임베디드시스템설계및실험

3주차 결과보고서

2023.09.25

2조

202155592 이지수

202155553 박은재

201924515 유승훈

201524410 고상현

201924402 강민수

A. 실험 목적

1. 임베디드 시스템의 기본 원리 습득
2. 디버깅 툴 사용방법 습득 및 레지스터 제어를 통한 임베디드 펌웨어 개발

B. 세부 목표

1. IAR Embedded Workbench 프로젝트 생성 및 환경 설정
2. Data Sheet, Reference Manual, Schematic을 참고하여 레지스터 설정 이해
3. 푸쉬 버튼을 이용하여 LED 제어

    Key1: PD2 LED 토글 (ON/OFF)

    Key2: PD3 LED 토글 (ON/OFF)

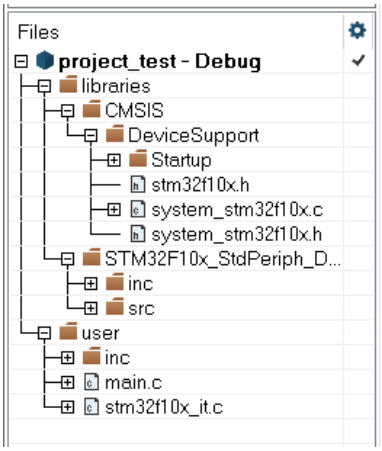
    Key3: PD4 LED 토글 (ON/OFF)

    Key4: PD7 LED 토글 (ON/OFF)

C. 실험 과정

1. 프로젝트 생성 및 설정

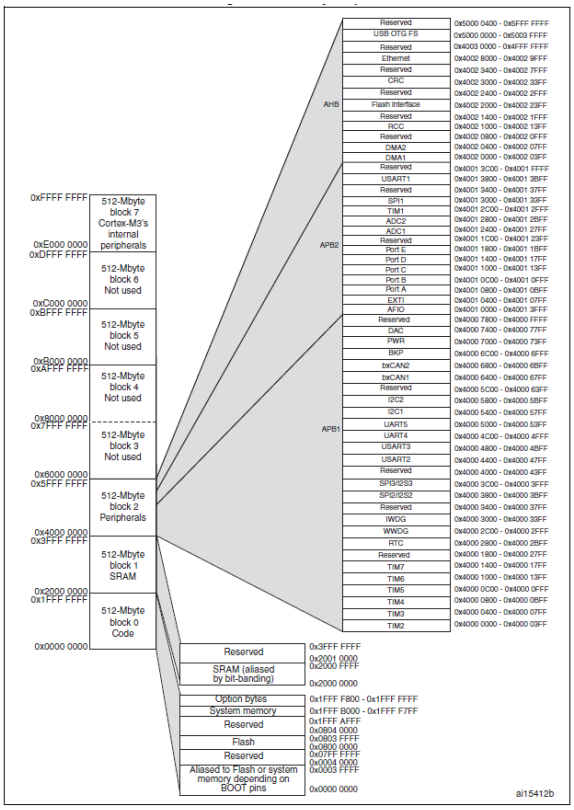
IAR Embedded Workbench에서 프로젝트를 생성 후 [그림1]과 같은 구조로 디렉토리 구조를 만들고 파일을 추가한다. 이후 타겟 디바이스, 라이브러리, Output Converter등 프로젝트 설정을 해준다. 설정이 끝났다면, 보드에 전원과 JTAG를 연결하고, PC와 연결 후 보드에 포팅을 해준다.



[그림 1] 프로젝트 디렉토리 구조

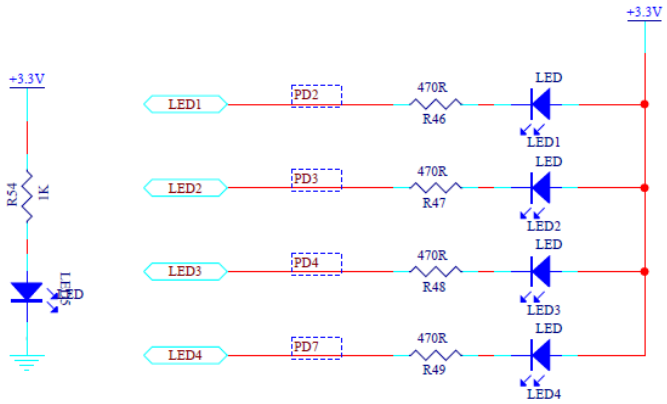
2. RCC 주소 선언, LED와 푸쉬 버튼의 GPIO포트 확인 및 주소 선언

