실험 4. 착시 이용한 7-seg LED 디스플레이

전자공학과 21611646 유준상

Ⅰ 실험 목적

➔ [1] 타이머/카운터0 오버플로 인터럽트 사용

[2] 두 곳을 빠른 시간차로 번갈아 가며 디스플레이

착시 현상으로 두 자리 숫자를 디스플레이 – 디지털 I/O 핀 수를 감소 효과

[3] 트랜지스터를 이용한 간접 구동으로 7-세그먼트 LED에 필요한 전류를 공급

[4] 너무 잦은 오버플로 인터럽트가 발생하면, 주 프로그램의 동작이 정지될 수 있음을 확인

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 2개 피복 단선 0.6mm, JTAG 다운로더, PNP 트랜지스터 2개

Ⅲ 소스 코드

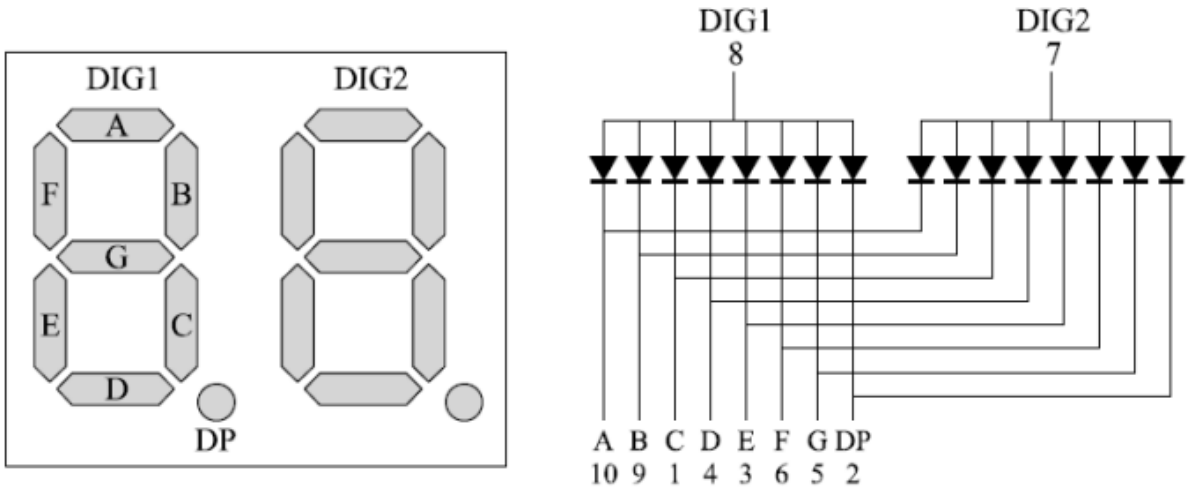
```
#include <avr/io.h>

#include <avr/interrupt.h>


volatile   unsigned long timer0;      // Overflow마다 1씩 증가될 변수
```

```
volatile   unsigned int number;       // 증가되어 7-세그먼트 LED에 디스플레이될 숫자
unsigned           char led[]={0x48, 0x7D,0xC4, 0x64, 0x71, 0x62, 0x43, 0x7C, 0x40, 0x70};

// 타이머/카운터0 인터럽트 서비스 루틴
ISR(TIMER0_OVF_vect)
{
        timer0++;          // Overflow마다 1씩 증가
        // Overflow count가 4의 배수일 때 10자리 또는 2의 배수일 때 1자리 디스플레이
        if(timer0%2==0){
                PORTC=(timer0%4==0) ? led[(number%100) / 10] : led[number%10];
                PORTD=(PORTD|0xC0)&~(1<<((timer0%4==0) ? PD7:PD6));
        }
}
int main(void)
{
        DDRC=0xFF;
        DDRD |= 1<<PD7 | 1<<PD6;

        TCCR0 = 1<<CS02 | 1<<CS01;
        TIMSK |=1<<TOIE0;

        timer0=0;
        sei();

        number=12;
```

```
        while(1);

        return 0;
}
```

