임베디드 시스템 설계 및 실험 3주차 실험 결과 보고서

004분반 9조

조장 – 김주송 조원 – 강선호, 박형주, 손봉국

1. 실험 목적

- 임베디드 시스템 설계의 기본 원리 습득
- 디버깅 툴 사용방법 습득 및 레지스터 제어를 통한 임베디드 펌웨어 개발

2. 실험 목표

1 Datasheet 및 Reference Manual을 참고하여 해당 레지스터 및 주소에 대한 설정 이해

2. IAR EW에서 프로젝트 생성 후 관련 설정 변경

3. 버튼을 이용한 LED 제어 (별도 flag 변수 필요, delay 필요 시 for문으로 대체)

KEY1 : PD1 LED On / Off (toggle)
KEY2 : PD2 LED On / Off (toggle)
KEY3 : PD3 LED On / Off (toggle)
KEY4 : PD4 LED On / Off (toggle)

4. 정상적인 동작 유무 확인

5. 오실로스코프를 이용한 디버깅

(버튼 toggle 시 LED가 켜졌다가 꺼진 시간을 측정, Digital Pin의 trigger를 이용하여 캡쳐)

3. 실험 내용

이번 프로젝트는 레지스터의 주소 제어를 통해 GPIO를 제어하는 것이다. 아래의 System architecture를 확인해보면, GPIO 포트들은 APB2에 할당되어 있다. 따라서 APB2 포트의 주소를 알아야 한다.

Figure 1. System architecture (low-, medium-, XL-density devices)

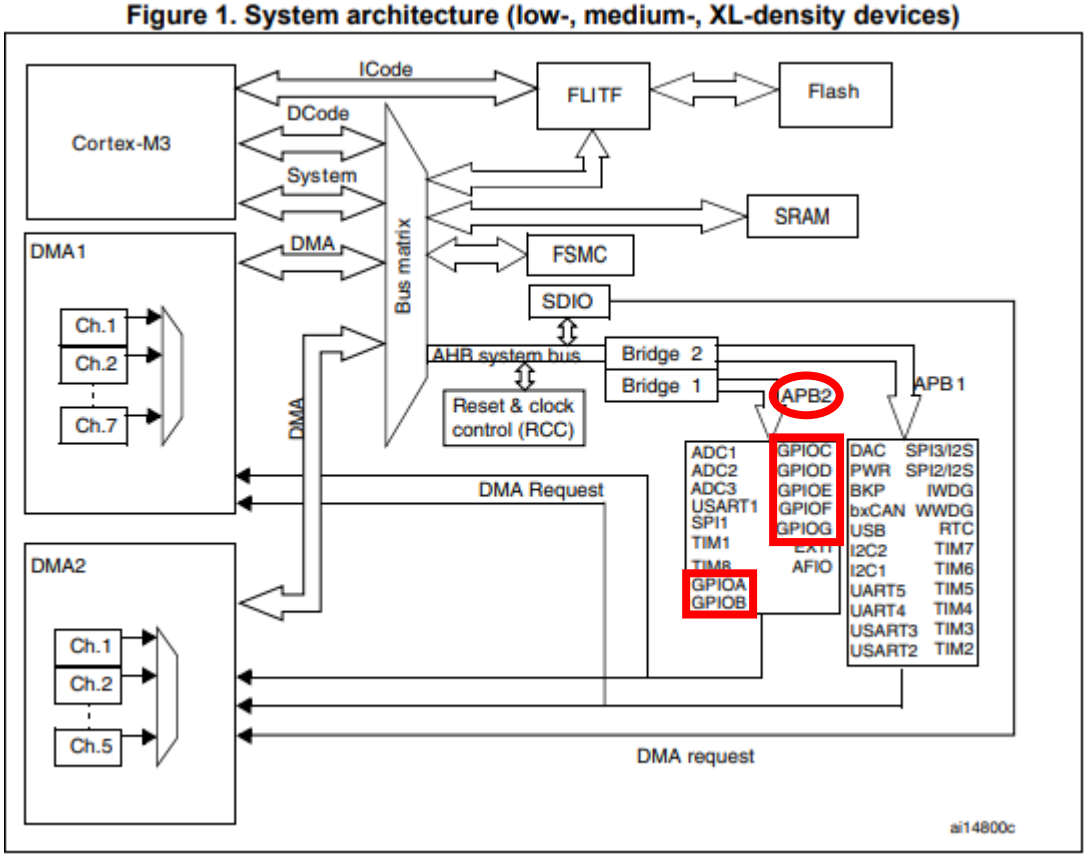

그림 1. System architecture(Reference Manual p.48)

Memory mapping을 확인하여 필요한 주소를 가져온다. RCC(Reset and Clock Control)의 Base Address가 0x4002 1000임을 알 수 있다.

