실험 3. 딜레이루틴을 이용한 스위치디바운스

전자공학과 21611646 유준상

Ⅰ 실험 목적

➔ - 스위치 바운싱 현상에 의한 오동작을 확인

- 시간 지연을 이용한 스위치 디바운싱 현상을 확인

- 스위치 디바운싱을 위한 회로

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, 피복 단선 0.6mm, JTAG 다운로더, 스위치

Ⅲ 소스 코드

(1) 바운싱 현상을 제거하지 못한 프로그램

```
#define F_CPU          16000000UL // CPU 주파수가 16MHz일 때 다음과 같이 정의


#include <avr/io.h>


#define     PRESSED   1 // 스위치가 한번 눌린 상태를 1로 정의
#define     RELEASED 0 // 스위치 눌린 이후의 상태를 0으로 정의
```

```
unsigned char digit[]={0x88, 0xBE,0xC4, 0xA4, 0xB2, 0xA1, 0x83, 0xBC, 0x80, 0xB0};
// 0~9 포트 출력값에 해당하는 16진수 값(공통 애노드 기준)

void display_7segled(unsigned char led[],unsigned int number) // sub routine
{		PORTC = led[number];		} // PORTC에 원하는 led에 해당하는 16진수값 할당

int main(void) // main program
{
		int number; // 메인 프로그램 실행동안 스위치가 눌린 횟수를 저장할 변수
		int before; // 이전 상태를 저장할 변수

		DDRC = 0xFF; // PORT C를 출력으로 사용
		DDRD = DDRD & ~(1<<PD0); // PD0를 스위치 사용을 위해 입력핀으로 설정
		PORTD = PORTD | 1<<PD0; // PD0를 내부 풀업저항으로 연결

		number = 0; // number 변수의 초깃값 설정
		before = RELEASED; // before 변수의 초기값을 설정

		while(1){
			display_7segled(digit, number%10);
			// number가 10이상일 때를 위해 %10을 사용해 나머지로 인덱싱
			if(before==RELEASED && !(PIND&1<<PD0)){ //전에 눌리지 않은 상태에서 처음으로 눌림
				number++; // 스위치가 눌렸으니 number를 1증가
				before=PRESSED; // 눌린상태로 저장
			}else if(before==PRESSED&&(PIND&1<<PD0)){ // 전에 눌린 상태에서 처음으로 떨어짐
				before=RELEASED; // 떨어진 상태로 저장
			}
		}
		return 0;
}
```