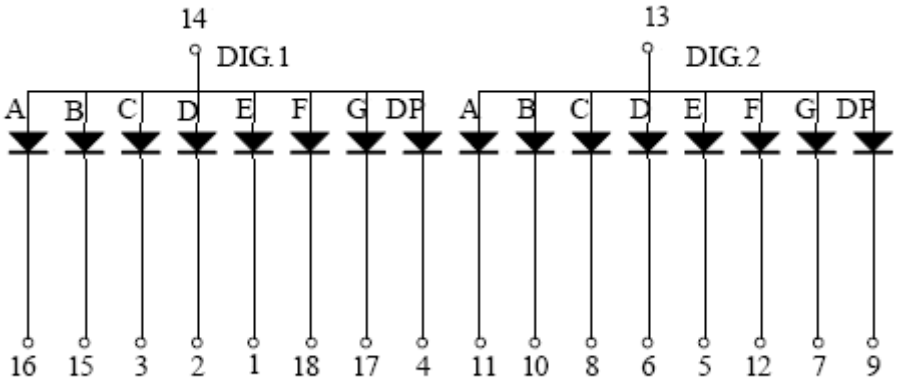
Ⅳ 사전 지식

➔ 브레드 보드에 PWR B/D를 연결 후 각 GND와 VCC를 편하게 쓰기위해 위와 같이 색에 구분을 둬서 연결했다. 검은선은 GND, 붉은선은 VCC를 의미한다. ATmega128의 PC0~PC7에 330Ω을 연결하고 각각 7-세그먼트에 해당하는 위치와 연결 후 사용했다.

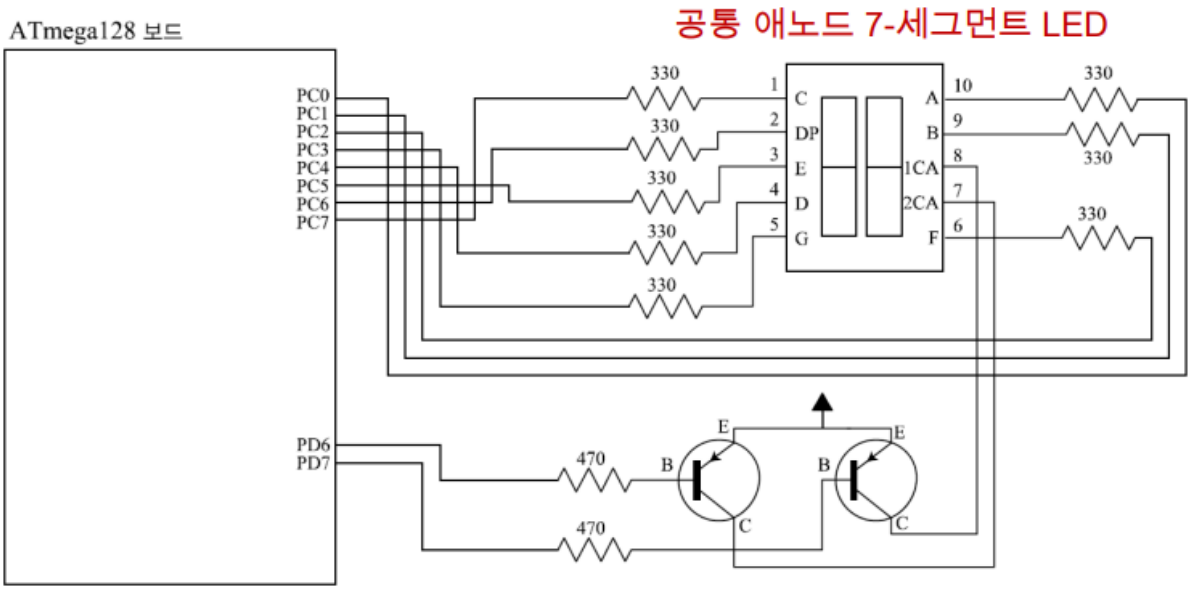


SR-2156A 공통 애노드 핀 구성

강의 자료에서는 SR-2156A 공통 애노드 7-segment를 사용하였지만, 본 실험에서는 5263asr1을 사용하였다. 따라서 7-segment 내부에 각 LED를 켜는 핀들이 연결되어 있지 않기 때문에 회로적으로 연결하여 사용했다.



위의 사진은 실험에서 사용한 7-세그먼트 5263ASR1에 해당하는 회로도이다. 본 실험에서는 DIG1, DIG2에 해당하는 번호로 연결 후 사용했다.



ATmega128의 각 포트와 7-segment의 해당하는 위치와 저항을 직렬 연결하여 사용했다.

7-세그먼트 LED의 내부 LED와 ATmega128 PORT C의 핀과의 관계이다.