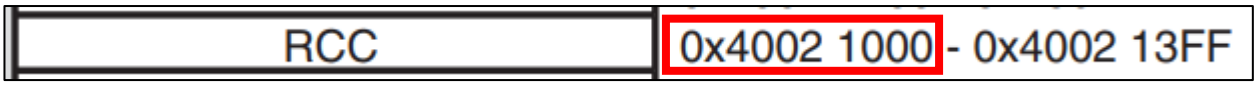

그림 2. RCC의 Base Register 주소(stm32_Datasheet p.33)

그리고 이를 통해 APB2를 제어하기 위한 Offset이 필요하다. 제공된 Reference Manual을 참고하여 RCC_APB2ENR의 Offset이 0x18임을 알 수 있다.

8.3.7 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.

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
Res.	USART1EN	Res.	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	Reserved		IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.	AFIO EN
	rw		rw	rw	rw	rw			rw	rw	rw	rw	rw		rw

그림 3. APB2 peripheral clock enable register(Reference Manual p.146)

따라서 RCC의 Base Address에 Offset을 더한 0x4002 1018이 APB2의 Clock와 연관된 레지스트리 주소이므로 이를 이용하여 APB2의 Clock을 다음과 같이 매크로로 정의하였다.

```
#define RCC_APB2ENR_CLK *((volatile unsigned int *) 0x40021018)
```

그림 4. APB2의 Clock Enable Register 매크로 선언문

이어서 KEY와 LED를 제어하는 레지스터의 주소를 파악해야 한다. 아래는 제공된 파일 중 KEY(Button)와 LED의 Schematic 구조를 나타낸 것이다. 이를 통해 Key들은 각각 PC4, PB10, PC13, PA0(순서대로 Key1, Key2, Key3, Key4) 핀을 사용하고 있고, LED는 모두 PD2, PD3, PD4, PD7(순서대로 LED1, LED2, LED3, LED4) 핀을 사용하고 있다는 것을 알 수 있다. 따라서 Key를 제어하기 위해서는 GPIOA, GPIOB, GPIOC, LED를 제어하기 위해 GPIOD 레지스터를 사용해야한다.

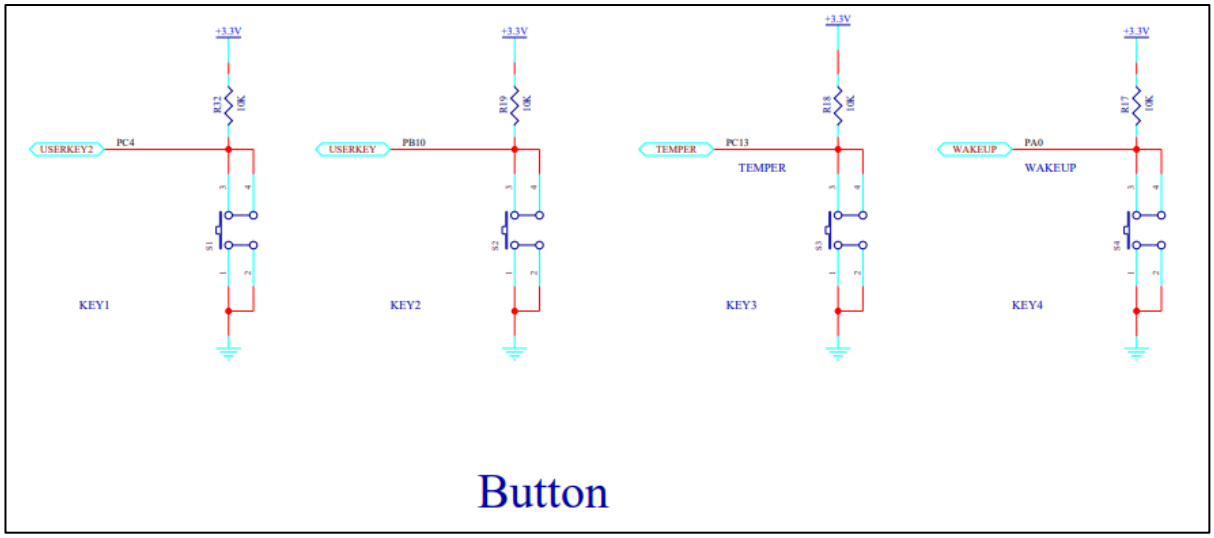

그림 5. KEY의 Schematic 구조(STM32_Schematic p.3)