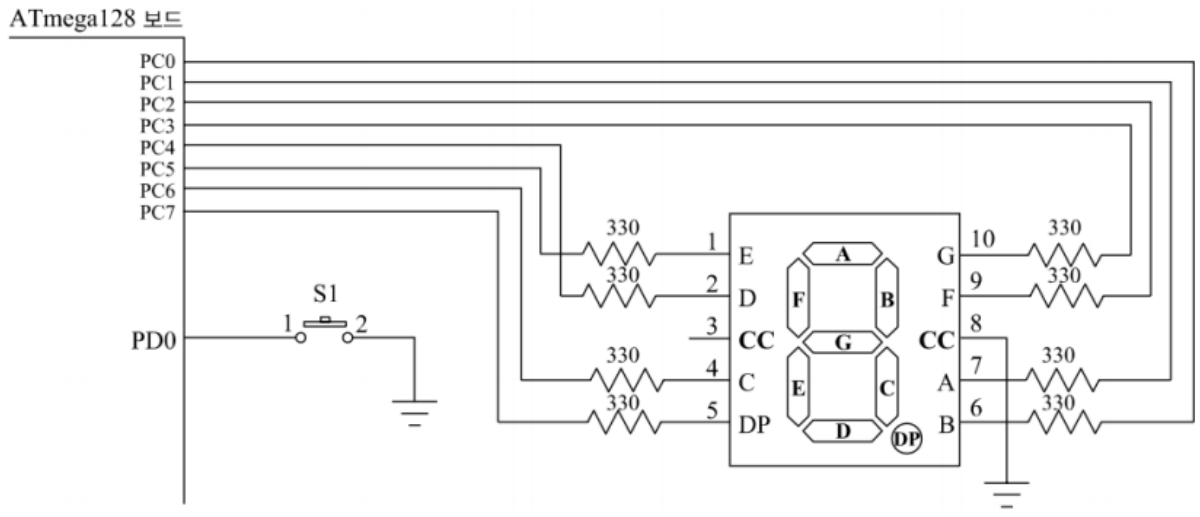
(2) 바운싱 현상을 제거한 프로그램

```
#define F_CPU          16000000UL // CPU 주파수가 16MHz일 때 다음과 같이 정의


#include <avr/io.h>

#include <util/delay.h> // _delay_ms()함수를 사용하기 위해


#define PRESSED      1 // 스위치가 한번 눌린 상태를 1로 정의

#define     RELEASED 0 // 스위치 눌린 이후의 상태를 0으로 정의

#define DEBOUNCE_MS            1000 // 1000ms delay를 주기위해 1000으로 정의


unsigned char digit[]={0x88, 0xBE,0xC4, 0xA4, 0xB2, 0xA1, 0x83, 0xBC, 0x80, 0xB0};

// 0~9 포트 출력값에 해당하는 16진수 값(공통 애노드 기준)


void display_7segled(unsigned char led[],unsigned int number) // sub routine

{              PORTC = led[number];              } // PORTC에 원하는 led에 해당하는 16진수값 할당


int main(void) // main program

{

              int number; // 메인 프로그램 실행동안 스위치가 눌린 횟수를 저장할 변수

              int before; // 이전 상태를 저장할 변수


              DDRC = 0xFF; // PORT C를 출력으로 사용

              DDRD = DDRD & ~(1<<PD0); // PD0를 스위치 사용을 위해 입력핀으로 설정

              PORTD = PORTD | 1<<PD0; // PD0를 내부 풀업저항으로 연결


              number = 0; // number 변수의 초깃값 설정

              before = RELEASED; // before 변수의 초기값을 설정


              while(1){
```

```
                display_7segled(digit, number%10);
                // number가 10이상일 때를 위해 %10을 사용해 나머지로 인덱싱
                if(before==RELEASED && !(PIND&1<<PD0)){ //전에 눌리지 않은 상태에서 처음으로 눌림
                        number++; // 스위치가 눌렸으니 number를 1증가
                        _delay_ms(DEBOUNCE_MS); // 1000ms만큼 딜레이
                        before=PRESSED; // 눌린상태로 저장
                }else if(before==PRESSED&&(PIND&1<<PD0)){ // 전에 눌린 상태에서 처음으로 떨어짐
                        _delay_ms(DEBOUNCE_MS); // 1000ms만큼 딜레이
                        before=RELEASED; // 떨어진 상태로 저장
                }
        }
        return 0;
}
```

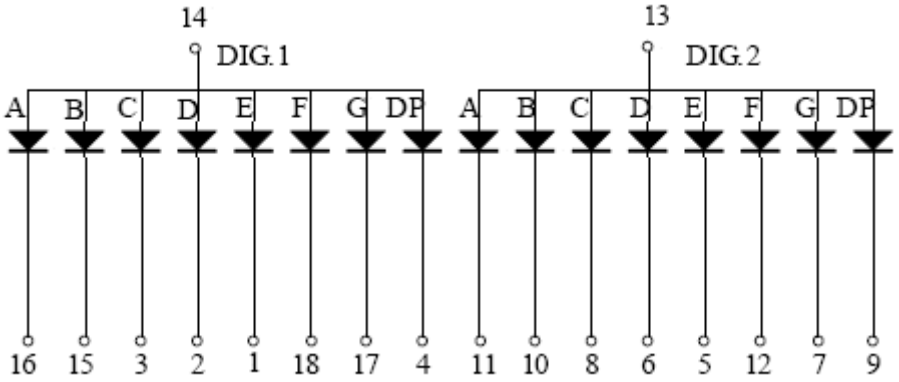
IV 회로도 및 7-segment 작동

➔ 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.



위의 사진은 PORT C와 7-segment의 연결 관계에 대해 나타낸 회로도이다. 위와 다르게 본 실험

에서는 2칸 7-segment를 사용했으므로 번호가 다르다.