LED와 푸쉬 버튼을 사용하기 위해서는 이 장치들의 GPIO포트와 포트 핀 번호, 해당하는 메모리 주소를 확인해야 한다. GPIO포트는 Data Sheet를 확인해보면 [그림 3]과 같이 APB2에 할당되어 있는 것을 볼 수 있다.

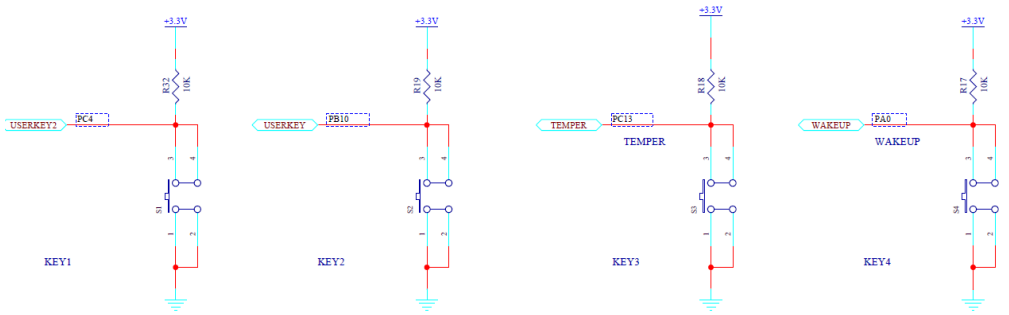


[그림 3] STM32 보드의 Memory Map

그리고 보드의 Schematic에서 LED와 푸쉬 버튼이 어떤 포트와 핀을 사용하는지 확인 할 수 있는데, [그림 4]를 보면 LED는 각각 PD2, PD3, PD4, PD7로 포트 D를 사용하고, [그림 5]를 보면, 푸쉬 버튼의 Key1은 PC4, Key2는 PB10, Key3는 PC13, Key4는 PA0으로 A, B, C 포트를 사용하는 것을 확인할 수 있다.



[그림 4] Schematic of LED



[그림 5] Schematic of Button

GPIO 포트를 사용하기 위해 RCC를 사용하여 clock을 각 포트에 인가해주어야 한다. 포트 A, B, C, D는 APB2에 할당이 되어 있으므로 APB2 peripheral clock enable register(RCC_APB2ENR)에 접근해야 한다.

[그림 3]에서 RCC의 Base Address는 0x40021000이고, [그림 6]을 보면 RCC_APB2ENR의 offset은 0x18이므로 두 주소를 더해주면 RCC_APB2ENR의 주소는 0x40021018임을 알 수 있다.

7.3.7 APB2 peripheral clock enable register (RCC_APB2ENR)

Address: 0x18

Reset value: 0x0000 0000

Access: word, half-word and byte access

No wait states, except if the access occurs while an access to a peripheral in the APB2 domain is on going. In this case, wait states are inserted until the access to APB2 peripheral is finished.

Note: When the peripheral clock is not active, the peripheral register values may not be readable by software and the returned value is always 0x0.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16

[그림 6] RCC_APB2ENR offset

```
// GPIOA, B, C, D에 clock 인가
#define RCC_APB2ENR (*(volatile unsigned int*) 0x40021018) // 0x4002 1000 + 0x18
```

Clock enable 시키는 레지스터 주소를 선언하였으니, 이제는 푸쉬 버튼의 key1, key2, key3, key4를 사용하는 GPIO 포트와 LED에서 사용하는 레지스터 주소를 선언하여야 한다. 푸쉬 버튼에서 사용하는 GPIO 포트는 A, B, C 포트이고, GPIO A,B,C포트의 시작 주소는 [그림 3]을 보면 확인할 수 있다. 마찬가지로 LED에서 사용하는 GPIO D포트의 시작 주소도 [그림 3]을 보면 확인 가능하다.

