임베디드 시스템 설계 및 실험
3주차 실험 결과 보고서

조 : 8조

조원 : 서진욱, 이승민, 하연지, 하태훈

● 실험목표

1. 임베디드 시스템의 기본 원리 습득

- 임베디드 시스템의 기본 원리 및 디버깅 툴 사용방법을 습득하고 레지스터와 주소 제어를 통한 임베디드 펌웨어 개발을 이해해보고자 한다.

2. 레지스터와 주소 제어를 통한 임베디드 펌웨어 개발 이해

- 임베디드 시스템의 기본 원리 및 디버깅 툴 사용방법을 습득하고 레지스터와 주소 제어를 통한 임베디드 펌웨어 개발을 이해해보고자 한다.

● 실험 진행 및 각 과정별 이론

1. 프로젝트 생성 및 각 레지스터 별 memory mapping 한 주소 정의

i) 프로젝트 생성 후 동작 확인

- 바탕화면에 project_test 폴더 생성 후

ii) 각 레지스터 별 memory mapping 한 주소 정의

- **RCC**
  - RCC 초기화를 위한 RCC 레지스터의 메모리 맵핑 주소 : **0x4002 1000**
  - 이 때, APB2 도메인을 사용하고자 하기 때문에 해당하는 Address offset **0x18**을 더함
    ▷ 이에 대한 결과 = RCC의 주소
  - Button과 LED를 사용하고자 하므로 해당 레지스터들이 위치한 A, B, C, D 포트를 활성화해야 함
    ▷ bit 2, bit 3, bit 4, bit 5 가 각각 Port A, B, C, D에 해당하므로 1을 부여
    ▷ 따라서 앞서 정의한 RCC 주소에 0x3C 값을 부여
- **LED1 → PD2, LED2 → PD3, LED3 → PD4, LED4 → PD7**
  - 4개의 LED는 모두 Port D에 위치
  - Port D 레지스터의 메모리 맵핑 주소 : **0x4001 1400**
  - Address offset : **0x00**
    ▷ LED의 주소 : **0x4001 1400**
  - LED 주소에 먼저 Reset Value인 **0x4444 4444** 을 할당
  - LED는 모두 CRL에 해당하는 포트 번호를 가짐
    ▷ 따라서 CRL로써 정의
    ▷ LED는 Output 모드로 동작하므로, CNF는 모두 00(General purpose output push-pull), MODE는 01(Output mode, max speed 10 MHZ)를 할당
    ▷ 따라서 앞서 정의한 CRL 주소에 **0x1001 1100** 을 할당

- LED 조절을 위해 비트 값을 Set 해야 하므로 BSRR을 정의
  - ▷ Port D 레지스터의 메모리 맵핑 주소인 **0x4001 1400**에 set/reset register 설정을 위한 Address offset **0x10**을 더함
  - ▷ 해당 레지스터 주소 : **0x4001 1410**
  - ▷ 해당 레지스터의 Reset Value는 **0x0000 0000** 이므로 해당 값을 할당하여 Reset 함

- **Button1 → PC4, Button2 →PB10, Button3 → PC13, Button4 → PA0**
  - LED 조절을 위한 Input 값인 Button 4개 : Port A, B, C를 이용
    - ▷ 각 레지스터 메모리 맵핑 주소 : **0x4001 0800**, **0x4001 0C00**, **0x4001 1000**
  - Port A에 연결된 Button4와 Port C에 연결된 Button1
    - ▷ 해당 포트 값이 0~7 사이에 위치
    - ▷ CRL에 대응되어 정의
    - ▷ 두 개의 Button에는 **0x00**의 Address offset 을 더함
    - ▷ **레지스터 주소 값은 0x4001 0800, 0x4001 1000**
    - ▷ **Reset Value인 0x4444 4444를 할당**
  - Port B에 연결된 Button2와 Port C에 연결된 Button3
    - ▷ 포트 값이 8~15 사이에 위치
    - ▷ CRH에 대응되어 정의
    - ▷ Address offset은 모두 **0x04** 임
    - ▷ **레지스터 주소는 각각 0x4001 0C04, 0x4001 1004**
    - ▷ **Reset Value는 CRL과 마찬가지로 0x4444 4444 이므로 해당 값을 할당**
  - 네 개의 Button은 Input 모드로 동작
    - ▷ Port configuration register에 CNF는 01을 부여하여 Floating input을 할당
    - ▷ MODE는 00을 주어 Input mode로 동작
    - ▷ **Button1(PC4) : 0x4 0000, Button2(PB10) : 0x400, Button3(PC13) : 0x40 0000, Button4(PA0) : 0x04**