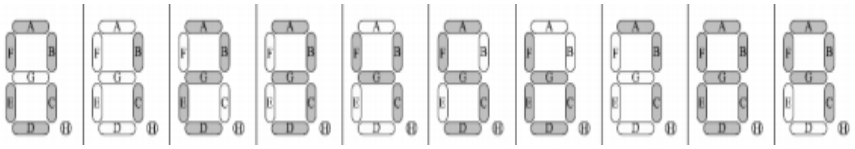


위의 사진은 실험에서 사용한 7-세그먼트 5263ASR1에 해당하는 회로도이다. 본 실험에서는 DIG2에 해당하는 번호로 연결 후 사용했다.

7-세그먼트 LED의 내부 LED와 ATmega128 PORT C의 핀과의 관계인데,

ATmega128 C포트의 핀 이름	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
7-Segment LED의 내부 LED	DP	C	E	D	G	F	A	B

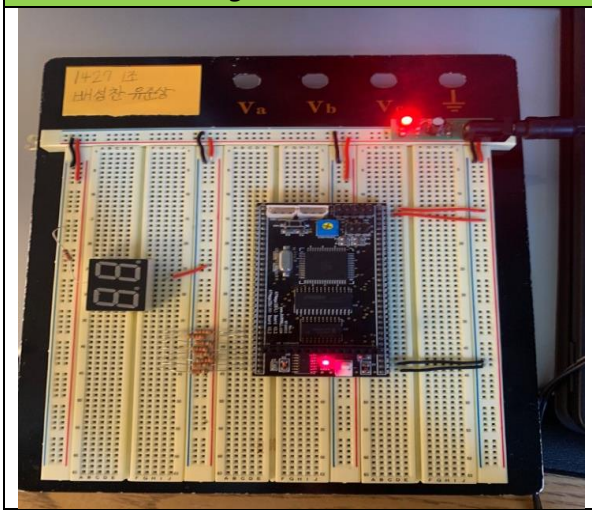
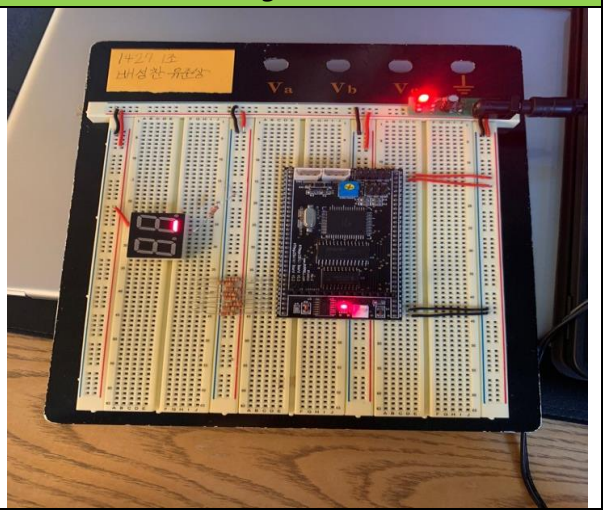
강의자료의 표는 공통 캐소드 기준에서의 16진수 값이므로 아래와 같이 실험에서 사용한 공통 애노드일 때의 값으로 변환 후 코드에 적용하여 사용했다.

			디스플레이 숫자									
		LED	0	1	2	3	4	5	6	7	8	9
		위치										
포	PC7	DP	1	1	1	1	1	1	1	1	1	1

트	PC6	C	0	0	1	0	0	0	0	0	0	0
출력값	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	A	0	1	0	0	1	0	1	0	0	0
	PC0	B	0	0	0	0	0	1	1	0	0	0
16진수값			88	BE	C4	A4	B2	A1	83	BC	80	B0

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

사용하는 7segment가 공통 애노드인지, 공통 캐소드인지 알아보기 위해 아래와 같이 vcc와 GND만 연결 후 알아보았고, 확인 결과 공통 애노드 7segment임을 확인했다. 공통 애노드 7segment는 LOW값이 들어갈 때 LED가 켜지는 특성을 가진다.

