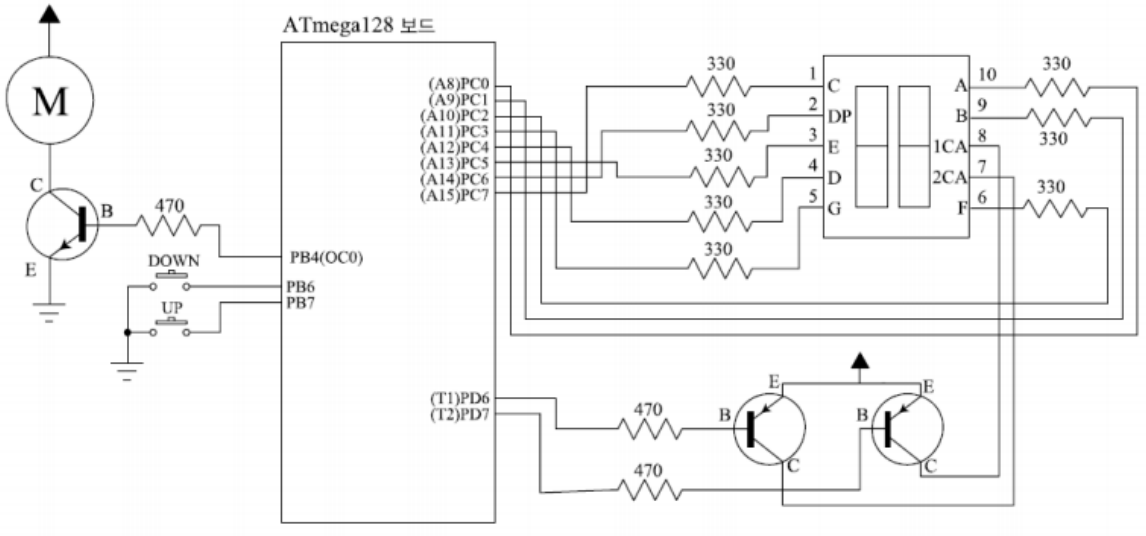
- 높은 주파수 PWM 파형발생이 필요할 때 사용
- 상향카운터 (Single-Slope Operation)
- 0x00 ~ 0xFF 계수 동작 반복
- TCNT0과 OCR0의 Compare Match되면 OC0에 LOW출력(COM0 1:0 = 2)
- 0xFF → 0x00 오버플로우되면 OC0에 HIGH출력(COM0 1:0 = 2)



0x00~0xFF까지 반복하다가, OCR0 값과 TCNT0값이 같아지면 OC0에 Low가 출력된다. 그리고 0xFF에서 0x00이 되면 OC0에 High가 출력된다.

2. 위상정정 PWM을 이용한 DC 모터 속도 제어

- 실험 회로



- **DOWN 스위치로 OCR0값 감소시킴**
- **타이머/카운터0 오버플로를 사용하여 스위치 디바운싱 기능 적용**

Fast PWM 회로 구성에서 PB6와 PB7에 스위치를 각각 연결하여 구성한다. 각 스위치는 OCR0값

을 증가/감소 시키는 기능을 한다.

[회로 구동]