2. 정의한 내용을 바탕으로 버튼을 이용한 LED 제어를 위한 code 구현

i) 초기화

- RCC 초기화 및 LED, Button 사용을 위한 base 레지스터 초기화
- RCC초기화(코드 n1): RCC 레지스터를 설정해 클럭을 활성화하는 code

```
RCC_APB2_ENR = 0x3C; // RCC clock enable
```

코드 1. RCC 초기화 code

- LED, Button 사용을 위한 base 레지스터 초기화(코드 n2)
  - ▷ CRL은 포트의 0~7번 핀, CRH는 8~15번핀을 설정하므로 이에 맞는 포트만 reset(line 1~5)
  - ▷ Button 설정
    - ◆ Key1 – PC4, Key2 – PB10, Key3 – PC13, Key4 – PA0 로 포트의 핀 설정
    - ◆ <코드 2>의 line 9~12와 같이 필요한 포트만 reset 시킴
  - ▷ LED 설정
    - ◆ **GPIOD_CRL = 0x10011100**; : GPIOD(PD) 2, 3, 4, 7을 output으로 설정하는 code
    - ◆ PD2, 3, 4, 7 => 모두 0~7에 해당하기에 GPIOD_CRL에서 사전 정의됨
  - ▷ 모든 LED가 reset으로 시작하므로 GPIOD_BSRR = 0x00000000; 추가
  - ▷ LED가 꺼진 상태를 0으로 정의한다(off = 0, on = 1)

```
GPIOD_CRL = 0x44444444;
GPIOC_CRH = 0x44444444;
GPIOC_CRL = 0x44444444;
GPIOB_CRH = 0x44444444;
GPIOA_CRL = 0x44444444;

GPIOD_CRL = 0x10011100; // LED(PD2, PD3, PD4, PD7)

GPIOD_BSRR = 0x00000000; // all led reset

int led1 = 0, led2 = 0, led3 = 0, led4 = 0; // led state
```

코드 2. LED, Button 사용을 위한 base 레지스터 초기화 code

ii) while(1)문 내에 LED가 작동할 조건 구현

- if (~GPIOC_IDR & 0x08) 조건문
  - ▷ Button1 눌렀을 때 LED가 on상태이면 off로 바꾸고, LED가 off상태이면 on으로 전환
    - ◆ set하면 해당되는 핀이 활성화되고 비트를 reset하면 해당되는 핀이 비활성화
    - ◆ 해당 과정을 통해 LED가 on/off 되는 조건을 설정
  - ▷ Button1 눌렀을 때 LED가 on상태이면 off로 바꾸고, LED가 off상태이면 on으로 전환
    - ◆ set하면 해당 핀이 활성화되고 비트를 reset하면 해당 핀이 비활성화

```
while(1) {
  // press button1
  if (~GPIOC_IDR & 0x08){
    // led on
    if (led1 == 1){
      GPIOD_BSRR |= 0x240000;
      led1 = 0;
    }
    // led off
    else {
      GPIOD_BSRR |=  0x04;
      led1 = 1;
    }
    delay(1000000);
  }
```

코드 3. LED1, Button1 작동 구현 code

- Button 2, 3, 4 (<코드 4>)