공통 캐소드 7segment가 아니라서 LED OFF	공통 애노드 7segment이므로 LED ON
	

위까지의 과정은 실험2와 같다. 실험2와 다른 점은 PD0에 스위치를 연결한다는 점이다. 스위치를 사용하는 이유는 스위치의 On/Off 동작에 따른 바운싱 현상을 관찰하고 소프트웨어적으로 해결

하는 지도 보기 위해서 이다. 스위치 디바운싱 효과란 아래이다.

★ 스위치 바운싱 현상

- 스위치를 여러 번 누른 동작으로 오동작할 수 있음

- 스위치의 기계적인 특성으로 발생 ➔ 전압 변화 유발

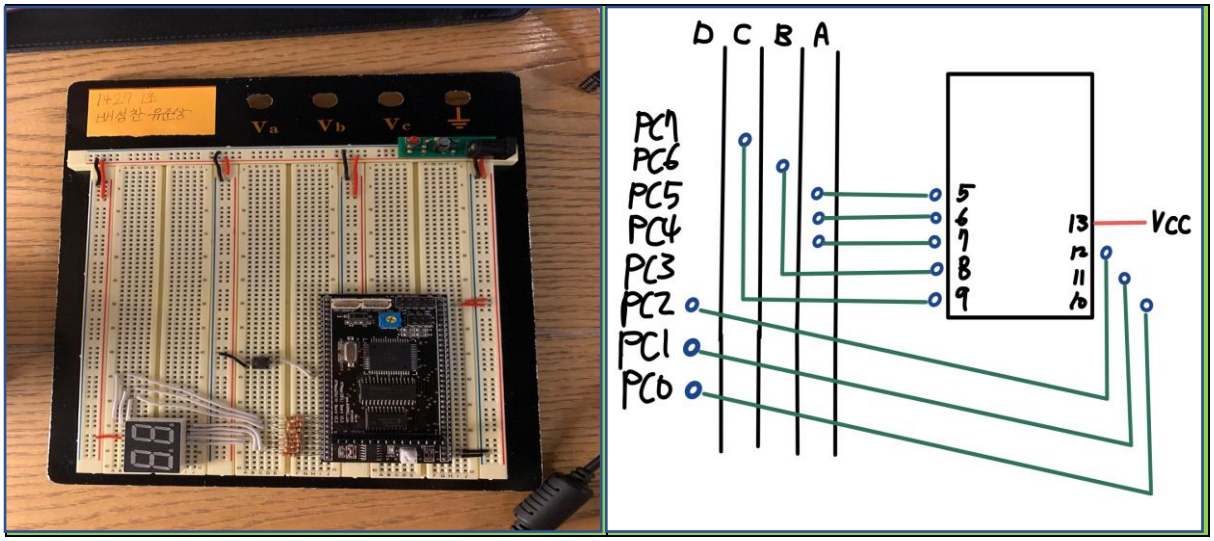
- 소프트웨어적으로 스위치 디바운싱 역할 수행

  -- 스위치 변화를 처음 감지 후, 일정시간 동안 변화를 무시함

[스위치 바운싱에 의한 전압 변화]

그리고, 스위치 바운싱 현상은 시간 지연 함수(_delay_ms())를 사용하여 해결한다. 이를 스위치 디바운싱 이라 한다.

