3주차 결과보고서

10조 (001분반)
과목명:임베디드시스템설계및실험
담당 교수 : 정상화교수님
담당 조교 : 최호진 조교님
조원 : 201924603 하규승(조장)
201727102 강준혁
201924525 이광훈
202023139 박지원
제출 날짜 : 2023.09.24

목차

목표

1. 임베디드 시스템의 기본 원리 습득
2. 레지스터와 주소 제어를 통한 임베디드 펌웨어 개발 이해

세부 실험 내용

1. Datasheet 및 Refence Manual을 참고하여 해당 레지스터 및 주소에 대한 설정 이해
2. 버튼을 이용한 LED 제어
   KEY1: PD2, PD4 LED On
   KEY2: PD2, PD4 LED Off
   KEY3: PD3, PD7 LED On
   KEY4: PD3, PD7 LED Off
3. 정상적인 동작 유무 확인
4. 오실로스코프를 이용한 디버깅

(버튼의 값 변화에 따른 LED 점등까지의 시간 측정, Digital Pin의 Trigger를 이용하여 캡쳐)

목표

1. 임베디드 시스템의 기본원리 습득

임베디드 시스템(Embedded System)은 특정한 작업을 수행하기 위해 설계된 전용 컴퓨터 시스템입니다. 이러한 시스템은 주로 다음과 같은 특징을 갖고 있습니다.

특정한 기능 수행: 임베디드 시스템은 특정한 작업 또는 기능을 수행하기 위해 설계됩니다. 이러한 작업은 주로 제어, 모니터링, 데이터 수집 및 처리와 같은 것들이며, 예를 들어 자동차의 엔진 제어 시스템, 스마트폰의 센서 모니터링, 가전제품의 제어 등이 있습니다.

하드웨어와 소프트웨어 통합: 임베디드 시스템은 특정한 하드웨어와 소프트웨어의 조합으로 구성됩니다. 하드웨어는 주로 마이크로컨트롤러 또는 마이크로프로세서와 같은 특수한 칩으로 이루어져 있으며, 소프트웨어는 이 하드웨어를 제어하고 원하는 작업을 수행하기 위해 프로그래밍됩니다.

실시간 동작: 많은 임베디드 시스템은 실시간 시스템으로서 동작합니다. 즉, 시스템은 특정한 시간 내에 정확한 응답을 제공해야 합니다. 이는 자동차의 에어백 시스템이나 의료 기기와 같은 응용 분야에서 중요합니다.

제한된 자원: 임베디드 시스템은 종종 제한된 자원을 가집니다. 이는 제한된 프로세서 속도, 메모리, 저장 공간 및 전력 등을 의미합니다. 따라서 소프트웨어 개발자는 자원을 효율적으로 활용해야 합니다.

안정성 및 신뢰성: 많은 임베디드 시스템은 안전 및 신뢰성이 중요합니다. 실수 또는 결함이 큰 문제를 발생시킬 수 있으므로 신뢰성 있는 동작이 요구됩니다.

특화된 운영 체제: 일부 임베디드 시스템은 특화된 운영 체제를 사용합니다. 이 운영 체제는 작은 크기로 최적화되어 있으며 실시간 동작을 지원합니다.

임베디드 시스템은 우리 주변의 다양한 기기와 시스템에서 찾아볼 수 있으며, 이러한 시스템은 우리 일상생활의 여러 측면을 향상시키고 자동화하는 데 중요한 역할을 합니다.

2. 레지스터와 주소 제어를 통한 임베디드 펌웨어 개발 이해

임베디드 펌웨어 개발에서 레지스터(register)와 주소 제어(address control)는 중요한 개념입니다. 이러한 요소들은 임베디드 시스템의 하드웨어를 제어하고 프로그래밍하는 데 필수적입니다.