- NPN 트랜지스터로 모터에 흐르는 전류 간접 구동

- OCR0 값을 2자리 7-segmnet LED에 비례하여 0~99로 디스플레이

- 7-segment LED 디스플레이를 위한 트랜지스터 설정 비트를 출력 방향

- OC0 (PB4) 핀을 출력 방향으로 설정

- 파형 발생모드를 WGM00 / WGM01 비트를 조절해 Fast PWM 모드와 Phase Correct PWM 모드로 설정

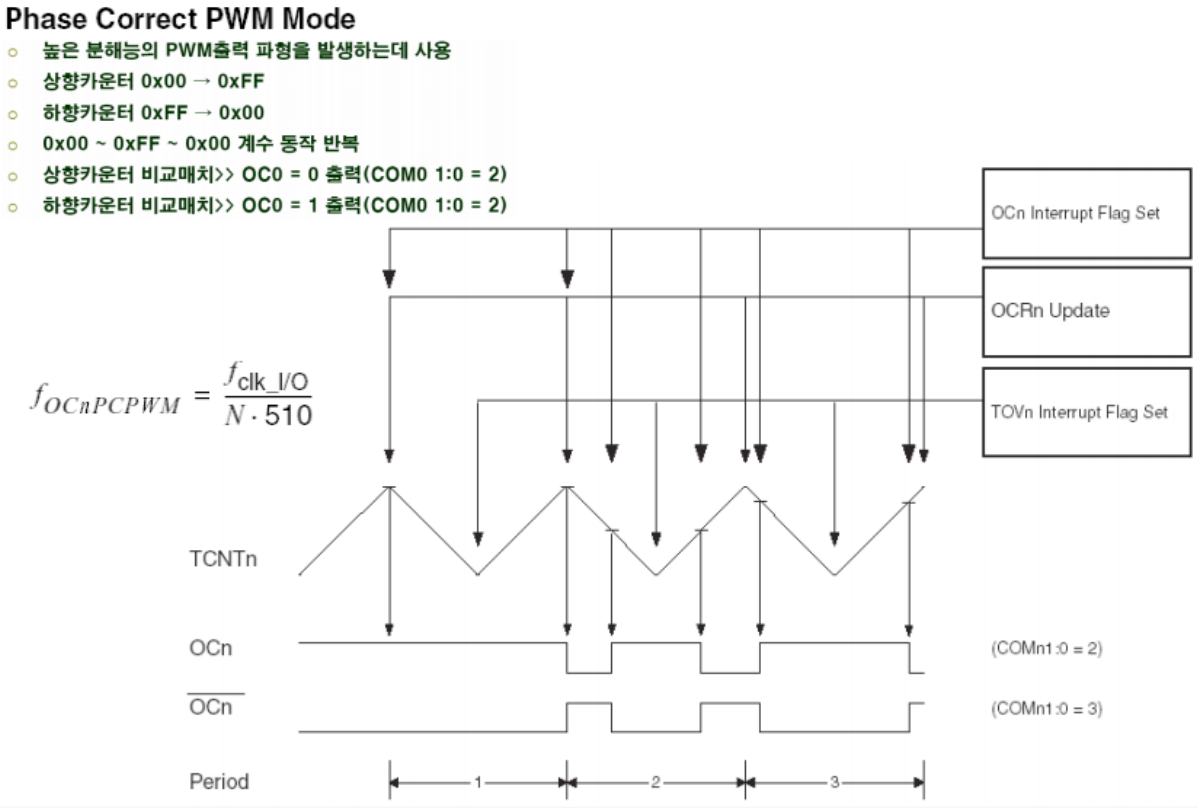
- 프리스케일러를 내부 CPU 클럭의 256 분주로 설정

- OC0 출력 극성을 TCNT0 값이 OCR0 값과 일치하면 0이 되는 모드로 설정

- 타이머 오버플로 인터럽트를 활성화

- OCR0값을 바꾸면서 모터의 속도 변화를 관찰

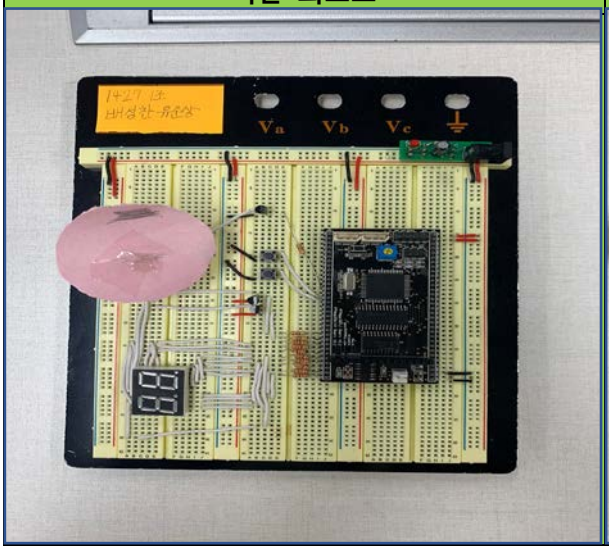
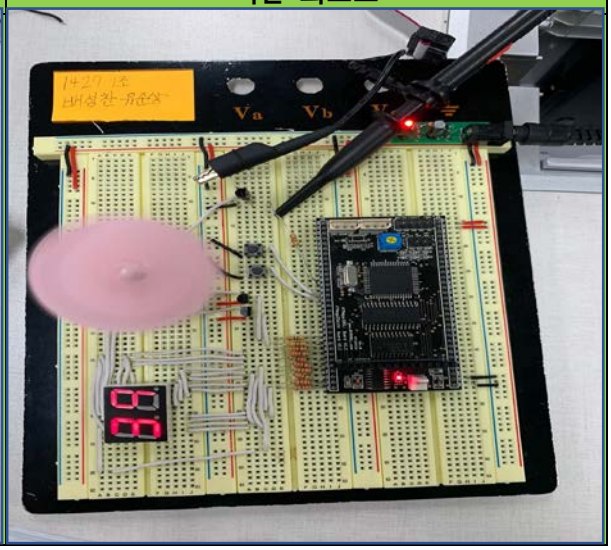
- 7-segment LED에 디스플레이되는 number에 OCR0값을 디스플레이



다른 모드들이 0x00~0xff -> 0x00으로 반복되는 것과 다르게 위상 정정 PWM 모드는 0xFF이후 바로 0x00으로 떨어지지 않고 상승과 반대로 하강이 일어난다. 따라서 주파수 계산 시에 0과 최댓값(255)일 때를 제외한 510을 한 주기 펄스로 쳐서 계산한다. 또 위의 이미지를 보다시피 OCR 값이 커지면 Pulse 폭도 커지는 것을 볼 수 있다.

V 회로도 및 7-segment 작동

➔ 위의 회로와 동일하게 연결하여 회로를 구성하였고 디스플레이 하길 원한 숫자 '12'와 '34'가 정상적으로 디스플레이 됨을 확인할 수 있다.