기본 회로도	회로도 이해를 돕기 위한 이미지

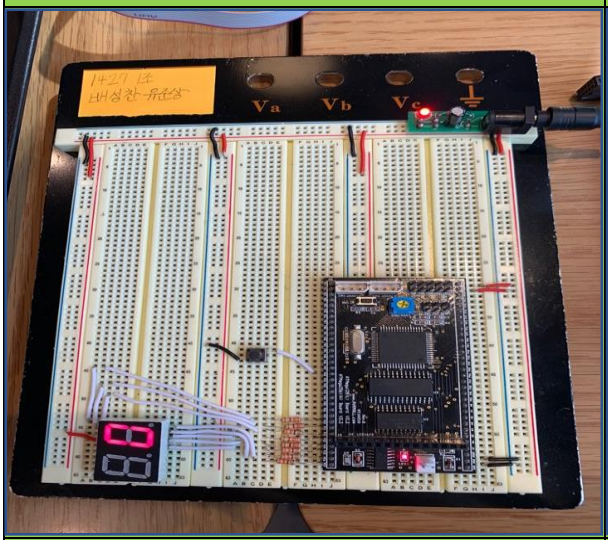
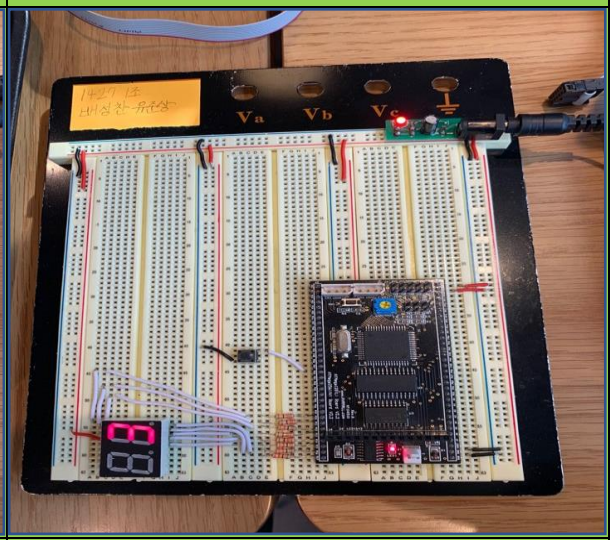


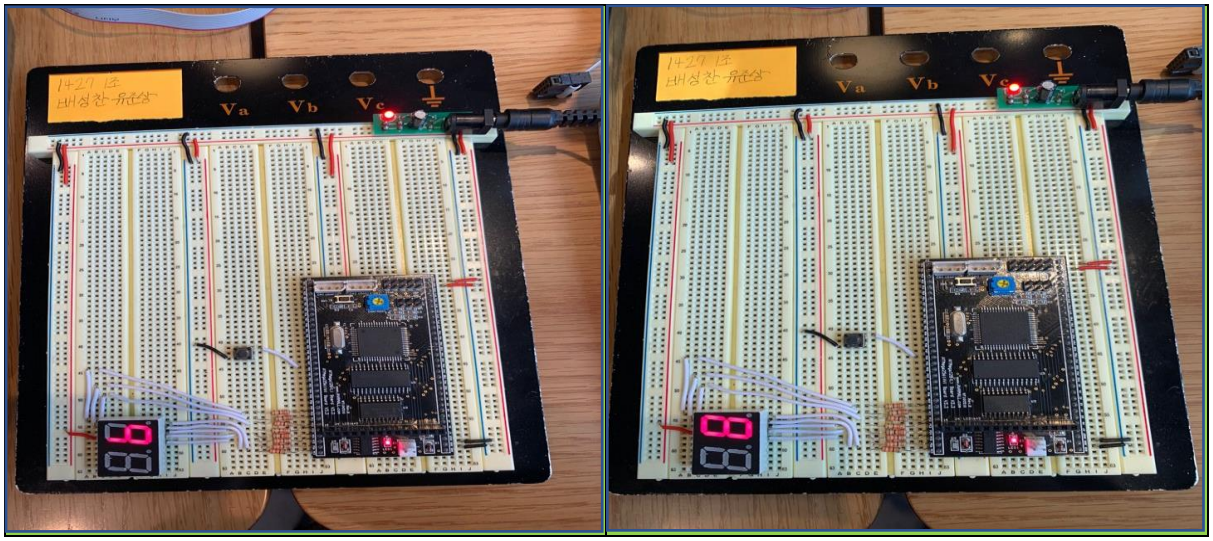
위는 JTAG을 통한 프로그래밍 전 회로도 구성을 완료한 이미지이다.

[바운싱 효과를 제거하지 못한 동작 이미지]

딜레이를 몇 이상 적용했을 때 바운싱 효과를 제거할 수 있는 지 알아보기 위해서, 1ms부터 1ms 씩 올리며 딜레이를 적용하며 실험을 수행해보았다. 결과부터 말하자면 본 실험에서는 5ms에서는 바운싱 효과를 확인할 수 있었고 6ms부터는 디바운스 된 것을 실험적으로 알 수 있었다.