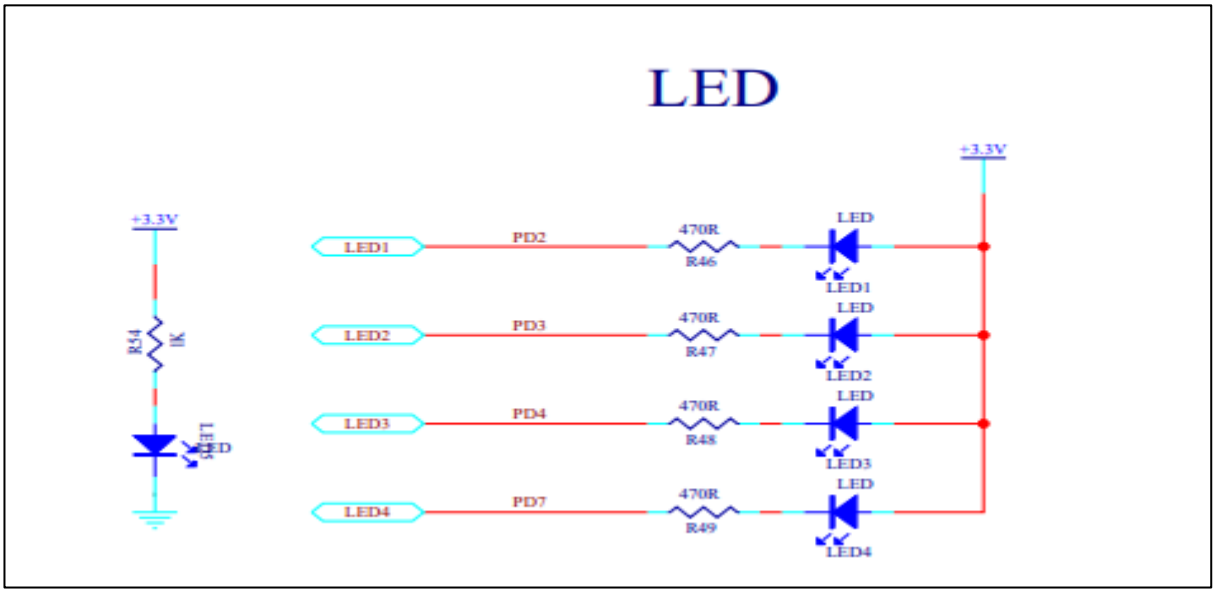

그림 6. LED의 Schematic 구조(STM32_Schematic p.3)

GPIOA, GPIOB, GPIOC, GPIOD의 Base Address는 RCC와 마찬가지로 Datasheet 파일을 참고하여 찾을 수 있다.

APB2	Reserved	0x4001 1C00 - 0x4001 23FF
	Port E	0x4001 1800 - 0x4001 1BFF
	Port D	0x4001 1400 - 0x4001 17FF
	Port C	0x4001 1000 - 0x4001 13FF
	Port B	0x4001 0C00 - 0x4001 0FFF
	Port A	0x4001 0800 - 0x4001 0BFF
	EXTI	0x4001 0400 - 0x4001 07FF
	AFIO	0x4001 0000 - 0x4001 3FFF

그림 7. APB2 Port A, B, C, D의 Base Register 주소(stm32_Datasheet p.33)

GPIO 레지스터의 Offset은 low인지 high인지에 따라 나뉘는데 해당 포트의 low를 사용한다면 0x00을 더해주고 high를 사용한다면 0x04를 더해주어야 한다. 이를 통해 아래와 같이 GPIOA, GPIOB, GPIOC, GPIOD의 레지스터를 매크로로 정의할 수 있다.