기본 회로도	기본 회로도
	

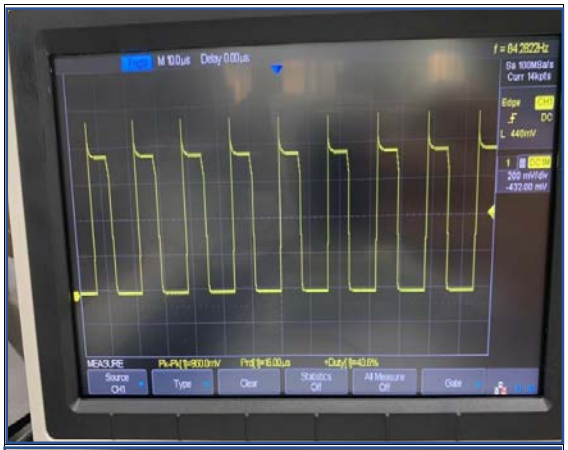
※ 주기 확인은 프리스케일러 #define PRESCALE 256L 부분과 TCCR0 |= 1<<CS02 | 1<<CS01; 부분을 1~1024까지 조정했다. 모든 프리스케일러 조정 후 OCR0 = 100이고 듀티비가 39일 때를 기준으로 잡았다. 주기 계산은

$\frac{prescaler*256}{f}(Fast\ PWM\ mode), \frac{prescaler*510}{f}(Phase\ Correct\ PWM\ mode)$ 으로 한다.

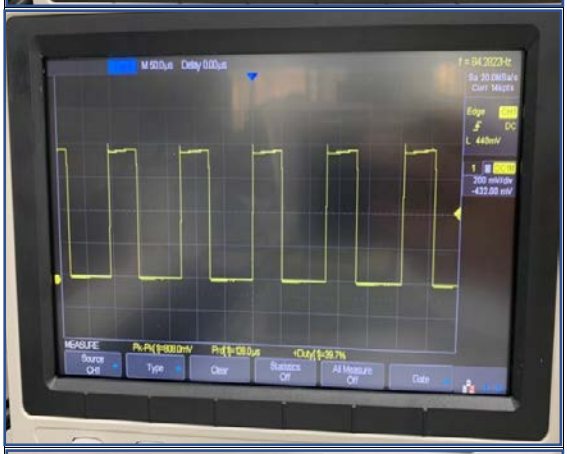
듀티비 확인은 스위츠를 이용하여 OCR0값을 0에서부터 50 단위로 255까지 높여가며 number = OCR0*100/256;를 통해 계산한 이론값과 오실로스코프를 통해 얻은 실험값을 비교하였다.

★ Fast PWM 모드

주기 확인 (OCR0 : 100 -> 듀티비 : 39로 설정)



- Prescaler : 1
- 이론값 : $\frac{256 \times 1}{16MHz} = 16\mu s$
- 실험값 : $16\mu s$



- Prescaler : 8
- 이론값 : $\frac{256 \times 8}{16MHz} = 128\mu s$
- 실험값 : $128\mu s$

- Prescaler : 32
- 이론값 : $\frac{256 \times 32}{16MHz} = 512\mu s$
- 실험값 : $512\mu s$

- Prescaler : 64
- 이론값 : $\frac{256 \times 64}{16MHz} = 1024\mu s$
- 실험값 : $1.02ms$

- Prescaler : 128

- 이론값 : $\frac{256 \times 128}{16MHz} = 2048\mu s$

📁 실험값 : $2.05ms$

- Prescaler : 256

- 이론값 : $\frac{256 \times 256}{16MHz} = 4096\mu s$

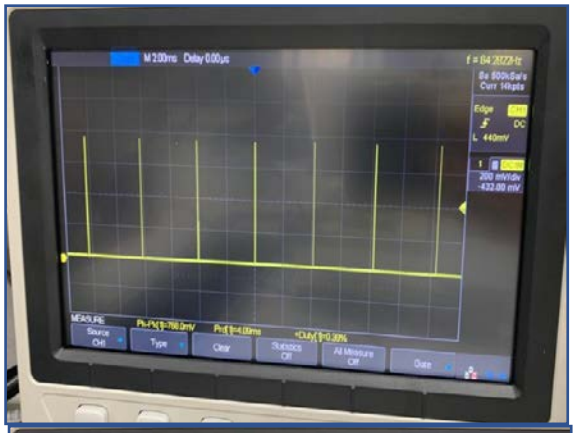
📁 실험값 : 16.