레지스터(Register):

레지스터는 CPU 내부에 있는 작은 메모리 저장 공간으로, 데이터나 명령을 일시적으로 저장하는 데 사용됩니다. 레지스터는 매우 빠른 읽기와 쓰기 속도를 가지며, CPU가 연산을 수행하는 데 필수적입니다. 임베디드 시스템에서는 레지스터를 사용하여 데이터를 저장하고 연산을 수행하며, 성능 향상을 위해 주로 저수준 언어인 어셈블리 언어 또는 C와 같은 언어로 프로그래밍됩니다. 특정 레지스터는 특정한 목적을 위해 설계되어 있으며, 예를 들어 데이터 레지스터, 주소 레지스터, 제어 레지스터 등이 있습니다. 레지스터는 주로 레지스터 맵(register map)이라고 불리는 하드웨어 문서에서 정의되며, 프로그래머는 이 맵을 참조하여 레지스터를 사용합니다.

주소 제어(Address Control):

임베디드 시스템에서 주소 제어는 메모리와 입출력 장치 등의 하드웨어에 대한 접근을 관리하고 제어하는 것을 의미합니다.

메모리나 입출력 장치는 각각 고유한 주소 범위를 가지며, 이 주소 범위를 사용하여 해당 장치에 데이터를 읽고 쓸 수 있습니다. 주소 제어는 메모리 매핑(memory mapping)과 관련이 있으며, 메모리 주소와 하드웨어 주소 간의 매핑을 설정하고 관리합니다. 또한, 입출력 장치에 대한 주소와 데이터를 전달하기 위해 입출력 포트(IO port) 또는 주소 버스(address bus)를 사용할 수 있습니다. 주소 제어를 통해 특정 주소에 데이터를 읽고 쓰는 것은 레지스터와 연결되며, 이를 통해 하드웨어와 상호작용할 수 있습니다. 임베디드 펌웨어 개발에서 레지스터와 주소 제어는 하드웨어와 소프트웨어 간의 효율적인 통신과 상호작용을 위한 중요한 도구입니다. 개발자는 레지스터와 주소 제어를 이용하여 하드웨어를 제어하고 임베디드 시스템의 기능을 프로그래밍적으로 구현합니다. 이를 통해 특정한 작업을 수행하는 임베디드 시스템을 개발하고 제어할 수 있습니다.

세부 실험 내용

1. Datasheet 및 Refence Manual을 참고하여 해당 레지스터 및 주소에 대한 설정 이해

APB2	Reserved	0x4001 3C00 - 0x4001 FFFF
	USART1	0x4001 3800 - 0x4001 3BFF
	Reserved	0x4001 3400 - 0x4001 37FF
	SPI1	0x4001 3000 - 0x4001 33FF
	TIM1	0x4001 2C00 - 0x4001 2FFF
	ADC2	0x4001 2800 - 0x4001 2BFF
	ADC1	0x4001 2400 - 0x4001 27FF
	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 1 Datasheet의 APB2 레지스터의 주소값

위 그림1은 DataSheet의 APB2 레지스터의 주소값을 나타낸다. 예를 들어 Port A를 사용하고 싶으면 기본 값인 0x4001 0800을 사용하면 됩니다. 그 전에 RCC레지스터로 사용을 원하는 레지스터에 클럭을 인가해 주어야 하는데 해당 기능을 하는 레지스터는 RCC_APB2ENR 레지스터입니다. 해당 레지스터 reference는 아래 그림2에 나와있습니다.

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.