그림 8. Port configuration reigster low(Reference Manual p.171)

그림 9. Port configuration reigster high(Reference Manual p.172)

```
#define GPIOA_CRL *((volatile unsigned int *) 0x40010800) // PA0
#define GPIOB_CRH *((volatile unsigned int *) 0x40010C04) // PB10
#define GPIOC_CRL *((volatile unsigned int *) 0x40011000) // PC4
#define GPIOC_CRH *((volatile unsigned int *) 0x40011004) // PC14
#define GPIOD_CRL *((volatile unsigned int *) 0x40011400) // LED 모드 설정 address
```

그림 10. GPIOA, GPIOB, GPIOC, GPIOD의 Configuration Register들에 대한 매크로 선언문

그리고 데이터를 읽고 쓰기 위한 레지스터도 필요한데 이번 실험에서는 KEY의 입력으로 LED를 제어하므로 KEY는 입력, LED는 출력으로 사용해야 한다. 따라서 GPIOA, GPIOB, GPIOC는 input, GPIOD는 output으로 사용해야 한다. GPIO의 input data register의 Offset은 0x08이고, GPIO의 output은 Port bit set/reset register와 Port bit reset register를 사용할 것이며, Offset은 0x10, 0x14이다. output에서 두 가지 레지스터를 사용하는 이유는 KEY를 누르는 행위로 LED를 키고 끄는 두 가지 동작을 수행하기 때문에 키는 동작에서는 BRR, 끄는 동작에서는 BSRR을 사용한다.

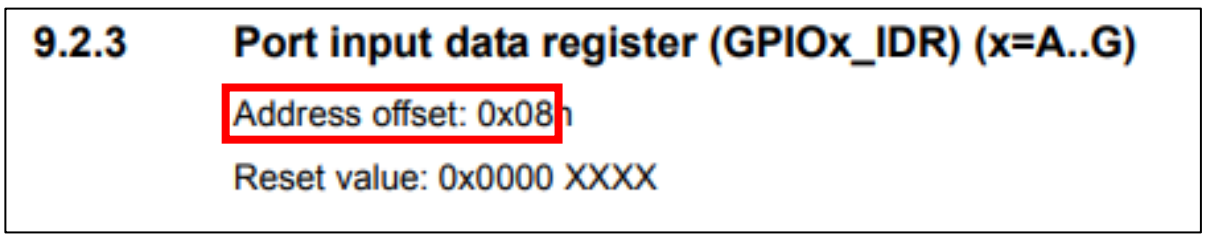

그림 11. GPIO의 Port Input Data Register(Reference Manual p.172)

그림 12. GPIO의 Port Bit Set/Reset Register(Reference Manual p.173)

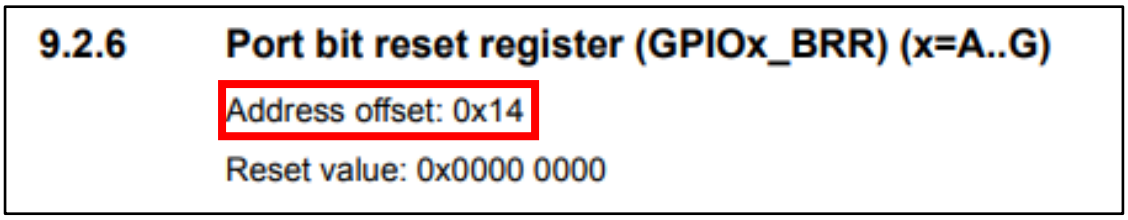

그림 13. GPIO의 Port bit reset Register(Reference Manual p.174)

이를 통해 아래와 같이 GPIOA, GPIOB, GPIOC의 Input Register와 GPIOD의 BSRR, BRR을 매크로로 정의할 수 있다.

```
// 버튼 눌렸는지 확인용 주소 (PA0, PB10, PC4, PC14)
#define GPIOA_IDR *((volatile unsigned int *) 0x40010808)
#define GPIOB_IDR *((volatile unsigned int *) 0x40010C08)
#define GPIOC_IDR *((volatile unsigned int *) 0x40011008)
#define GPIOC_H_IDR *((volatile unsigned int *) 0x4401140C)
#define GPIOD_IDR *((volatile unsigned int *) 0x40011408)

#define GPIOD_BSRR *(volatile unsigned int *)0x40011410 // P
#define GPIOD_BRR *((volatile unsigned int *) 0x40011414) //
```

그림 14. GPIO(A, B, C) Input Data Register와 GPIOD Bit Set/Reset Register, Bit reset Register에 대한 매크로 선언문

레지스터의 주소를 매크로로 정의했으므로 APB2의 RCC Clock을 Port A, B, C와 D에 대해 활성화를 시켜주어야 한다. 아래 그림을 참고하면, GPIOA, GPIOB, GPIOC와 GPIOD를 사용하기 위해선 각각 2, 3, 4, 5번 비트를 활성화해주어야 한다. 따라서 위에서 정의해준 RCC_APB2ENR_CLK의 값을 2, 3, 4, 5번 비트가 1인 값으로 설정해야 하며, 이 값을 16진수로 표현하면 0x3C가 된다.