- Prescaler : 1024

- 이론값 : $\frac{256 \times 1024}{16MHz} = 16384\mu s$

📁 실험값 : $16.38ms$

듀티비 확인



- OCR0 : 0

- 이론값 : $\frac{0+1}{256} \times 100 = 0.39\%$

- 실험값 : 0.39%

- OCR0 : 50

- 이론값 : $\frac{50+1}{256} \times 100 = 19.92\%$

- 실험값 : 19.5%

- OCR0 : 100

- 이론값 : $\frac{100+1}{256} \times 100 = 39.45\%$

- 실험값 : 39.8%

- OCR0 : 150

- 이론값 : $\frac{150+1}{256} \times 100 = 58.98\%$

- 실험값 : 58.9%

- OCR0 : 200

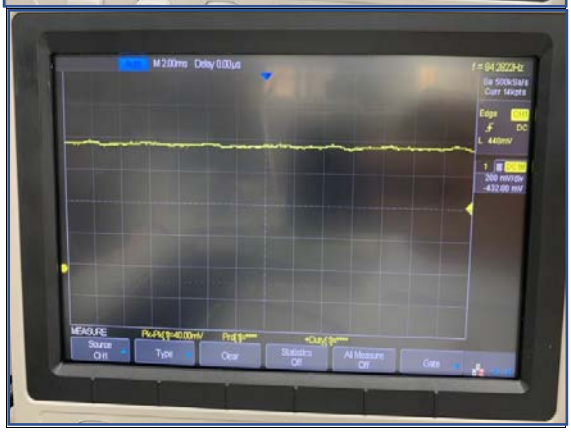
- 이론값 : $\frac{200+1}{256} \times 100 = 78.52\%$

- 실험값 : 78.9%

- OCR0 : 250

- 이론값 : $\frac{250+1}{256} \times 100 = 98.05\%$

- 실험값 : 98.8%

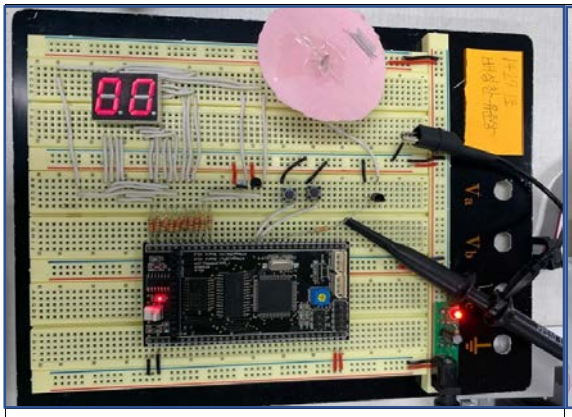


- OCR0 : 255

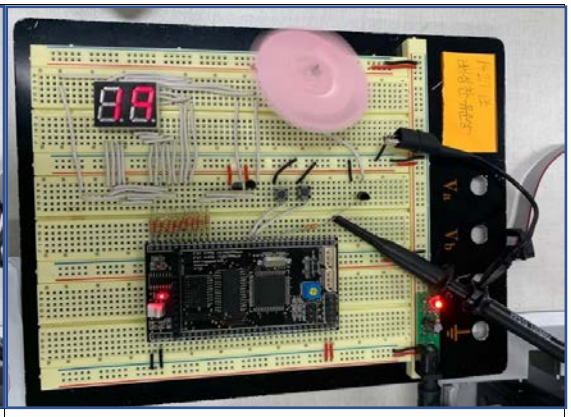
- 이론값 : $\frac{256+1}{256} \times 100 = 100\%$

- 실험값 : 측정 x

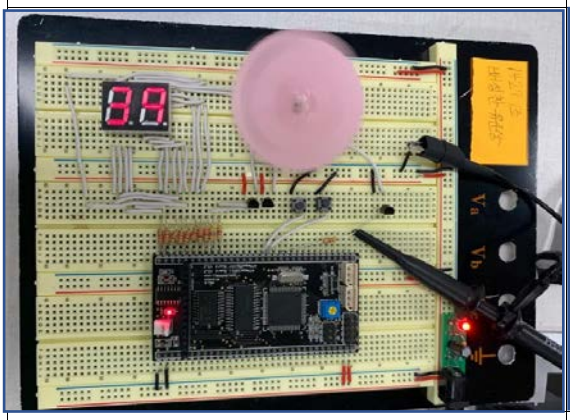
➔ 이론값과 실험값을 비교해보면 거의 같다. 하지만 약간의 오차가 발생했다. 그 이유로는 오실로스코프를 통해 측정한 실험값은 소수점 첫째 자리까지 계산했다는 점과 실제 실험에서 사용한 저항과 전압값이 정확한 값이 아니기 때문에 오차가 발생했다고 판단했다.