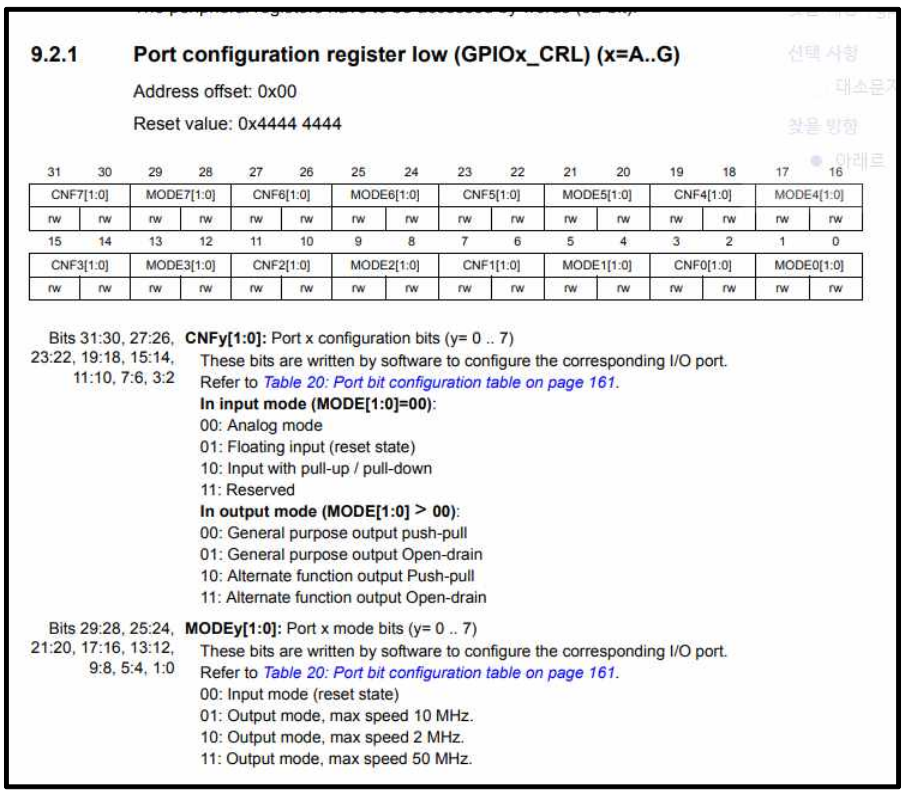
Note: When the peripheral clock is not active, the peripheral register values may not be readable by software and the returned value is always 0x0.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved										TIM11 EN	TIM10 EN	TIM9 EN	Reserved		
										rw	rw	rw			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADC3 EN	USART 1EN	TIM8 EN	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	IOPG EN	IOPF EN	IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.	AFIO EN
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw

그림 2 각 GPIO PIN의 클럭인가

위 그림에서 Reset value가 의미하는 것은 RCC레지스터의 주소값이고, Address는 RCC_APB2ENR 레지스터의 주소값입니다. RCC_APB2ENR 레지스터는 기본 RCC 레지스터에 offset값 0x18을 더한 값이기 때문에 RCC_APB2ENR 레지스터의 주소값은 0x40020018이 됩니다. 레지스터의 단위

가 16진수이고 4비트씩 나누어져 있기 때문에 만약 GPIOA핀에 클럭을 인가하고 싶으면 주소값에 역참조를 통해 합연산으로 해당 핀에 해당하는 비트값을 더해주면 됩니다. 표현하자면 *((volatile unsigned int*) 0x40020018) |= 0x4;로 표현하면 A pin에 클럭이 인가가 됩니다. 이번 실험에서는 A,B,C,D pin이 사용되기 때문에 0x0000003C를 합연산 해주면 됩니다.



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

Bits 31:30, 27:26, 23:22, 19:18, 15:14, 11:10, 7:6, 3:2 **CNFy[1:0]:** Port x configuration bits (y= 0 .. 7)
These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table on page 161.*
In input mode (MODE[1:0]=00):
00: Analog mode
01: Floating input (reset state)
10: Input with pull-up / pull-down
11: Reserved
In output mode (MODE[1:0] > 00):
00: General purpose output push-pull
01: General purpose output Open-drain
10: Alternate function output Push-pull
11: Alternate function output Open-drain