9.2.1 Port configuration register low (GPIOx_CRL) (x=A..G)

Address offset: 0x00

Reset value: 0x4444 4444

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF7[1:0]		MODE7[1:0]		CNF6[1:0]		MODE6[1:0]		CNF5[1:0]		MODE5[1:0]		CNF4[1:0]		MODE4[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF3[1:0]		MODE3[1:0]		CNF2[1:0]		MODE2[1:0]		CNF1[1:0]		MODE1[1:0]		CNF0[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

[그림 7] Configuration Register Low

9.2.2 Port configuration register high (GPIOx_CRH) (x=A..G)

Address offset: 0x04

Reset value: 0x4444 4444

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF15[1:0]		MODE15[1:0]		CNF14[1:0]		MODE14[1:0]		CNF13[1:0]		MODE13[1:0]		CNF12[1:0]		MODE12[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF11[1:0]		MODE11[1:0]		CNF10[1:0]		MODE10[1:0]		CNF9[1:0]		MODE9[1:0]		CNF8[1:0]		MODE8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

[그림 8] Configuration Register High

각각의 GPIO 포트는 0~7번까지는 Configuration Register Low를 사용하고, 8~15번까지는 Configuration Register High를 사용한다.

```
// Port A, B, C, D : configuration register low
#define GPIOA_CRL (*(volatile unsigned int *) 0x40010800) // A (0 => CRL)
#define GPIOB_CRH (*(volatile unsigned int *) 0x40010C04) // B (10 => CRH)
#define GPIOC_CRL (*(volatile unsigned int *) 0x40011000) // C (4 => CRL)
#define GPIOC_CRH (*(volatile unsigned int *) 0x40011004) // C (13 => CRH)
#define GPIOD_CRL (*(volatile unsigned int *) 0x40011400) // D (2, 3, 4, 7 => CRL)
```

각각의 포트 레지스터의 base Address에 알맞은 값을 Reference Manual의 [그림 7], [그림 8]을 참고하여 더하여 선언해주었다.

3. GPIO 포트 A, B, C의 Input Register 주소 선언 및 D 포트 BRR, BSRR 주소 선언

9.2.3 Port input data register (GPIOx_IDR) (x=A..G)

Address offset: 0x08h

Reset value: 0x0000 XXXX

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **IDRy**: Port input data (y= 0 .. 15)
These bits are read only and can be accessed in Word mode only. They contain the input value of the corresponding I/O port.

[그림 9] Port input data register

푸쉬 버튼인 GPIO A, B, C에 input이 들어왔을 때 그 값을 읽어와야 함으로 각각의 input data register의 주소도 [그림 9]에 나와있는 값들을 참고하여 선언해준다.

```
// Port A, B, C : input data register
#define GPIOA_IDR (*(volatile unsigned int *) 0x40010808) // A 0x40010800 + 0x08
#define GPIOB_IDR (*(volatile unsigned int *) 0x40010C08) // B 0x40010C00 + 0x08
#define GPIOC_IDR (*(volatile unsigned int *) 0x40011008) // C 0x40011000 + 0x08
```

또한 LED의 OUTPUT 값을 조작하기 위하여 bit set/reset Register와 bit reset Register의 시작 주소를 선언해준다. Output Register의 값을 직접 조작하지 않고 BSRR과 BRR을 사용하는 이유는 [그림 10]을 참고하면 ODR 비트는 BSRR을 통해서 set, clear될 수 있기 때문이다.

Note: For atomic bit set/reset, the ODR bits can be individually set and cleared by writing to the GPIOx_BSRR register (x = A .. G).

[그림 10] Note of ODR Register

따라서 D포트에 대한 BSRR과 BRR을 [그림 11], [그림 12]을 참고하여 선언해준다.

```
// Port D  : ouput -> BRR : reset, BSRR : reset or set
#define GPIOD_BRR (*(volatile unsigned int *) 0x40011414) // 0x40011400 + 0x14
#define GPIOD_BSRR (*(volatile unsigned int *) 0x40011410) // 0x40011400 + 0x10
```

9.2.6 Port bit reset register (GPIOx_BRR) (x=A..G)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

[그림 11] Port bit reset register

9.2.5 Port bit set/reset register (GPIOx_BSRR) (x=A..G)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

[그림 12] Port bit set/reset register

4. main함수 구현 및 동작 확인

사용하는 레지스터의 주소를 모두 선언했으니 기능 구현을 위해 main함수를 작성하였다. 먼저 GPIO A, B, C, D 포트를 모두 사용하므로, 네 포트에 전부 clock을 인가하였다.