디스플레이 숫자		0	1	2	3	4	5	6	7	8	9
C포트 핀	LED										
PC7	C	0	0	1	0	0	0	0	0	0	0
PC6	DP	1	1	1	1	1	1	1	1	1	1
PC5	E	0	1	0	1	1	1	0	1	0	1
PC4	D	0	1	0	0	1	0	0	1	0	1
PC3	G	1	1	0	0	0	0	0	1	0	0
PC2	F	0	1	1	1	0	0	0	1	0	0
PC1	B	0	0	0	0	0	1	1	0	0	0
PC0	A	0	1	0	0	1	0	1	0	0	0
16진값		48	7D	C4	64	71	62	43	7C	40	70

포트 출력값에 해당하는 16진수값을 먼저 계산 후 코딩에 적용하여 사용했다.

★ 두 자리 7-segment LED 디스플레이

➔ 병렬 신호출력 디스플레이 방법

- 16개의 디지털 I/O 핀 사용

- 10의 자리 8핀, 1의 자리 8핀

: static 방식이고, 16개의 I/O port가 필요

➔ 착시를 이용한 디스플레이 방법

- 10개의 디지털 I/O 핀 사용

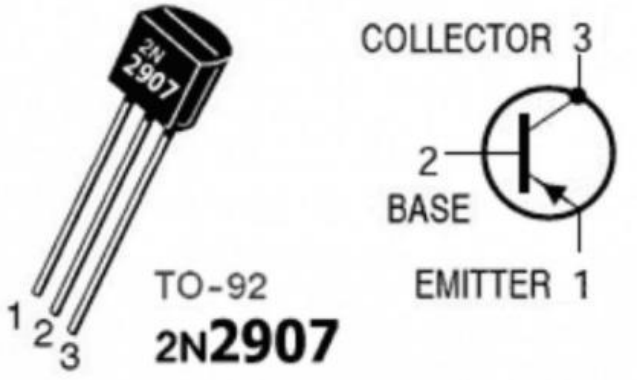
-- 숫자 데이터 : 8핀 (PC0~PC7)

-- 10의 자리, 1의 자리 전류 경로 선택 : 2핀 (PD6, PD7)

: Dynamic 방식이고, 10개의 I/O port 필요

➔ Dynamic 방식을 사용하는 것이 총 6개의 I/O port를 덜 쓰기 때문에 경제적이라고 볼 수 있고 본 실험에서도 Dynamic 방식을 사용하였다.

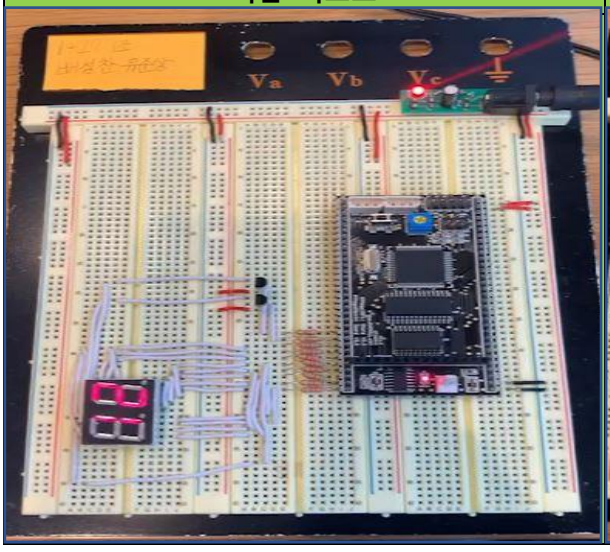
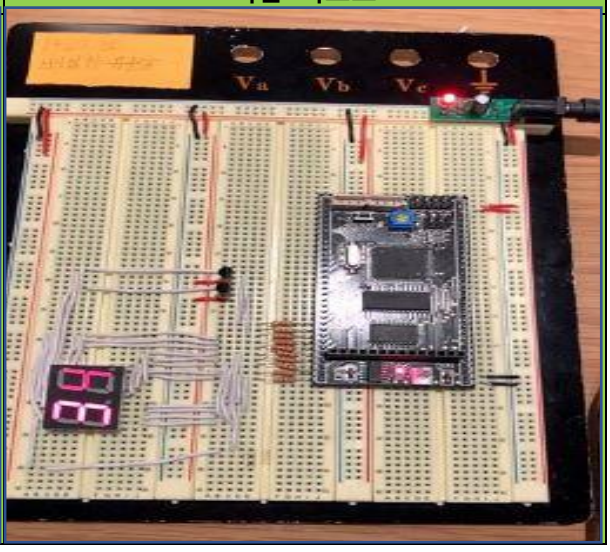
본 실험에서는 PNP 트랜지스터 2개를 사용한다. 7-segment에 vcc를 직접 연결하여 사용하지 않고, 트랜지스터를 이용한 간접 구동으로 7-segment LED에 필요한 전류를 공급하기 위해 사용된다. 또, 각각 10의 자리, 1의 자리 디스플레이 자리 선택을 위해 스위치의 역할도 한다. 10의 자리를 위해 PD7과 연결하여 사용하고, 1의 자리를 위해 PD6과 연결하여 사용한다.