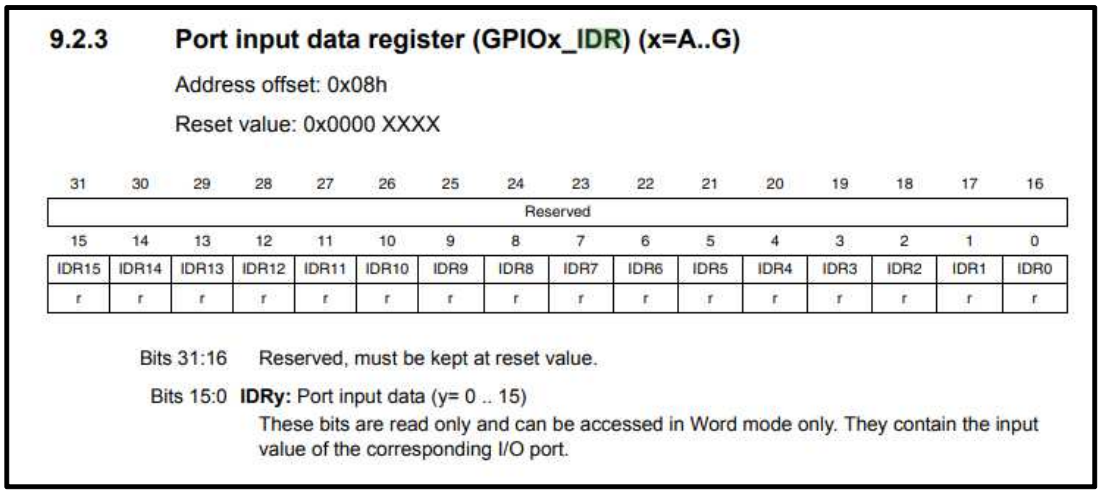
Bits 29:28, 25:24, 21:20, 17:16, 13:12, 9:8, 5:4, 1:0 **MODEy[1:0]:** Port x mode bits (y= 0 .. 7)
These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table on page 161.*
00: Input mode (reset state)
01: Output mode, max speed 10 MHz.
10: Output mode, max speed 2 MHz.
11: Output mode, max speed 50 MHz.

그림 3 원하는 핀 번호사용을 위한 레지스터 주소

위 그림의 GPIOx_CRL은 원하는 핀 번호 모드를 Input이냐, Output인지를 설정해줄 수 있는 레지스터입니다. GPIOx_CRL은 4비트씩 끊어 읽으며, 이번 실험에서 Input모드로 설정하기 위해서 CNFy에서 Input with pull-up/pull-down모드와 MODEy에서 Input mode(reset state)로 설정하여 0번핀을 사용할 때 GPIOx_CRL의 값에서 0번 핀을 리셋 해주고 0x8을 합연산 해 주었습니다. 코드상으로는 아래 그림과 같이 표현 해 주었습니다.

```
GPIOA_CRL &= 0xFFFFFFF0; // reset
GPIOA_CRL |= 0x00000008; // switch PA 0 input mode
GPIOA_IDR |= 0x00000000; // switch Input Data Register reset
```

그림 4 PA0핀 Input mode 설정



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 5 GPIOx_IDR의 reference

그림 4에서 GPIOA_IDR이 있는데 IDR은 해당 핀에 전압이 가해졌을 때 GPIOA_IDR의 값이 해당 핀에 해당하는 핀에 전압이 가해진다는 뜻입니다.

```
GPIOD_CRL &= 0x0FF000FF; // reset
GPIOD_CRL |= 0x30033300; // led PD 2,3,4,7 output push-pull

GPIOD_BSRR |= 0x0000009C;
```

그림 6 PD핀 Output mode 설정

마지막으로 Output설정과 BSRR에 대해 설명하겠습니다. Output 설정은 그림3에서 나와 있듯이, CNFy에서 General purpose output Push/pull 모드와 MODEy에서 Output mode, max speed 50Mhz로 설정하였고, 2,3,4,7번 핀을 사용하기에 GPIOD_CRL의 값에서 2,3,4,7번 핀을 리셋 해주고 0x3033300을 합연산 해 주었습니다. 그리고 BSRR에 대한 설명을 이어가자면, 아래 그림 7에서 Portx Set bit, Portx Reset bit라는는 문구와 설명이 있는데 간단하게 설명하면, 0~15비트는 해당 핀에 전압을 가하는 것이고, 16~31비트는 해당 핀의 전압을 가하지 않는 것입니다.

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

Bits 31:16 **BRy:** Port x Reset *bit y (y= 0 .. 15)*
These bits are write-only and can be accessed in Word mode only.
0: No action on the corresponding ODRx bit
1: Reset the corresponding ODRx bit
Note: If both BSx and BRx are set, BSx has priority.