Bit 5 **IOPDEN**: IO port D clock enable
Set and cleared by software.
0: IO port D clock disabled
1: IO port D clock enabled

Bit 4 **IOPCEN**: IO port C clock enable
Set and cleared by software.
0: IO port C clock disabled
1: IO port C clock enabled

Bit 3 **IOPBEN**: IO port B clock enable
Set and cleared by software.
0: IO port B clock disabled
1: IO port B clock enabled

Bit 2 **IOPAEN**: IO port A clock enable
Set and cleared by software.
0: IO port A clock disabled
1: IO port A clock enabled

[그림 13] Port enable

[그림 13] RCC의 enable 비트를 참고하여 구현하였다. 구현 내용은 다음과 같다.

```
// PORT D ENABLE
RCC_APB2ENR |= 0x20; //0x0000 0000 0010 0000

// PORT C ENABLE
RCC_APB2ENR |= 0x10; //0x0000 0000 0001 0000

// PORT B ENABLE
RCC_APB2ENR |= 0x8; //0x0000 0000 0000 1000

// PORT A ENABLE
RCC_APB2ENR |= 0x4; //0x0000 0000 0000 0100
```

또한 GPIO 포트 A, B, C는 input모드로 설정해야 하기 때문에 Reference manual의 [그림 7], [그림 8]의 configuration bit와 mode bit를 참고하여 input모드로 설정해준다. 구현 내용은 다음과 같다.

```
// PORT A : Input pull up, pull down
GPIOA_CRL &= ~0x0000000F; //0번째 비트에 해당하는 부분을 0으로 초기화
GPIOA_CRL |= 0x00000008; //1000(=8)을 0번째 비트에 넣기

// PORT B : Input pull up, pull down
GPIOB_CRH &= ~0x00000F00;  //10번째 비트에 해당하는 부분을 0으로 초기화
GPIOB_CRH |= 0x00000800; //1000(=8)을 10번째 비트에 넣기

// PORT C : Input pull up, pull down
GPIOC_CRL &= ~0x000F0000; //4번째 비트에 해당하는 부분을 0으로 초기화
GPIOC_CRL |= 0x00080000; //1000(=8)을 4번째 비트에 넣기
GPIOC_CRH &= ~0x00F00000; //13번째 비트에 해당하는 부분을 0으로 초기화
GPIOC_CRH |= 0x00800000; //1000(=8)을 13번째 비트에 넣기
```

D포트도 마찬가지로 output 모드로 설정해준다. 구현 내용은 다음과 같다.

```
// PORT D : General purpose output push-pull
GPIOD_CRL &= ~0xF00FFF00; //2,3,4,7번째 비트에 해당하는 부분을 0으로 초기화
GPIOD_CRL |= 0x30033300; //0011(=3)을 2,3,4,7번째 비트에 넣기
```

```
// 제일 처음에 모든 LED 끄기
GPIOD_BSRR &= 0x00000000;
GPIOD_BSRR |= 0x9C;
```

마지막으로 동작 시작 시 네 개의 LED는 꺼진 상태로 시작해야 하므로 BSRR의 모든 비트를 0으로 설정한 뒤, 2, 3, 4, 7번만 1로 설정하여 Reset한다.

동작을 무한 반복시키기 위해 while 무한루프 안에서 기능을 구현하였다.

또한 각각의 key1, key2, key3, key4 동일한 방식으로 작동하므로 이 보고서에서는 Key1에 대해서만 설명하겠다.

먼저 PD2, PD3, PD4, PD7의 버튼의 On/Off 값을 코드 내에 저장하기 위해

```
uint8_t button_flag = 0; // For PD2, PD3, PD4, PD7
```

4 비트용 toggle-flag변수를 만들었다.