  ▷ <코드 3>과 동일하게 구현, IDR값과 포트의 BSRR 값 변경

  ▷ BSRR 값 : Reference manual 9.2.5, 9.2.6을 참고하여 bits를 설정

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 1. BSRR 값을 설정하기 위한 Reference Manual

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

Bits 31:16 Reserved

Bits 15:0 **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

그림 2. 각 포트를 설정하기 위한 Reference Manual

```
// press button2
if (~GPIOB_IDR & 0x400) {
  // led on
  if (led2 == 1){
    GPIOD_BSRR |= 0x80000;
    led2 = 0;
  }
  // led off
  else {
     GPIOD_BSRR |= 0x08;
     led2 = 1;
  }
  delay(1000000);
}
```

```
// press button3
if (~GPIOC_IDR & 0x2000) {
  // led on
  if (led3 == 1){
    GPIOD_BSRR |= 0x100000;
    led3 = 0;
  }
  // led off
  else {
     GPIOD_BSRR |=  0x10;
     led3 = 1;
  }
  delay(1000000);
}
```

```
// press button4
if (~GPIOA_IDR & 0x01){
  // led on
  if (led4 == 1){
    GPIOD_BSRR |= 0x800000;
    led4 = 0;
  }
  // led off
  else {
     GPIOD_BSRR |=  0x80;
     led4 = 1;
  }
  delay(1000000);
}
```

코드 4. LED2, Button2(좌) / LED3, Button3(중) / LED4, Button4(우) 작동 구현 code

+) button 사용 시, 구분을 위한 delay 함수 구현(<코드 5>)

- if 모든 버튼 사용시, 버튼을 눌렀을 때 즉시 반응하지 않고 일정시간 동안 버튼의 입력을 무시하고 작동을 안정화 하여 노이즈나 튀는 현상을 방지하기 위한 함수 구현
- delay함수를 각 버튼입력 했을 경우, if문의 마지막에서 delay 실행
- delay의 지속시간을 나타내는 매개변수 nCount를 받아 반복문을 사용

▷ nCount가 0이 될 때까지 반복을 실행하는 함수로 delay를 일정시간 생성

```
void delay(uint32_t);

void delay(__IO uint32_t nCount){
  for(; nCount != 0; nCount--){}
}
```

코드 5. delay 함수 구현 code

3. 정상적인 동작 유무 확인(동작 사진 첨부)