Bits 15:0 **BSy:** Port x *Set bit y (y= 0 .. 15)*
These bits are write-only and can be accessed in Word mode only.
0: No action on the corresponding ODRx bit
1: Set the corresponding ODRx bit

그림 7 GPIOx_BSRR의 reference

2. 버튼을 이용한 LED 제어

KEY1: PD2, PD4 LED On
KEY2: PD2, PD4 LED Off
KEY3: PD3, PD7 LED On
KEY4: PD3, PD7 LED Off

```
#define RCC_APB2ENR *((volatile unsigned int *)0x40021018)
```

먼저 앞서 설명했던 것처럼 RCC 레지스터로 사용을 원하는 레지스터에 클럭을 인가해주기 위해 해당 기능을 하는 RCC_APB2ENR 레지스터를 선언해주었습니다. 그림2를 통해 RCC_APB2ENR 레지스터는 기본 RCC 레지스터의 offset 0x18을 더한 값임을 알 수 있으며 0x40020000+0x18 = 0x40020018로 선언해주었습니다.

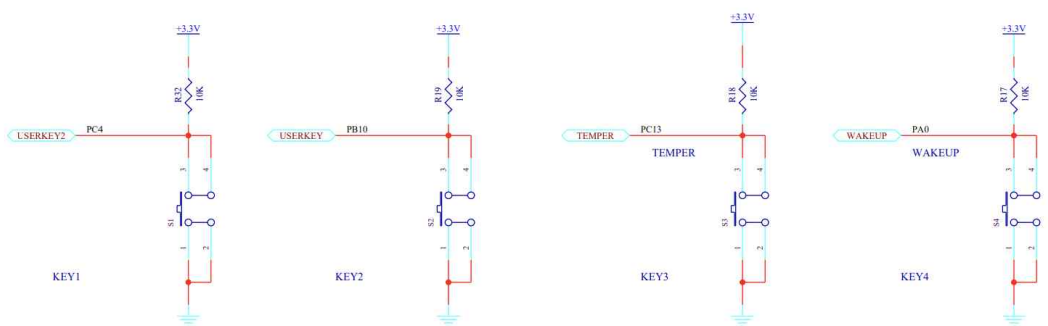


그림 8 각 버튼들의 schematic

그리고 KEY1, KEY2, KEY3, KEY4 버튼은 그림8을 통해 각각 PC4, PB10, PC13, PA0과 연결되어있는 것을 알 수 있습니다. 그에 따라 포트 A, B, C, D를 사용하게 되므로 포트에 맞는 주소값을 그림1에서 찾아준 후 계산하였습니다.

```
#define GPIOA_CRL *(volatile unsigned int *)0x40010800 //switch Co
#define GPIOA_IDR *(volatile unsigned int *)0x40010808 //switch In
```

먼저 KEY4가 PA0과 연결되어있으므로 Port A의 주소인 0x40010800에 CRL(0~7이므로 Low)의 address offset인 0x00, IDR의 address offset인 0x08를 더하여 선언해주었습니다.

```
#define GPIOB_CRH *(volatile unsigned int *)0x40010C04 //switch Co
#define GPIOB_IDR *(volatile unsigned int *)0x40010C08 //switch In
```

KEY2가 PB10과 연결되어있으므로 Port B의 주소인 0x40010C00에 CRH(8~15이므로 High)의 address offset인 0x04, IDR의 address offset인 0x08를 더하여 선언해주었습니다.

```
#define GPIOC_CRL *(volatile unsigned int *)0x40011000 //switch C
#define GPIOC_CRH *(volatile unsigned int *)0x40011004 //switchCo
#define GPIOC_IDR *(volatile unsigned int *)0x40011008 //switch I
```

KEY1과 KEY3이 PC4, PC13과 연결되어있으므로 마찬가지로 Port C의 주소인 0x40011000에 CRL의 address offset인 0x00, CRH(8~15이므로 High)의 address offset인 0x04, IDR의 address offset인 0x08를 더하여 선언해주었습니다.

```
#define GPIOD_CRL *(volatile unsigned int *)0x40011400 //LED Config
#define GPIOD_BSRR *(volatile unsigned int *)0x40011410 //LED Bit S
```

PD2, PD3, PD4, PD7에 출력값 또한 전달해야하므로 Port D의 주소값인 0x40011400에 CRL의 address offset인 0x00, BSRR의 address offset인 0x10을 더하여 선언해주었습니다.