본 실험에서 사용한 2개의 트랜지스터는 2n2907a이다. 트랜지스터의 헤드에서 평평한 면과 볼록한 면을 가지고 있다. 평평한 면을 기준으로 1(에미터), 2(베이스), 3(콜렉터)을 의미한다. ATmega128의 Port D와 저항 470Ω, 베이스를 연결하고, 에미터에 Vcc, 콜렉터에 다이오드와 저항 330Ω을 Port C와 연결하여 사용했다. Port D를 통해 Low 레벨이 나가면 에미터와 콜렉터가 붙어서 스위치 역할을 하는 PNP 트랜지스터가 ON되는 기능을 할 수 있다. 또, 에미터와 Vcc가 연결되는 것을 up side라고 부른다.

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
	

★ 딜레이 시간에 따른 다이나믹 디스플레이의 효과

딜레이 시간에 따른 다이나믹 디스플레이의 효과를 살펴보기 위해 세 가지로 전략을 미리 세우고 가능성과 결과 예상에 대해 먼저 토의했다. 토의 결과는 아래와 같다.

1. 실험 3과 같이 딜레이 함수(_delay_ms( ))를 LED 디스플레이 하는 루틴 내에 넣기

➔ 직관적이고 가장 먼저 든 기초적인 생각이었지만 본 실험의 목적인 착시 효과를 이용한 것과는 거리가 있는 것 같아서 수행하지 않음. 토의 결과는 아래와 같다.

2. 분주 프리스케일러 조정하기

➔ 큰 값으로 분주를 하면 타이머에 공급하는 입력 클럭의 속도를 느리게 할 수 있으므로 수행해 볼만 함.

3. Overflow count 조정

➔ 인터럽트 루틴 내에 LED 디스플레이할 주기를 기존의 2의 배수일 때 디스플레이 하는 것에서 2의 배수가 아닌 더 큰 숫자로 조정하면, 각 자리마다 디스플레이 하는 주기가 길어지므로 수행해볼만 함

4. 2, 3번을 같이 적용

➔ 2, 3번으로 딜레이 효과를 가질 수 있다면 두 방법을 한번에 적용한 것으로 최대의 딜레이를 줄 수 있을 것 같다고 생각하여 수행함

토의 후 2, 3, 4번을 수행해 보았다. 결과부터 말하자면 성공이었다.

2. 분주 프리스케일러 조정하기

Timer/Counter control register인 TCCR0의 0, 1, 2 bit인 CS02, CS01, CS00 BIT의 값을 이용하면 프리스케일러를 조정할 수 있다.

8-bit Timer/Counter Register Description

Timer/Counter Control Register – TCCR0

- **BIT 7 : FOC0 (Force Output Compare)**
  - WGM 비트가 non-PWM 모드로 설정되어 있을 때 FOC0 비트는 활성화된다. 그러나 AVR 의 상위 기종과 호
  - 환성을 유지하기 위해서 PWM 모드로 동작할 때는 "0"으로 설정되어야만 한다. 반면에 "1"로 설정하면 Waveform Gerneration Unit으로 동작한다.

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

Table 56. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	clk_{T0S}/(No prescaling)
0	1	0	clk_{T0S}/8 (From prescaler)
0	1	1	clk_{T0S}/32 (From prescaler)
1	0	0	clk_{T0S}/64 (From prescaler)
1	0	1	clk_{T0S}/128 (From prescaler)
1	1	0	clk_{T0S}/256 (From prescaler)
1	1	1	clk_{T0S}/1024 (From prescaler)

본 실험에서는 8분주(CS02:0, CS01:1, CS00:0)부터 1024분주(CS02:1, CS01:1, CS00:1)까지 값을 바꿔가며 수행했다.

본 실험의 기본 코드에서 아래의 부분을 수정하여 수행하면 된다.

```
TCCR0 = 1<<CS02 | 1<<CS01;
```

프리스케일의 조정에 따른 디스플레이 주기가 확연히 차이났지만 이미지로 그 차이를 나타내기는 힘드므로 동영상 촬영한 것으로 대체한다.