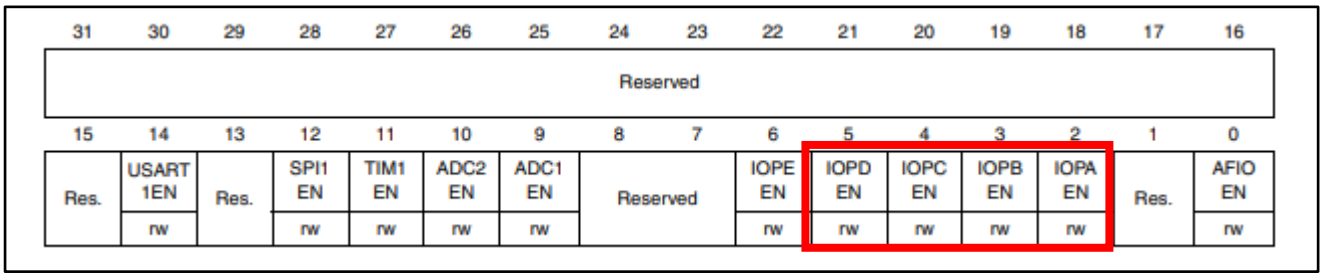

그림 15. APB2의 Peripheral Clock Enable Register의 비트 구성도(Reference Manual p.146)

```
RCC_APB2ENR_CLK |= 0x3C; // IOPAEN, IOPBEN, IOPCN, IOPDEN 각 bit에 1을 할당
```

그림 16. GPIOx(A, B, C, D)의 Clock을 활성화하는 코드

KEY에 해당하는 GPIOx(A, B, C)의 경우 차례대로 PC4, PB10, PC13, PA0 핀을 사용하는데 먼저 해당 핀에 대한 위치를 사용하기 위해 해당 핀의 자리를 제외한 곳은 F를 나머지는 0인 값을 &연산해준다. 이후 y번 핀을 Input으로 사용하기 위해 CNFy를 10으로, MODEy는 00으로 설정해줄 것이다. 이

를 16진수로 표현하면 8이 되는데, 각각 4, 10, 13, 0번 포트들만 값을 8로 설정해주어야 하기 때문에 | 연산을 해주었다. 이때 GPIOC의 경우 4번 핀과 13번 핀은 LOW와 HIGH로 나뉘어져 있기 때문에 이를 구분해서 설정해준다.

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 17. Port Configuration Register Low의 비트 구성도(Reference Manual p.171)

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 18. Port Configuration Register High의 비트 구성도(Reference Manual p.172)

```
// register address reset
GPIOA_CRL &= ~0x0000000F;
GPIOB_CRH &= ~0x00000F00;
GPIOC_CRL &= ~0x000F0000;
GPIOC_CRH &= ~0x00F00000;

// Use KEY1 = PC4 | KEY2 = PB10 | KEY3 = PC13 | KEY4 = PA0
GPIOA_CRL |= 0x00000008;
GPIOB_CRH |= 0x00000800;
GPIOC_CRL |= 0x00080000;
GPIOC_CRH |= 0x00800000;
```

그림 19. GPIOx(A, B, C)의 4, 10, 13, 0번 핀을 Read Mode로 활성화하는 코드

LED에 해당하는 GPIOD의 경우 Port D의 2, 3, 4, 7번 핀을 사용하는데 Input과 동일하게 먼저 해당 핀에 대한 위치를 사용하기 위해 해당 핀의 자리를 제외한 곳은 F를 나머지는 0인 값을

&연산해준다. 이후 y번 핀을 Output으로 사용하기 위해 CNFy를 00으로, MODEy를 01으로 설정해줄 것이다. 이를 16진수로 표현하면 1이 되므로 해당 위치들에 1을 | 연산해준다. 그리고 코드를 실행했을 때 LED는 꺼져있는 상태로 시작하기 위해서 BSRR을 0x10011100으로 초기화 해주어야 하는데 이를 16진수로 표현하면 9C이다.