이제 각각의 led를 켜기 위한 함수를 다음과 같이 선언해주었습니다.

```
void led2on() { //only pd 2 led off
  GPIOD_BSRR |= 0x00040000;
}
void led3on() { //only pd 3 led off
  GPIOD_BSRR |= 0x00080000;
}
void led4on() { //only pd 4 led off
  GPIOD_BSRR |= 0x00100000;
}
void led7on() { //only pd 7 led off
  GPIOD_BSRR |= 0x00800000;
}
```

이때 PD2, PD3, PD4, PD7에서의 각 포트의 set bit과 reset bit은 그림7을 참고하여 Reset bit는 31:16, Set bit는 15:0임을 고려하면 다음과 같습니다.
(PD)2 = 0b0100 = 0x4 / 0x00040000
(PD)3 = 0b1000 = 0x8 / 0x00080000
(PD)4 = 0b00010000 = 0x10 / 0x00100000
(PD)7 = 0b10000000 = 0x80 / 0x00800000
이 값들을 |=연산으로 각 주소에 집어넣어 led를 제어하는 신호를 발생시킵니다.

마찬가지로 각각의 led를 끄기 위한 함수를 다음과 같이 선언해주었습니다.

```
void led2off() { //only pd 2 led on
  GPIOD_BSRR |= 0x00000004;
}
void led3off() { //only pd 3 led on
  GPIOD_BSRR |= 0x00000008;
}
void led4off() { //only pd 4 led on
  GPIOD_BSRR |= 0x00000010;
}
void led7off() { //only pd 7 led on
  GPIOD_BSRR |= 0x00000080;
}
```

main함수로 들어가서 먼저 사용할 핀들을 reset시켜주었습니다.
이때 port A, B, C, D만 clock을 주어 사용할 것이므로 reset하기 위해

0b0000000000011100 = 0x3C 값을 입력해주었습니다.

```
int main(void){

  RCC_APB2ENR |=  0x0000003C;
```

```
  GPIOA_CRL &= 0xFFFFFFF0; // reset
  GPIOA_CRL |= 0x00000008; // switch PA 0 input mode
  GPIOA_IDR |= 0x00000000; // switch Input Data Register reset
```

1번에서 설명했던 것과 같이 Port A는 Input모드로 설정하기 위해서 그림3을 참고하여 CNFy에서 Input with pull-up/ pull-down모드와 MODEy에서 Input mode(reset state)로 설정하고 0번핀을 사용하므로 GPIOx_CRL의 값에서 0번 핀을 리셋 해주고 0x8을 합연산 해 주었습니다.

```
  GPIOB_CRH &= 0xFFFFF0FF ; // reset
  GPIOB_CRH |= 0x00000800; // switch PB 10 input mode
  GPIOB_IDR |= 0x00000000; // switch Input Data Register r
```

Port B 또한 Input 모드로 사용하기 위해 10번 핀을 리셋하고 0x8을 합연산 해 주었습니다.

```
  GPIOC_CRL &= 0xFF0FFFFF; // reset
  GPIOC_CRL |= 0x00800000; // switch PC 4 input mode
  GPIOC_IDR |= 0x00000000; // switch Input Data Register reset
```

Port C 또한 Input 모드로 사용하기 위해 4번 핀을 리셋하고 0x8을 합연산 해 주었습니다.

```
  GPIOD_CRL &= 0x0FF000FF; // reset
  GPIOD_CRL |= 0x30033300; // led PD 2,3,4,7 output push-pull

  GPIOD_BSRR |= 0x0000009C;
```

Port D는 Ouput 모드로 사용하기 위해 GPIOD_CRL의 값에서 PD2, PD3, PD4, PD7로 사용하게 될 2, 3, 4, 7번 핀을 리셋 해주고 0x3033300을 합연

산 해 주었습니다. BSRR 또한 2, 3, 4, 7번핀을 리셋 하기 위해 각각의 위치에 해당하는 비트로 얻은 0b10011100 -> 0x9C 을 입력해주었습니다.