LED 점멸 주기 계산

$$T_{overflow} = \frac{\text{오버플로 주기 펄수 수} \times \text{프리스케일}}{\text{클록 소스 주파수}}$$

이론적으로는 LED 점멸 주기 계산 시에 프리스케일 값의 변화에 따른 주기의 변화로 증명할 수 있다.

아래와 같이 수정하며 실험했다.

프리스케일	8	32	256	1024

차이가 많이 나는 8분주와 1024분주 일 때의 결과를 비교한다.

1. 프리스케일 $= 8 \rightarrow \mathrm{T} = \frac{256*8}{16000000} = 0.000128\ s$

2. 프리스케일 $= 1024 \rightarrow \mathrm{T} = \frac{256*1024}{16000000} = 0.016384\ s$

1024 분주를 하게되면 8분주를 했을 때보다 128배 정도 LED 점멸 주기가 오래 걸린다. 이러한 결과로 보아 프리스케일러를 조정하면 오버플로 인터럽트가 발생의 빈도 수 자체를 조정할 수 있으므로 딜레이 시간에 따른 다이나믹 디스플레이 효과를 확인할 수 있다.

3. Overflow count 조정

오버플로 인터럽트 서비스 루틴이 실행되었을 때, 각 1의 자리와 10의 자리가 디스플레이 될 때 ON/OFF 유지하는 시간을 늘려서 딜레이 효과를 얻을 수 있는 방법을 생각하였고 Overflow count를 이용하여 수행했다. 착시를 이용한 디스플레이는 각 자리의 숫자를 디스플레이가 ON/OFF를 빠르게 반복하여 실제로는 계속 켜지고 있지 않지만 샘플링이 느린 사람의 눈에는 계속 LED가 계속 켜진 것처럼 보이는 효과를 이용한 방법이다. 각 ON/OFF 유지 시간 자체를 딜레이 효과를 확인할 수 있도록 적용하면 사람의 눈으로도 ON/OFF를 반복하는 것을 확인할 수 있을 것이라고 예상하며 수행했고 결과 또한 그러하였다.

먼저, 본 레포트의 Ⅲ의 코드로는 Overflow count가 2의 배수일 때마다 각 자리의 LED가 ON/OFF를 교차로 반복한다. 여기서 Overflow count는 인터럽트가 걸리는 주기마다 카운트하는 것을 말하는데, 프리스케일=256을 사용한 주기는 약 4ms이고 2의 배수일 때마다 ON/OFF 되는 것이므로 8ms(=2*4ms)마다 각 LED가 ON/OFF를 반복한다. 유지 시간 자체를 조정하는 것으로 결과 변화를 관찰하려는 목적에 맞게 프리스케일은 고정하고 수행했다.

아래와 같이 설정하여 수행했다.

	아래 칸의 숫자의 배수일 때 해당 자리의 LED 디스플레이						
1의자리	2	4	8	16	32	64	128
10의자리	4	8	16	32	64	128	256

코드는 아래의 부분을 수정하며 수행했다.

```
if(timer0%2==0){
        PORTC=(timer0%4==0) ? led[(number%100) / 10] : led[number%10];
        PORTD=(PORTD|0xC0)&~(1<<((timer0%4==0) ? PD7:PD6));
```

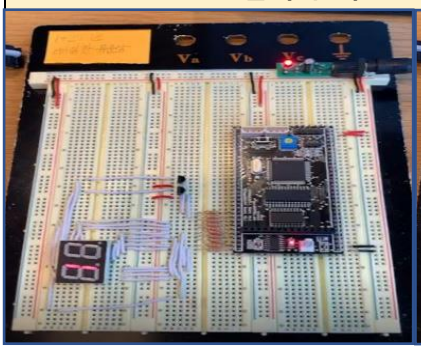
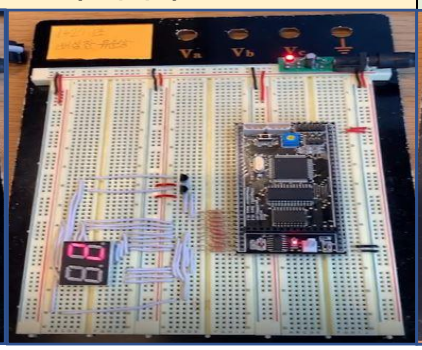
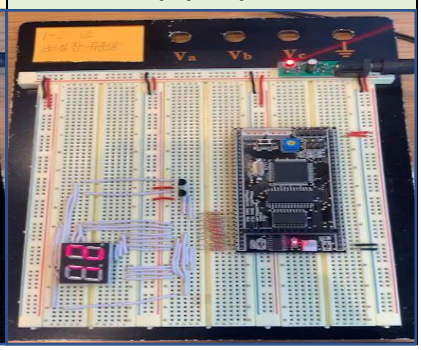
2와 마찬가지로 결과의 변화는 이미지로는 설명할 수 없어서 동영상으로 대체한다. 이론적으로 설명하면, 기존의 (2,4)일 때는 8ms마다 반복되었다면 마지막 (128,256)일 때는 512ms(=128*4ms)마다 반복하므로 사람의 눈으로도 각 자리의 LED가 번갈아 가며 ON되는 것을 확인할 수 있어서 착시 효과로 인한 디스플레이는 이루어지지 않을 만큼 딜레이 효과를 얻은 결과를 확인하였다.