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 20. Port Configuration Register Low의 비트 구성도

```
// Use PD 2, 3, 4, 7
// register address reset
GPIOD_CRL &= ~0xF00FFF00;

// LED(1, 2, 3, 4)연결
GPIOD_CRL |= 0x10011100;

// initial LED state -> off
GPIOD_BSRR |= 0x9C;
```

그림 21. GPIOD의 2, 3, 4, 7번 핀을 Write Mode로 활성화 후 OFF상태로 만드는 코드

KEY를 입력하면 해당 LED를 켜주는 코드를 작성해야 하는데 LED가 꺼져 있는 상태면 키고 켜져 있는 상태면 꺼져야 하므로 LED의 상태를 확인하는 변수를 선언해준다.

```
// LED의 상태 확인용 변수
int flag1 = 0;
int flag2 = 0;
int flag3 = 0;
int flag4 = 0;
```

그림 22. 각 LED의 상태를 확인하기위한 변수 선언문

기본적으로 원하는 작동을 전원이 꺼지기 전까지 수행해야 하기 때문에 while(1)문 안에 코드를 작성하였다. 각 KEY가 눌렸는지 확인하기 위해 if문의 조건식에 각각의 IDR의 NOT에 해당

KEY의 핀 값을 &연산으로 확인하였다. IDR은 눌릴 때 0이 되기 때문이다. 그 다음으론 flag 값을 확인해 꺼져 있으면 BRR을 이용해 해당 LED의 핀 값으로 1을 shift해준 값을 | 연산해 reset 해주면 LED가 켜지고 flag 값을 1로 바꿔주었다. 반대로 켜져 있는 경우 BSRR을 이용해 1을 SHIFT 해준 값을 | 연산해 set해주면 LED가 꺼지고 flag를 0으로 바꿔주었다.

```
if (~GPIOC_IDR & 0x00000010) { // Key1의 버튼이 눌렸는지 확인. (IDR은 눌렸으면 0, 뗐으면 1)
    if (flag1 == 0) {
        GPIOD_BRR |= (1 << 2); // LED 켜기 (reset)
        flag1 = 1;
    }
    else if (flag1 == 1) {
        GPIOD_BSRR |= (1 << 2); // LED 끄기 (set)
        flag1 = 0;
    }
}
```

그림 23. KEY1을 입력했을 때 LED1이 동작하는 코드

KEY의 입력을 통해 LED를 제어하는 반복문은 아래와 같다.

```
while (1) // 계속 반복
{
    if (~GPIOC_IDR & 0x00000010) { // Key1의 버튼이 눌렸는지 확인. (IDR은 눌렸으면 0, 뗐으면 1)
        if (flag1 == 0) {
            GPIOD_BRR |= (1 << 2); // LED 켜기 (reset)
            flag1 = 1;
        }
        else if (flag1 == 1) {
            GPIOD_BSRR |= (1 << 2); // LED 끄기 (set)
            flag1 = 0;
        }
    }
    if (~GPIOB_IDR & 0x400) {
        if (flag2 == 0) {
            GPIOD_BRR |= (1 << 3);
            flag2 = 1;
        }
        else if (flag2 == 1) {
            GPIOD_BSRR |= (1 << 3);
            flag2 = 0;
        }
    }
    if (~GPIOC_IDR & 0x2000) {
        if (flag3 == 0) {
            GPIOD_BRR |= (1 << 4);
            flag3 = 1;
        }
        else if (flag3 == 1) {
            GPIOD_BSRR |= (1 << 4);
            flag3 = 0;
        }
    }
    if (~GPIOA_IDR & 0x1) {
        if (flag4 == 0) {
            GPIOD_BRR |= (1 << 7);
            flag4 = 1;
        }
        else if (flag4 == 1) {
            GPIOD_BSRR |= (1 << 7);
            flag4 = 0;
        }

    }
}
```

그림 24. KEY의 입력으로 LED 제어 반복문 코드

아래는 해당 코드를 통해 이번 프로젝트를 수행한 결과 사진이다.