마지막으로 while문을 통해 계속 반복문을 돌리면서 입력값이 들어오는지 확인하여 입력값을 어떤 값인지 파악하여 원하는 동작을 하도록 하였습니다.

```
while(1){
  if(~GPIOC_IDR & 0x10){    //0100
    led2on();
    led4on();  // switch 1•• •••••• PD2, PD4 LED •• ••••
  }
```

Input값이 KEY1(PC4)인지 확인하기 위해 port C의 IDR인 ~GPIOC_IDR값과 4번핀을 나타내는 값인 0x10와 합연산 하여 확인해주었고 PD2, PD4에 해당하는 led을 켜기 위해 led2on(), led4on()함수를 실행시켜주었습니다.

```
  if(~GPIOB_IDR & 0x400){  //10 0000 0000
    led2off();
    led4off(); //switch 2•• •••••• PD2, PD4 LED •• ••••
  }
```

Input값이 KEY2(PB10)인지 확인하기 위해 port B의 IDR인 ~GPIOB_IDR값과 10번핀을 나타내는 값인 0x400와 합연산 하여 확인해주었고 PD2, PD4에 해당하는 led을 끄기 위해 led2off(), led4off()함수를 실행시켜주었습니다.

```
  if(~GPIOC_IDR & 0x2000) {  //10 0000 0000 0000
    led3on();
    led7on();
  }
```

또한, Input값이 KEY3(PC13)인지 확인하기 위해 port C의 IDR인 ~GPIOC_IDR값과 13번핀을 나타내는 값인0x2000와 합연산 하여 확인해주었고 PD3, PD7에 해당하는 led을 켜기 위해 led3on(), led7on()함수를 실행시켜주었습니다.

```
  if(~GPIOA_IDR & 0x1){  //0000 0001
    led3off();
    led7off();                // switch 4 •••••• PD3, PD7 LED •• ••••
  }
}
```

마지막으로, Input값이 KEY4(PA0)인지 확인하기 위해 port A의 IDR인 ~GPIOA_IDR값과 0번핀을 나타내는 값인 0x01와 합연산 하여 확인해주었고 PD3, PD7에 해당하는 led을 끄기 위해 led2off(), led4off()함수를 실행시켜주었습니다.

3. 정상적인 동작 유무 확인

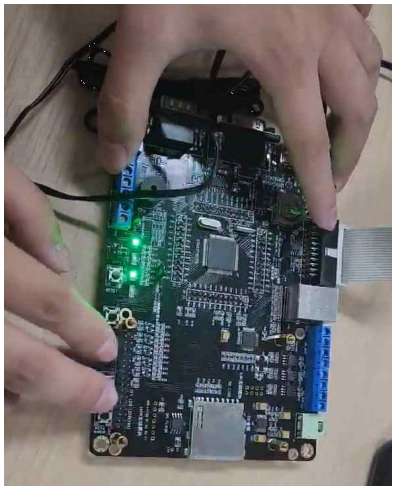


그림 9 KEY1 눌렀을 경우

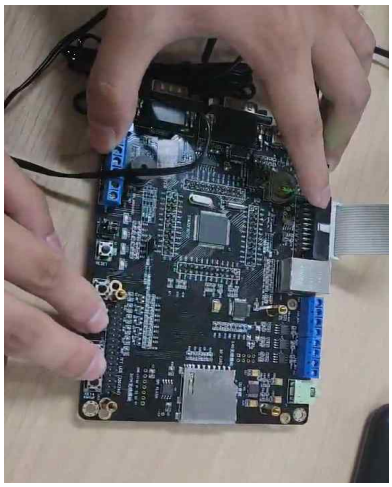


그림 10 KEY2 눌렀을 경우

그림 9, 10를 보면 KEY1을 눌렀을 경우 PD2, PD4에 해당하는 led1, led3이 켜진 것을 확인할 수 있었고 KEY2을 누르면 켜져있던 led1, led3이 꺼지는 것을 확인 할 수 있었습니다.