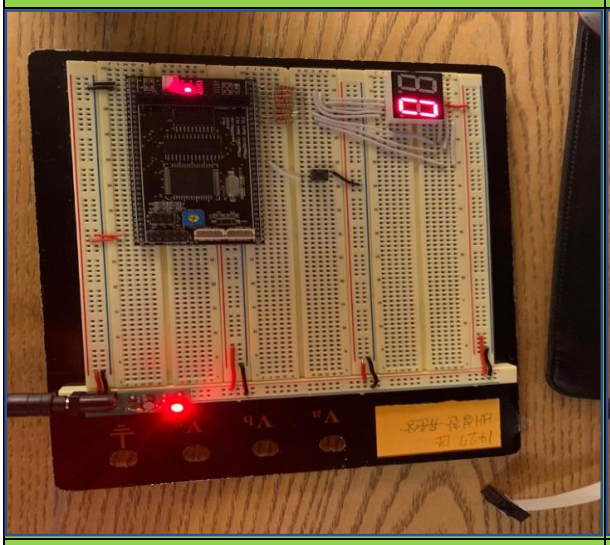
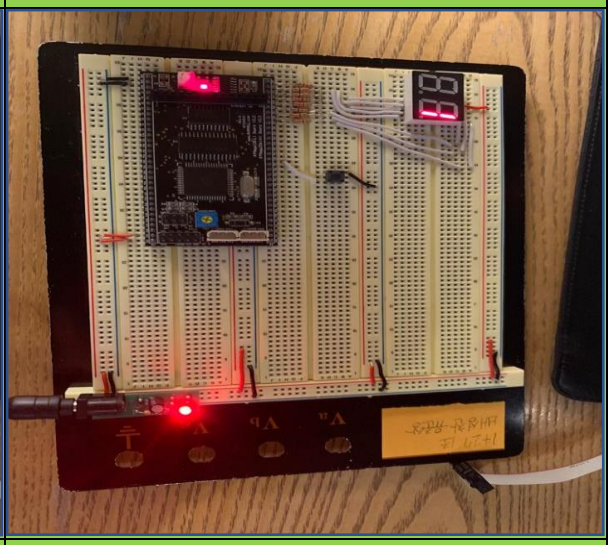
아래는 바운싱이 일어났을 때 변화를 알려주는 이미지이다. 7segment display가 0을 나타낼 때, 스위치를 한번 눌렀고, 1이 아닌 3이 나타났고, 6에서 스위치를 한번 눌렀을 때 7이 아닌 8이 표시됨으로 바운싱 효과를 확인할 수 있었다.