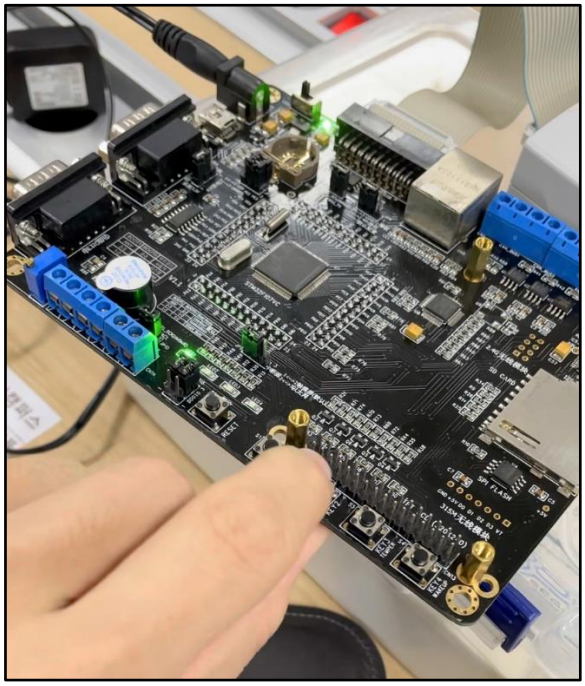

그림 25. KEY1을 입력한 사진

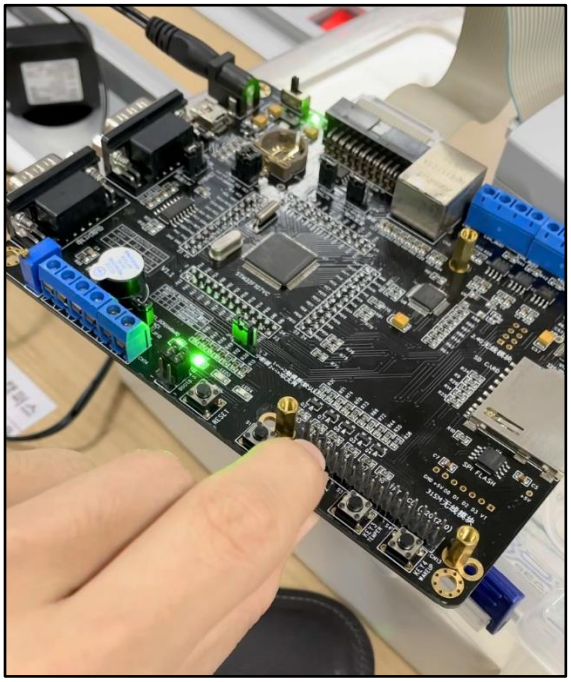

그림 26. KEY2을 입력한 사진

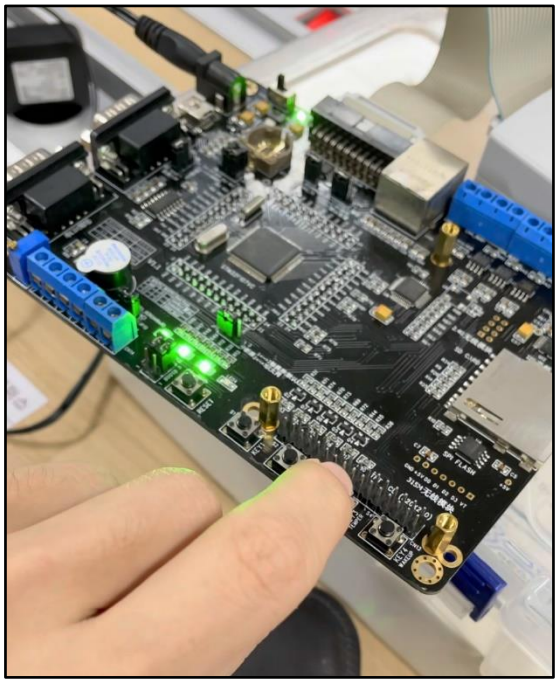

그림 27. KEY3을 입력한 사진

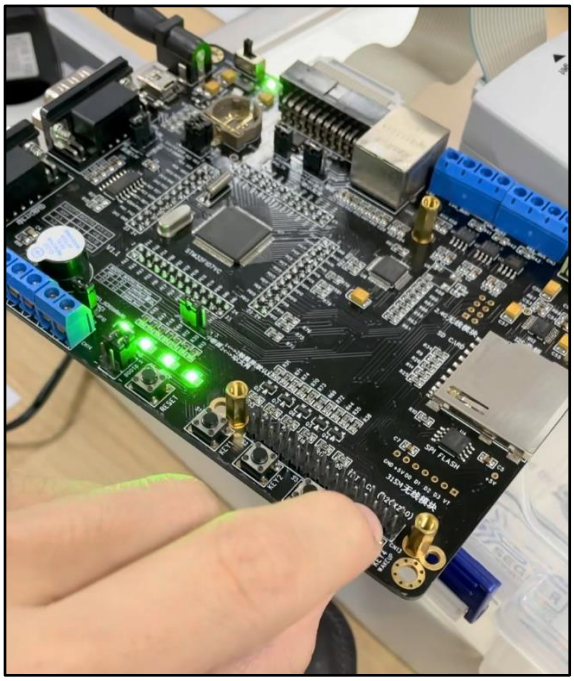

그림 28. KEY4을 입력한 사진

그리고 SMT32를 오실로스코프에 연결했는데 KEYn(1, 2, 3, 4)를 각각 dn(1, 2, 3, 4)에 연결하였다. 이는 KEY값을 입력했을 때 LED가 켜졌다가 꺼진 시간을 측정하기 위해서이다.

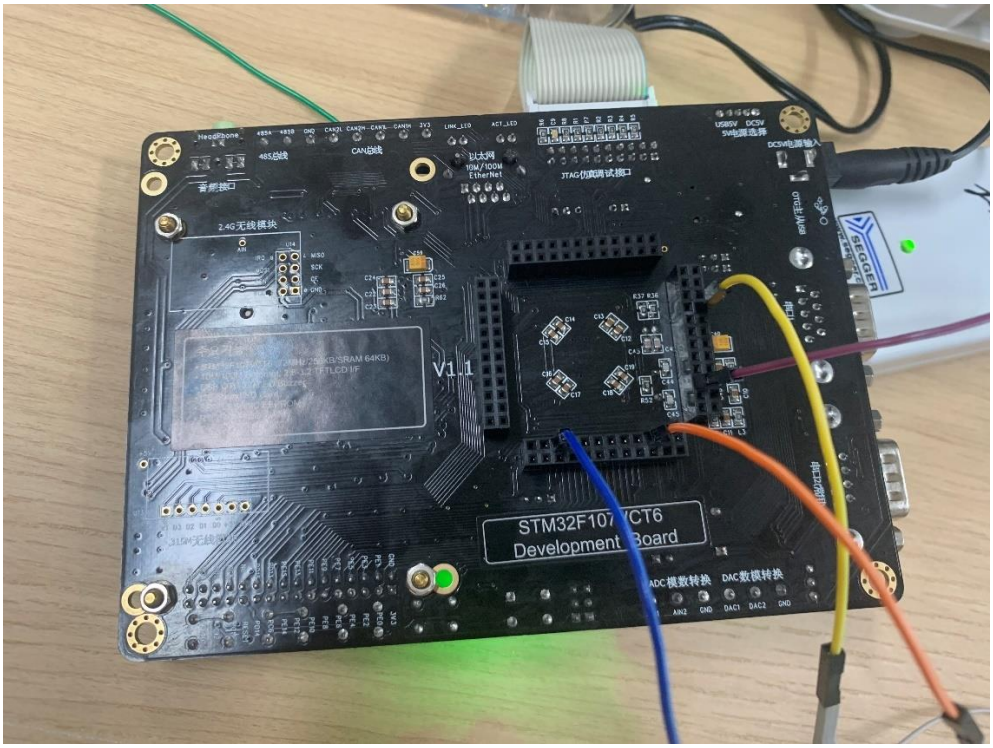

그림 29. SMT32의 KEY에 해당하는 포트들을 오실로스코프와 연결한 사진

오실로스코프에 연결해 KEY를 두 번 입력 후 measure을 이용해 LED가 켜졌다가 꺼진 시간을 측정하는데 cursor를 이용해 (입력으로 불이 꺼진 시간) – (입력으로 불이 켜진 시간)으로 정확히 측정할 수 있었다.

그림 30. 오실로스코프의 measure로 시간을 측정한 사진

각 시간을 표로 정리해보면 각 LED들이 켜져 있던 시간은 LED1 = 888ms, LED2 = 872ms, LED3 = 852ms, LED4 = 848ms 가 나오게 된다.

단위 ms / KEY	LED가 켜진 시점	LED가 꺼진 시점	(꺼진 시점) – (켜진 시점)
D4(KEY1)	-726	162	888
D3(KEY2)	-858	14	872
D2(KEY3)	-840	12	852
D1(KEY4)	-682	166	848

표 1. KEY들의 켜진 시점과 꺼진 시점과 그 차