4. 2, 3번을 같이 적용

2, 3번을 수행한 후 두 방법 모두 딜레이 효과를 얻을 수 있다는 것을 확인했다. 같이 적용하면 더 큰 딜레이 효과를 적용할 수 있을 것 같아서 수행하였다. 실험 조건은 각 방법에서 가장 큰 딜레이 효과를 얻은 1024분주와 (128,256)으로 설정하여 수행하였다. 마찬가지로 결과의 변화는 이미지로는 설명할 수 없어서 동영상으로 대체한다.

➔ 이론적으로, $T = \frac{256*1024}{16000000} = 0.016384 = 16.384ms$ 이고, ON/OFF가 유지되는 시간은 약 2.098s(=128*16.384ms)이다. 10의 자리가 ON되고 약 2초가 지나야 꺼지면서 1의 자리 LED가 켜진다. 가장 긴 딜레이 효과를 설정하는 방법임을 실험적, 이론적으로 확인했다.

➔ 딜레이(ON유지 시간으로 생각 가능)가 길어 착시효과가 일어나지 않고 육안으로도 번갈아 깜빡임을 확인할 수 있는 결과 이미지와 착시효과를 일으킬 때의 이미지를 비교하는 것으로 마치겠다.

딜레이 시간 多 결과 이미지		착시효과 O
		

V 결과 및 토의

➔ AVR Studio4를 사용하여 코딩한 것을 JTAG 다운로더를 통해 ATmega128 보드에 연결해서 Flash memory에 저장하였다. 이후 JTAG 케이블 제거 후 실험을 수행했다. 7-세그먼트의 LED에 디스플레이하고 싶은 숫자에 해당하는 미리 계산한 포트 출력값에 따른 16진수값을 AVR Stuido4에서의 코드에 지정하였다. 이후 구성한 회로도와 연결 후 프로그래밍했고 코드를 수정하고 프로그래밍을 반복하여 실험을 수행했다. 각 포트와 저항을 연결 후 7-segment LED의 해당하는 위치와 연결하여 회로를 구성했다. 이전에 수행했던 실험들과 다르게 7-segment LED의 두 자리 디스플레이를 위해 점프를 사용하여 연결 후 수행했다. 이번 실험에서는 추가적으로 딜레이 시간에 따른 다이나믹 디스플레이의 효과를 살펴보았다. 그러기 위해 세운 전략을 총 세가지로 1. 분주 설정, 2. ON 유지 시간 설정, 3. 1, 2 둘 다 설정하는 방법이었다. 세 방법 모두 딜레이 효과를 실험적으로 이론적으로 보임으로써 증명했다. 착시를 이용한 7-segment LED 디스플레이는 사람의 눈이 빠르게 반복되는 것을 캐치하지 못하는 점을 이용해서 경제적으로 사용하기 위한 방법이다. 각 자리의 LED를 번갈아 가며 ON/OFF를 빠르게 반복하는 것으로 실제로는 켜지고 꺼지는 것의 반복이지만 사람은 LED가 계속해서 켜짐으로 인식한다. 위의 세 방법을 사용하면 각 ON/OFF가 사람도 캐치할 수 있을 만큼의 딜레이를 가지고 동작함을 확인했다. 실험을 수행하면서 조원과 토의를 하며 전략을 먼저 세우고 실험해보는 과정이 매우 재밌게 느껴졌고 예상한대로 결과가 나와서 좋았다.