- 코드실행 초기상태
-

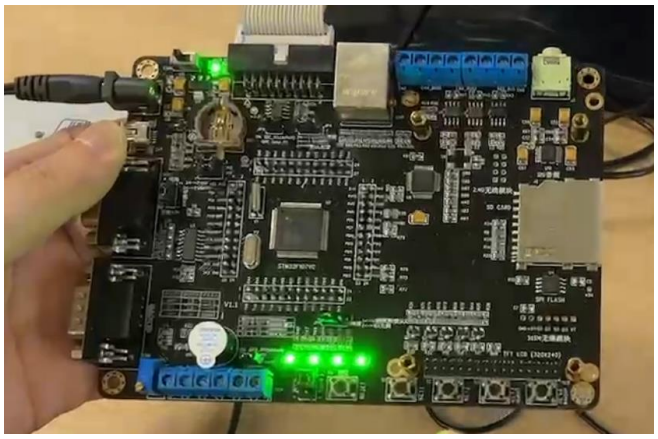

그림 3. code 실행 시 초기 상태

- LED off : Button1, 2, 3, 4를 눌러 모든 LED off

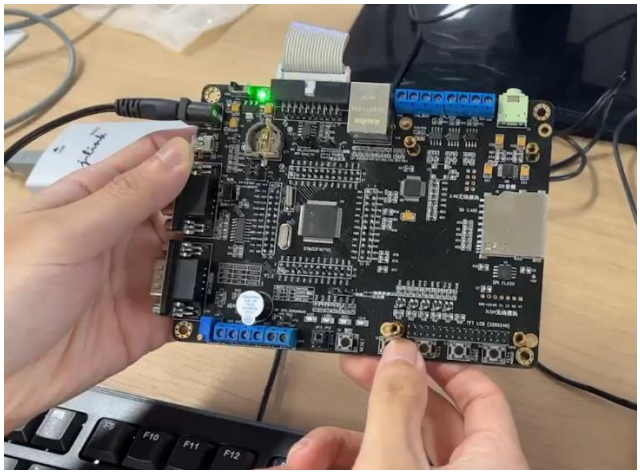

그림 4. 모든 LED Off 상태

- Button1을 눌러 LED1 on

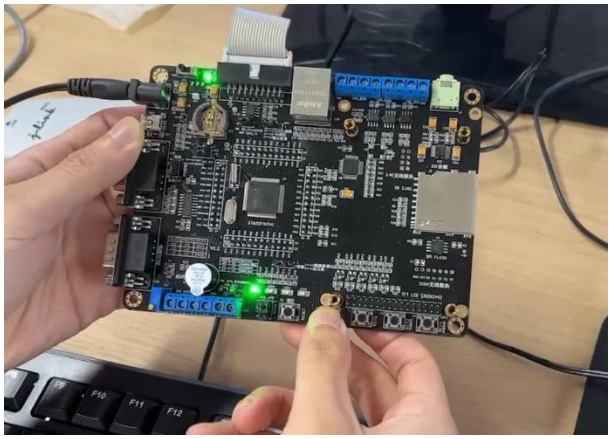

그림 5. Button1을 조작하여 LED1 On 상태

4. 오실로스코프를 이용한 디버깅

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

- 오실로스코프의 D1에 버튼1, D4에 LED1을 연결 후 실험 진행
  - ▷ Trigger 기능 활성화 후 single 버튼을 통해 버튼을 눌러 그래프의 변화 확인
  - ▷ Measure 버튼을 누른 후 노브를 돌려 시간 측정
- 버튼을 누른 후 LED의 반응속도 측정을 해 보았으나 오실로스코프 상으로도 버튼을 누른 즉시 LED가 반응하였음을 확인
- 사람의 손으로 버튼을 누르는 것이다 보니 버튼의 push/pull 사이 간격이 일정치 않음
- LED는 toggle 이므로 버튼 입력이 있기 전까지 현재 상태를 유지함
- 버튼을 계속 누르고 있을 경우 delay코드 때문에 간격을 두고 LED가 on/off 반복 동작함