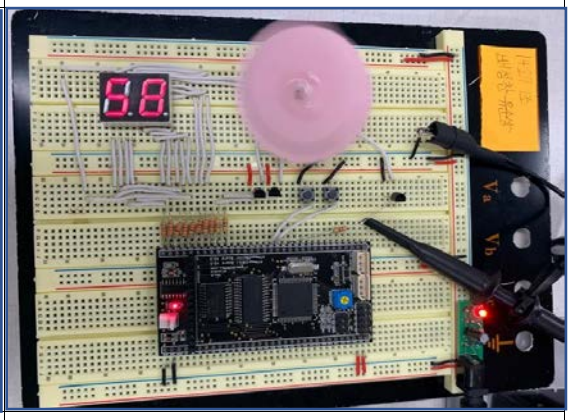
OCR0 - 0, 듀티비 - 0 출력



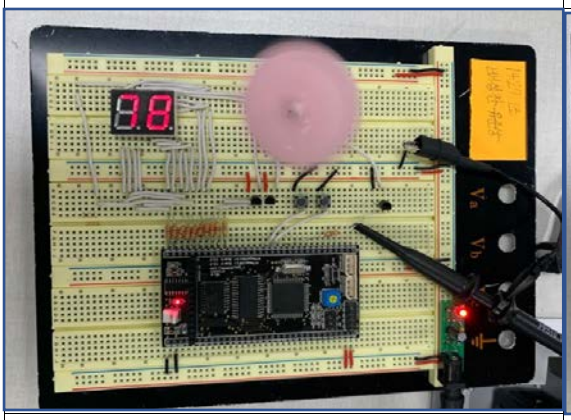
OCR0 - 50, 듀티비 - 19 출력



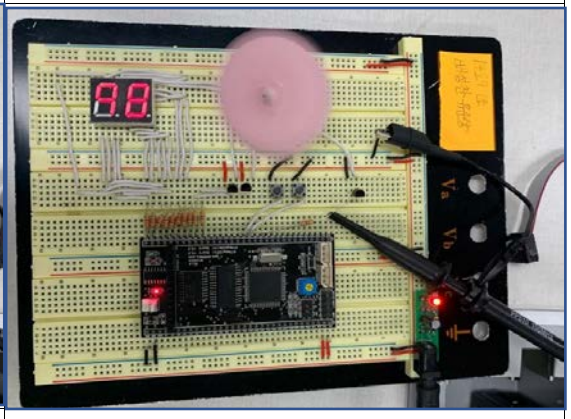
OCR0 - 100, 듀티비 - 39 출력



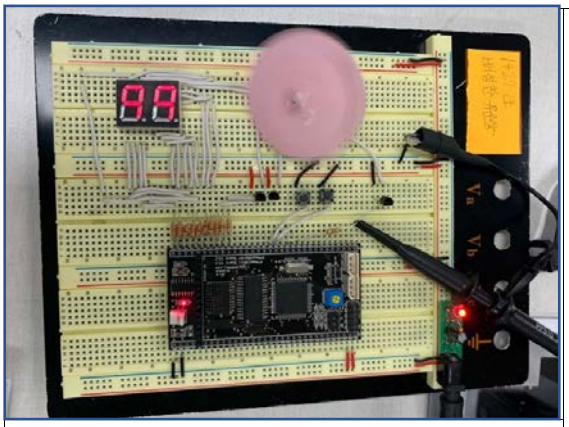
OCR0 - 150, 듀티비 - 58 출력



OCR0 - 200, 듀티비 - 78 출력



OCR0 - 250, 듀티비 - 98 출력



OCR0 - 255, 듀티비 - 99 출력

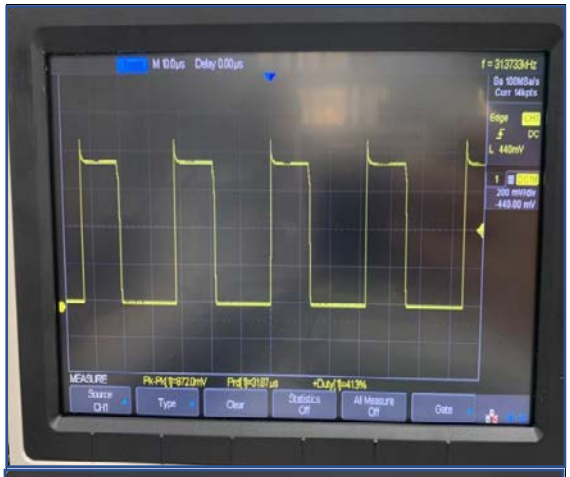
OCR0 값을 0~255 범위 중 50단위로 증가했을 때 7-segment LED에 출력되는 듀티비이다.

실험값과 이론값과 동일하게 LED에 출력되는 것을 볼 수 있다.

OCR0가 255일 때, 듀티비는 100이어야 하지만, 7-segment가 10의자리 까지만 출력하므로 99로 출력된다.