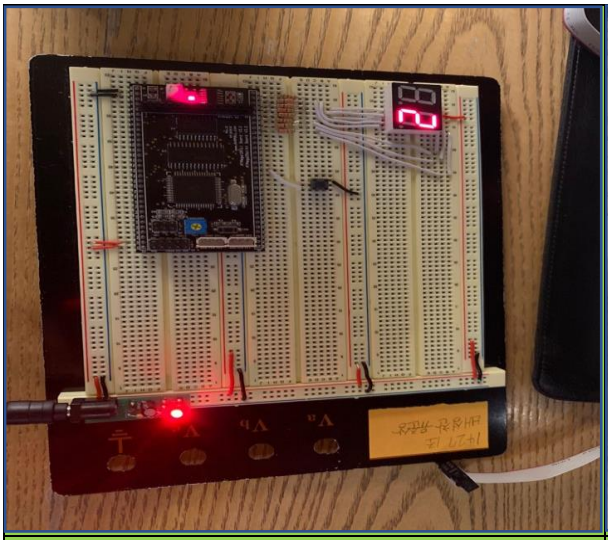
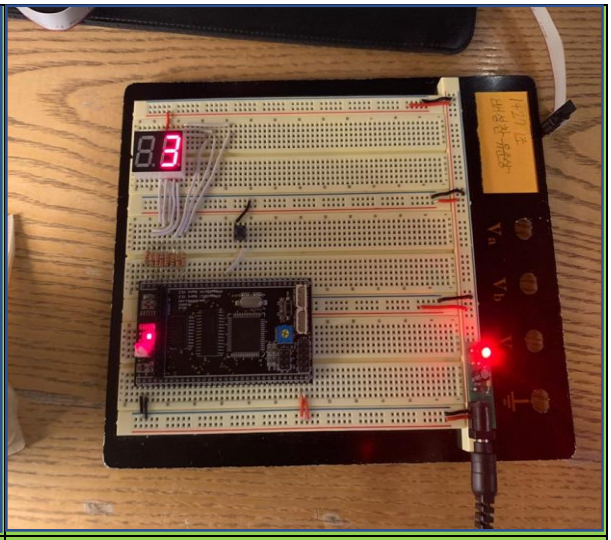
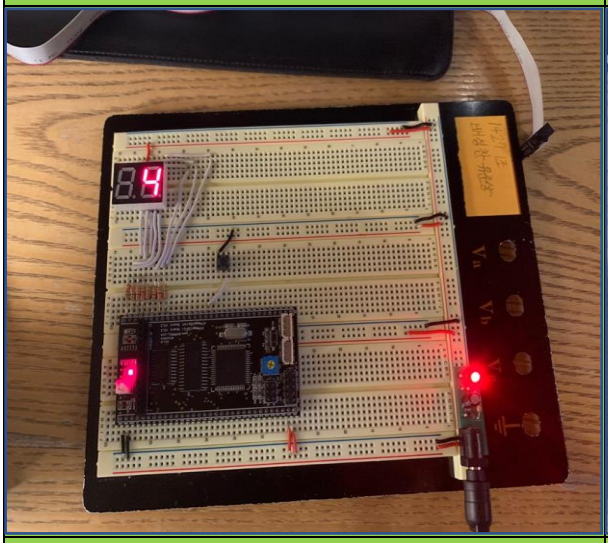
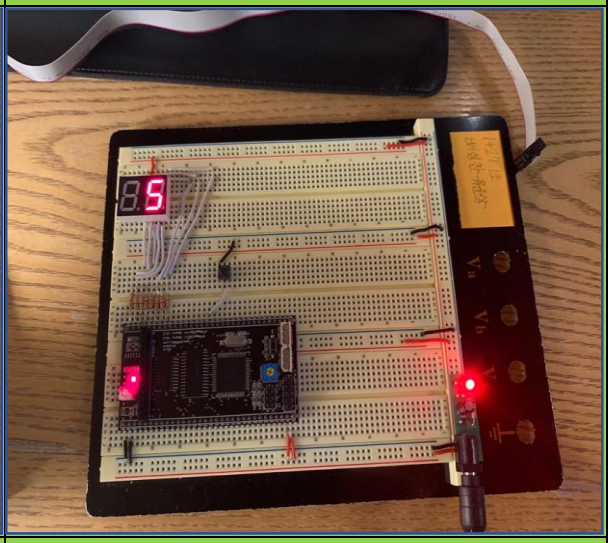
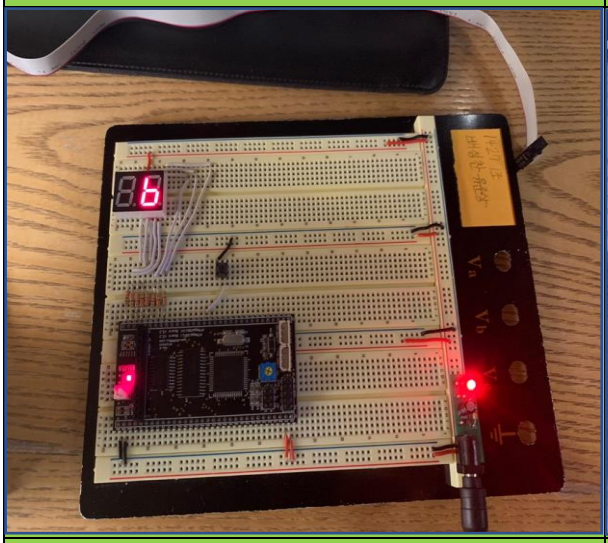
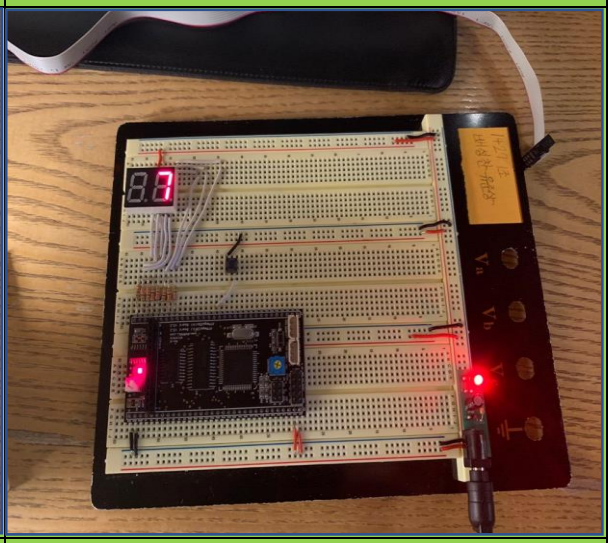
0 ➔	3
	