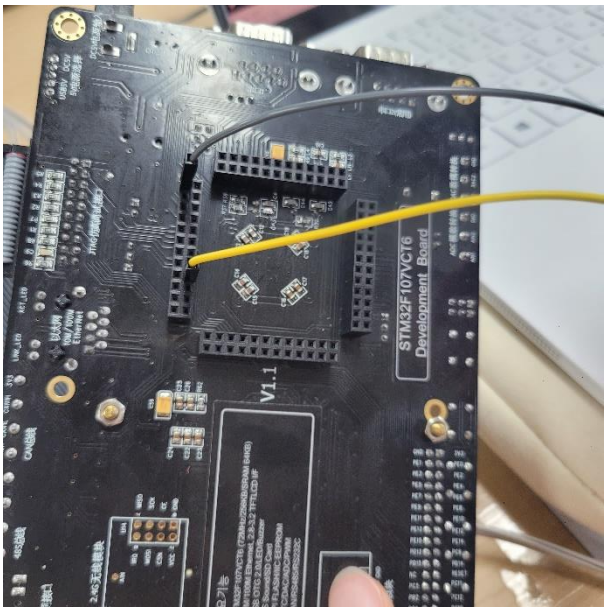
```
if (~GPIOC_IDR & (0x1 << 4)) { // KEY1
    button_flag ^= (1 << 0); // Toggle PD2

    if(button_flag & (1 << 0)) // PD2의 flag가 1일 때
        GPIOD_BSRR |= 0x04; // PD2 ON
    else
        GPIOD_BRR |= 0x04; // PD2 OFF

    delay(1000000);
}
```

Key1은 GPIO PC4가 사용되며, 따라서 IDR4 값을 read해서 key1의 값변화를 감지한다. 감지되었다면, toggle flag의 첫번째 비트를 XOR연산으로 Toggle 한다. 만약 PD2의 flag비트가 1이 되었다면(버튼이 눌렸다면) PD2를 ON으로 만들고 flag비트가 0이라면 PD2를 OFF상태로 바꾼다.

위와 같은 코드를 Key2, Key3, Key4에 대해서도 동일한 방식으로 구현하였다.

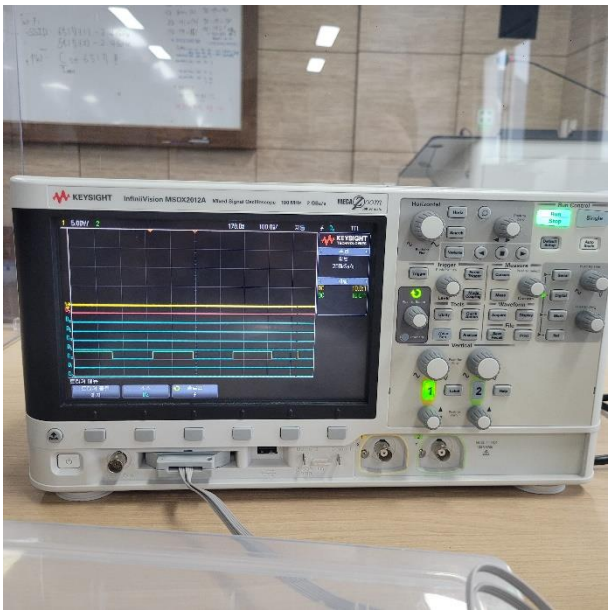
5. 오실로스코프 디버깅

보드에 올린 코드의 디버깅과 LED의 점등 시간 측정을 위해 오실로스코프를 활용하였다. [그림 14]와 같이 점퍼 케이블을 GND와 PD2(첫번째 LED)에 연결하였다.



[그림 14] GND와 PD2에 점퍼케이블을 연결

오실로스코프의 Time Scale을 조정하고, key1 버튼을 누르고, 한번 더 누르는 과정을 통해 Rising Edge와 Falling Edge간의 시간 간격을 측정하였고 LED가 켜졌다가 꺼진 시간을 뺐을 때, 178ms로 측정되었다.