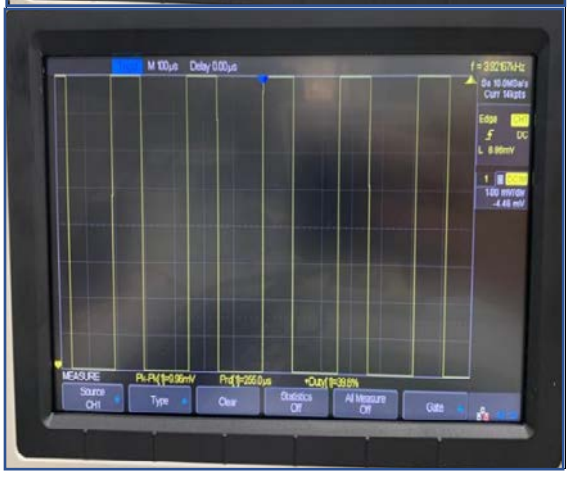
★ Phase Correct PWM 모드

주기 확인



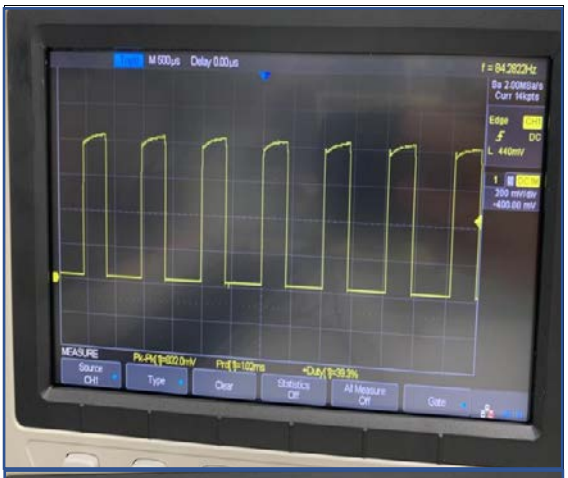
- Prescaler : 1
- 이론값 : $\frac{510 \times 1}{16MHz} = 31.88\mu s$

☞ 실험값 : $31.78\mu s$



- Prescaler : 8
- 이론값 : $\frac{510 \times 8}{16MHz} = 255\mu s$

☞ 실험값 : $255.0\mu s$



- Prescaler : 32

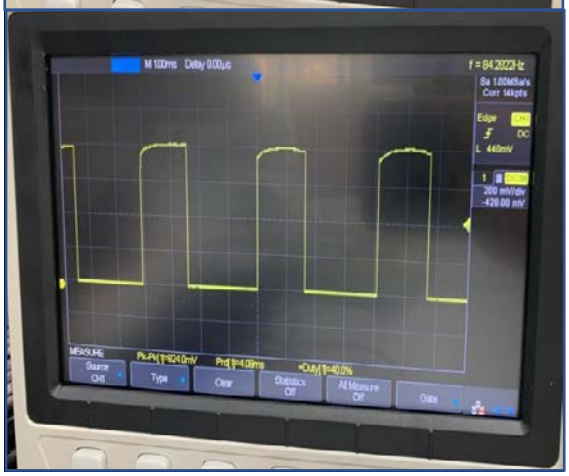
- 이론값 : $\frac{510\times32}{16MHz}=1020\mu s$

- 실험값 : $1.02ms$

- Prescaler : 64

- 이론값 : $\frac{510\times64}{16MHz}=2040\mu s$

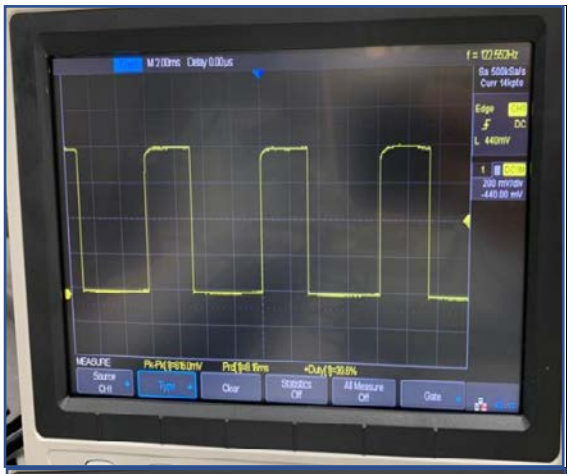
- 실험값 : $2.04ms$



- Prescaler : 128

- 이론값 : $\frac{510\times128}{16MHz}=4080\mu s$

- 실험값 : $4.08ms$



- Prescaler : 256

- 이론값 : $\frac{510 \times 256}{16MHz} = 8160\mu s$

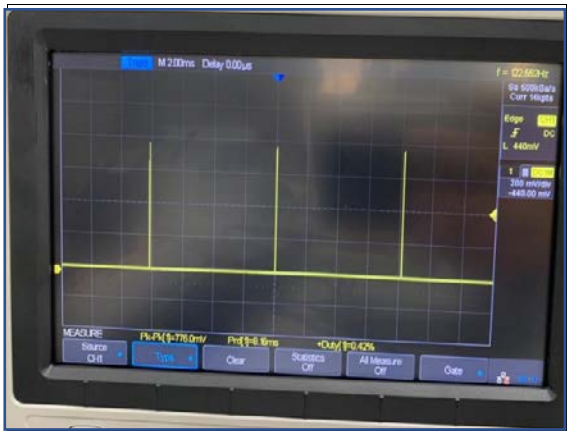
실험값 : $8.16ms$

- Prescaler : 1024

- 이론값 : $\frac{510 \times 1024}{16MHz} = 32640\mu s$

실험값 : $32.63ms$

듀티비 확인



- OCR0 : 0

- 이론값 : $\frac{0 \times 2}{510} \times 100 = 0\%$

- 실험값 : 0.42%

- OCR0 : 50

- 이론값 : $\frac{50\times 2}{510}\times 100 = 19.61\%$

- 실험값 : 19.2%

- OCR0 : 100

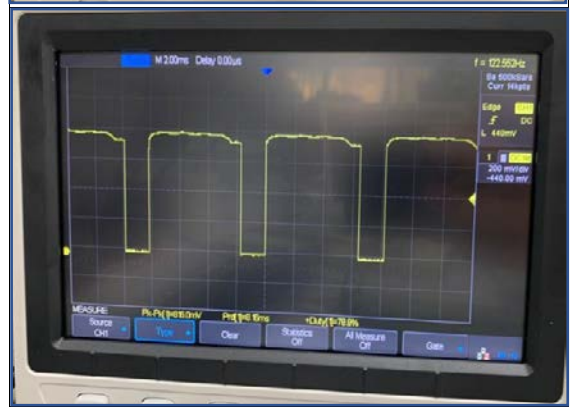
- 이론값 : $\frac{100\times 2}{510}\times 100 = 39.22\%$