6 ➔	8

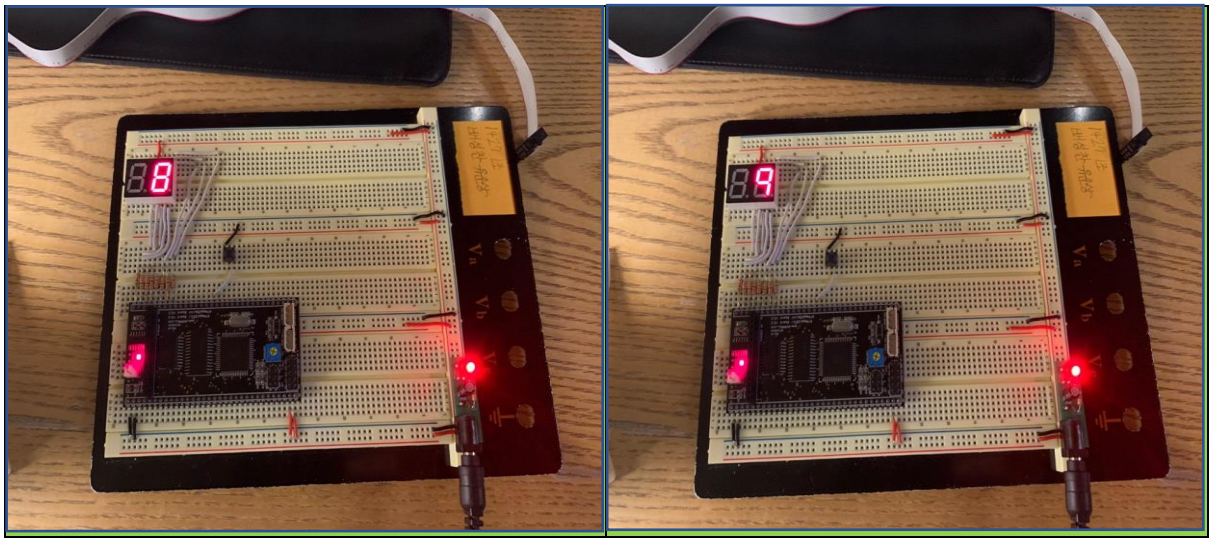


[바운싱 효과를 제거한 회로 및 동작 이미지]

아래는 시간 지연 함수를 사용해서 1초 딜레이를 주고 수행한 결과 이미지이다. 각 입력 숫자에 맞게 잘 작동하는 지 확인하기 위해 코드 수정, 다운로드, 실행 과정을 반복하여 수행했다. 잘 작동한다.

0	1
	
2	3

	
4	5
	
6	7
	
8	9



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.

실험 목표대로 딜레이를 적용하지 않은 실험에서 스위치를 한번 눌러도 LED가 1~2회 더 변하는 스위칭 바운싱 현상에 의한 오동작을 확인했다. 그리고 딜레이를 이용해서 해결한 스위치 디바운싱 현상을 확인했다.

이번 실험을 통해 바운싱 효과 및 디바운싱에 대해 알게 되었다. 정상 동작을 위해서는 일정 시간 이상의 딜레이를 적용해줘야 한다. 공학도로써 코딩을 할 때, 기계적인 특성을 인지하고 코딩을 해야 되겠다는 생각이 들었다.