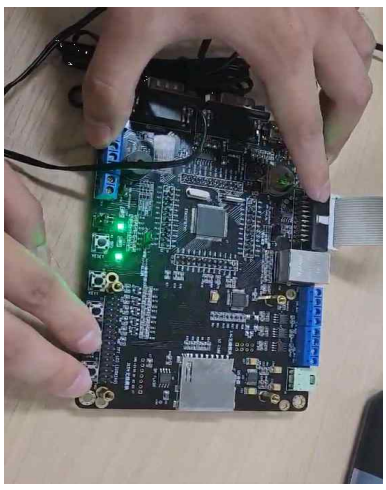


그림 11 KEY3 눌렀을 경우

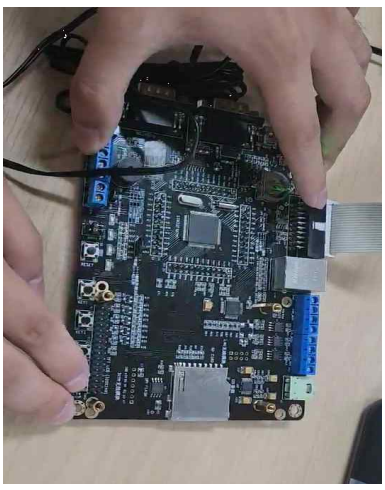


그림 12 KEY4 눌렀을 경우

마찬가지로 그림 11, 12을 보면 KEY3을 눌렀을 경우 PD3, PD7에 해당하는 led2, led4이 켜진 것을 확인할 수 있었고 KEY4을 누르면 켜져있던 led2,

led4이 꺼지는 것을 확인 할 수 있었습니다.

이를 통해 원래의 목표였던 KEY1: PD2, PD4 LED On, KEY2: PD2, PD4 LED Off, KEY3: PD3, PD7 LED On, KEY4: PD3, PD7 LED Off에 대한 정상적인 동작을 하는 것을 확인할 수 있었습니다.

4. 오실로스코프를 이용한 디버깅
(버튼의 값 변화에 따른 LED 점등까지의 시간 측정, Digital Pin의 Trigger를 이용하여 캡쳐)

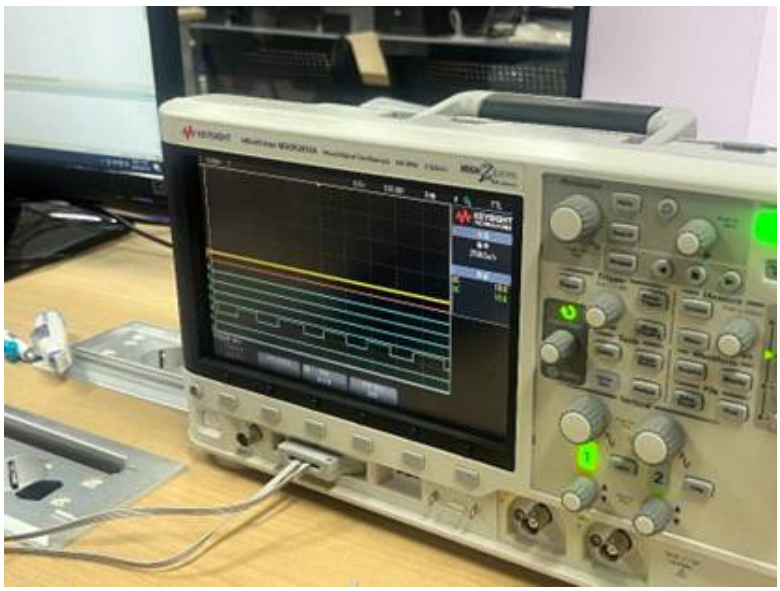


그림 13 오실로스코프

PC4와 GND핀에 오실로스코프를 연결한 후 버튼을 계속해서 눌렀을 경우의 전압변화를 측정해보았습니다. 이 결과는 그림 13과 같으며 KEY1버튼을 눌렀을 경우 전압이 발생해 LED가 켜지도록 신호가 가는 것을 확인할 수 있었으며 프로그램이 정상작동하는 것을 알 수 있었습니다.