- 실험값 : 39.6%

- OCR0 : 150

- 이론값 : $\frac{150\times 2}{510}\times 100 = 58.82\%$

- 실험값 : 58.8%



- OCR0 : 200

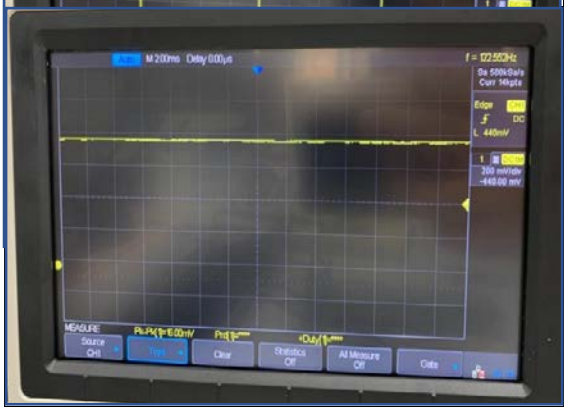
- 이론값 : $\frac{200\times 2}{510}\times 100 = 78.43\%$

- 실험값 : 78.8%

- OCR0 : 250

- 이론값 : $\frac{250 \times 2}{510} \times 100 = 98.04\%$

- 실험값 : 98.8%

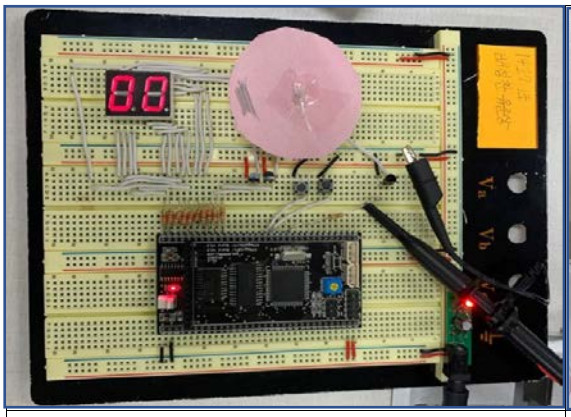


- OCR0 : 255

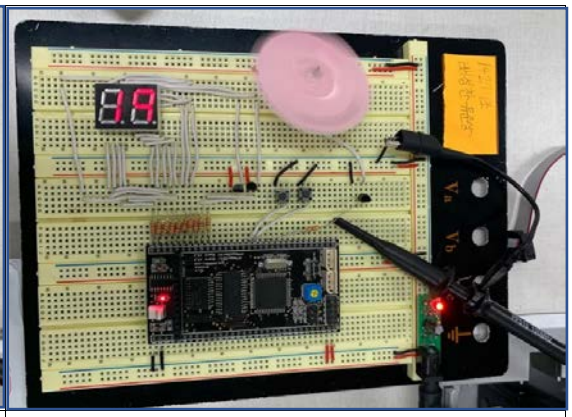
- 이론값 : $\frac{255 \times 2}{510} \times 100 = 100\%$

- 실험값 : 측정x

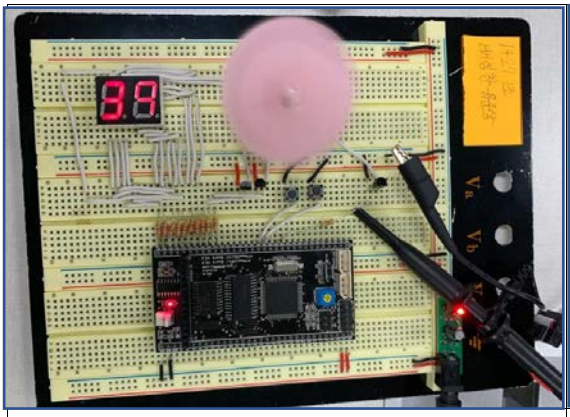
➔ 이론값과 실험값을 비교해보면 거의 같다. 하지만 약간의 오차가 발생했다. 그 이유로는 오실로스코프를 통해 측정한 실험값은 소수점 첫째 자리까지 계산했다는 점과 실제 실험에서 사용한 저항과 전압값이 정확한 값이 아니기 때문에 오차가 발생했다고 판단했다.