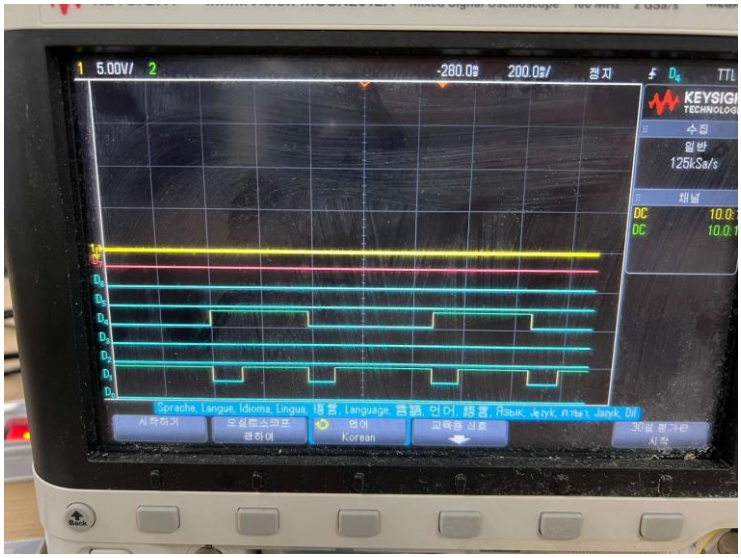

그림 6. Trigger 기능 활성화 후 Single 버튼을 통해 측정

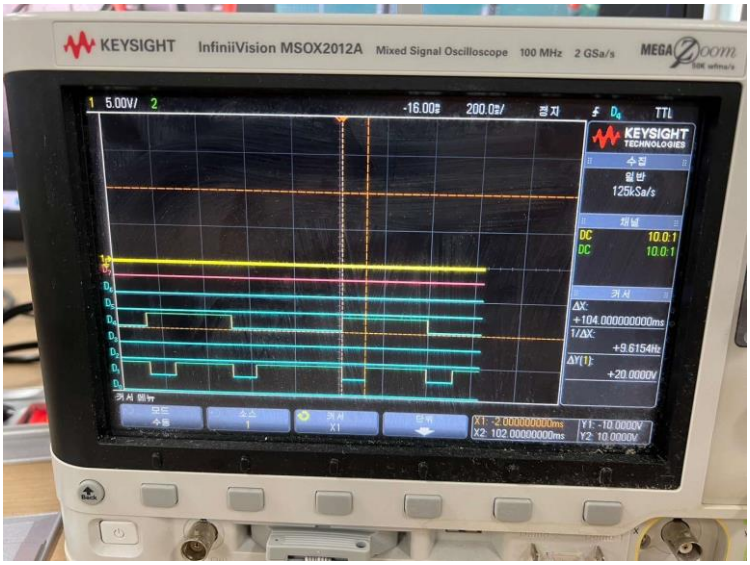

그림 7. 사람이 버튼을 빠르게 누르고 떼었을 경우의 push/pull 간격(약 104ms)

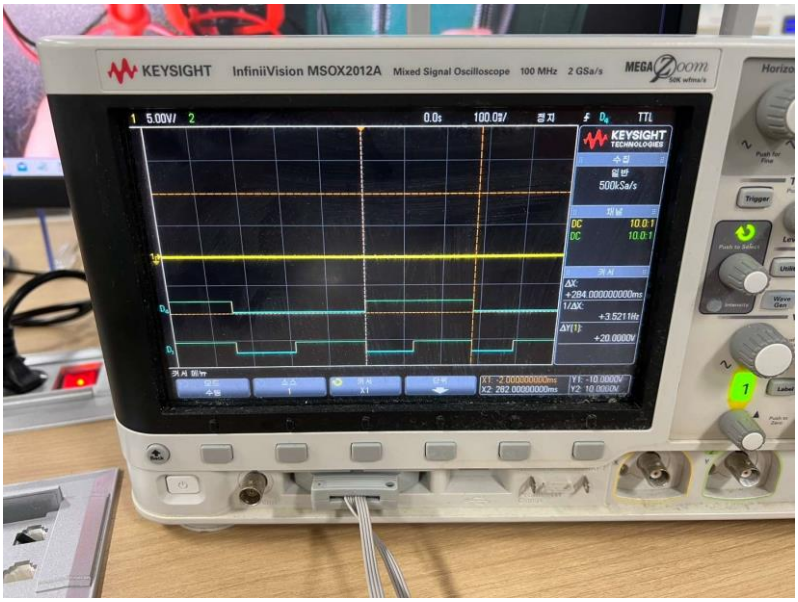

그림 8. LED의 반응을 통해 input delay 결과(< 284ms)

● 결론

- 3주차 실험을 통하여, 기본적인 GPIO 조작 방식을 익힐 수 있었고 이를 통해 기본적인 임베디드 보드를 제어하는 방법을 확인할 수 있었음
  ▷ 레지스터와 메모리 매핑 과정을 통하여 임베디드 시스템을 통한 프로그램을 구성할 수 있는 기술을 학습할 수 있었음
- 오실로스코프를 사용하여 정상 작동 여부 확인과 프로그램 구성 후 오류를 발견하는 안목과 능력을 익힐 수 있었음
- 다만, 처음 코드를 작성해보았던 만큼 어떠한 방식으로 정의하는지에 대한 예시가 없어, 초반 정의 및 구현에 있어 진행이 원활하게 되지 않아 약간의 무의미한 시간이 소요되었던 것이 있었던 것 같아 아쉬웠음