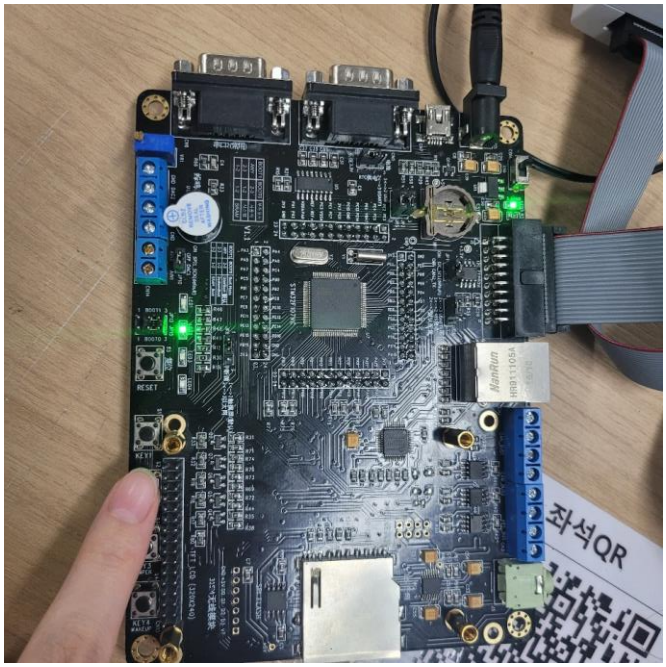
[그림 15]

D. 결론 및 느낀 점

3주차 실험으로 IAR Imbedded Workbench에서의 프로젝트 생성 및 설정 방법에 대해 익히고 Data Sheet, Reference Manual, Schematic등의 문서를 직접 읽고 이해한 내용을 바탕으로 펌웨어를 구현하는 방법을 알게 되었다. 처음에는 문서의 내용이 생소하여 많이 난해하기도 하였지만 레지스터 주소 값을 찾고 분석하는 과정이 익숙해지고 난 뒤에는 수월하게 실험을 진행할 수 있었다.

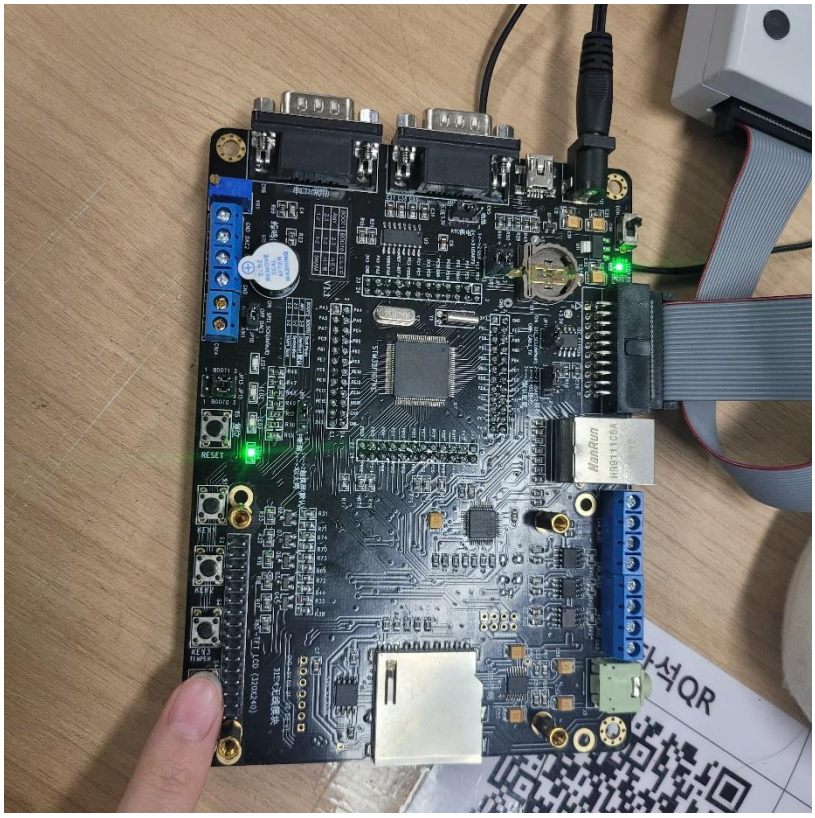
E. 결과 사진

[그림 16] Key 1을 눌렀을 때 LED1이 켜짐



[그림 17] Key 2를 눌렀을 때 LED2가 켜짐

[그림 18] Key 3을 눌렀을 때 LED3이 켜짐



[그림 19] Key 4를 눌렀을 때 LED4가 켜짐