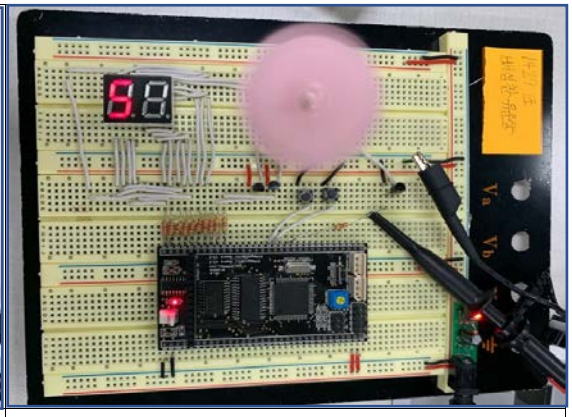
OCR0 - 0, 듀티비 - 0 출력



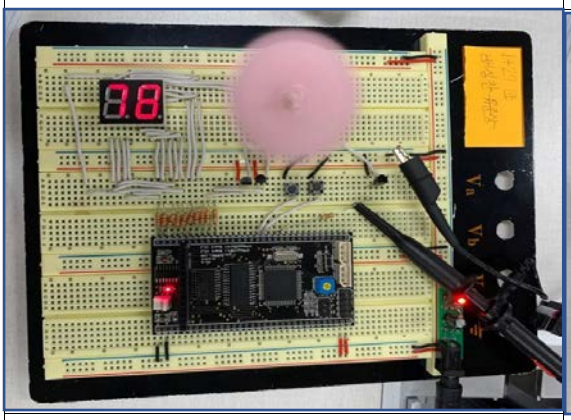
OCR0 - 50, 듀티비 - 19 출력



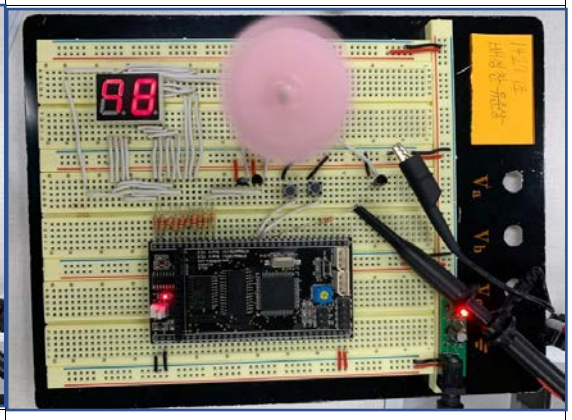
OCR0 - 100, 듀티비 - 39 출력



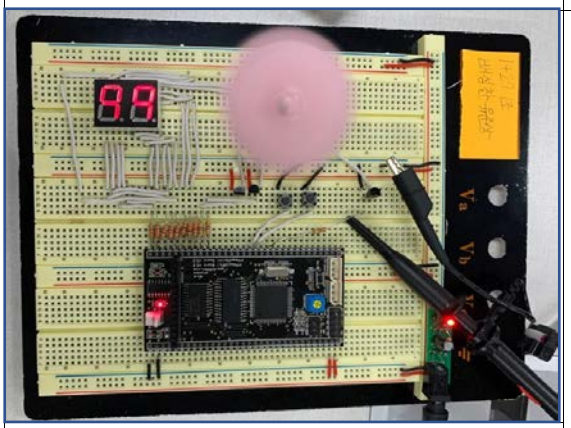
OCR0 - 150, 듀티비 - 58 출력



OCR0 - 200, 듀티비 - 78 출력



OCR0 - 250, 듀티비 - 98 출력



OCR0 - 255, 듀티비 - 99 출력

OCR0 값을 0~255 범위 중 50단위로 증가했을 때 7-segment LED에 출력되는 듀티비이다.

실험값과 이론값과 동일하게 LED에 출력되는 것을 볼 수 있다.

OCR0가 255일 때, 듀티비는 100이어야 하지만, 7-segment가 10의자리 까지만 출력하므로 99로 출력된다.

V 결과 및 토의

➔ 이번 실험에서는 Fast PWM mode와 Phase Correct PWM mode를 비교해 보았다. 각각 프리스케일 조정을 이용한 주기의 변화와 OCR0값의 조정으로 듀티비의 변화를 관찰했다. 실험 결과는 아래와 같다.

★ Fast PWM 모드와 Phase Correct PWM 모드 비교

➔ 두 모드 모두 프리스케일을 높이면 주기도 높아진다. 듀티비는 $\frac{OCR0+1}{256} * 100\ (Fast\ PWM\ mode), \frac{OCR0*2}{510} * 100(Phase\ Correct\ PWM\ mode)$로 각각 계산하는 데, 두 수식의 결과 값이 거의 같다.

두 모드의 차이점은 Fast PWM 모드는 0->255 상승 후 0으로 clear된 후 다시 주기가 시작되는 반면, Phase Correct PWM 모드는 0->255->0의 주기를 가지므로 Fast PWM 모드보다 약 2배 긴 주기를 가진다.

또, OCR0가 0일 때의 듀티비가 0(%)인지와 최댓값인 255일 때 100(%)인지에 대해서도 실험해보았는데 이론값은 그러하지만 실제 실험값은 두 모드 모두 다르거나 측정 불가(숫자가 아닌 **이 디스플레이 됨)함을 확인했다. 그 이유는 듀티비 계산 시의 분자에서 +1 해주는 것 때문에 0이 나오지 않는 점과 오실로스코프의 파형에서 볼 수 있듯이 다 붙어서 주기를 계산할 수 없어서라고 판단했다.