임베디드시스템설계및실습

4주차 실험 보고서

팀 4조

담당교수 백윤주

담당조교 나상진

실험날짜 2024 09 26

1. 제목 : 스캐터 파일, 플래시 프로그래밍, 릴레이 모듈, 폴링 방식의 이해
2. 실험목표
   ① 스캐터 파일의 이해 및 플래시 프로그래밍
   ② 릴레이 모듈의 이해 및 임베디드 펌웨어를 통한 동작
   ③ 폴링 방식의 이해
3. 실험 내용
   ① Datasheet 및 Reference Manual을 참고하여 해당 레지스터 및 주소에 대한 설정 이해
   ② 스캐터 파일을 통해 플래시 메모리에 프로그램 다운로드
   ③ 플래시 메모리에 올려진 프로그램 정상적인 동작 확인
4. 원리 및 이론
   ① Scatter File 이란
      1) 분산 적재 : 실행시킬 바이너리 이미지가 메모리에 로드 될 때 바이너리 이미지의 어떤 영역이 어느 주소에 어느 크기만큼 배치되어야 할 지 작성한 파일

```
/*###ICF### Section handled by ICF editor, don't touch! ****/
/*-Editor annotation file-*/
/* IcfEditorFile="$TOOLKIT_DIR$¥config¥ide¥IcfEditor¥cortex_v1_0.xml" */
/*-Specials-*/
define symbol __ICFEDIT_intvec_start__ = 0x08000000;
/*-Memory Regions-*/
define symbol __ICFEDIT_region_ROM_start__ = // TODO
define symbol __ICFEDIT_region_ROM_end__   = // TODO
define symbol __ICFEDIT_region_RAM_start__ = // TODO
define symbol __ICFEDIT_region_RAM_end__   = // TODO
/*-Sizes-*/
define symbol __ICFEDIT_size_cstack__ = 0x1000;
define symbol __ICFEDIT_size_heap__   = 0x1000;
/**** End of ICF editor section. ###ICF###*/

define memory mem with size = 4G;
define region ROM_region   = mem:[from __ICFEDIT_region_ROM_start__   to __ICFEDIT_region_ROM_end__];
define region RAM_region   = mem:[from __ICFEDIT_region_RAM_start__   to __ICFEDIT_region_RAM_end__];

define block CSTACK    with alignment = 8, size = __ICFEDIT_size_cstack__   { };
define block HEAP      with alignment = 8, size = __ICFEDIT_size_heap__     { };

initialize by copy { readwrite };
do not initialize  { section .noinit };

place at address mem:__ICFEDIT_intvec_start__ { readonly section .intvec };

place in ROM_region   { readonly };
```

      2) 스캐터 파일을 통해 원하는 메모리 위치에 프로그램 다운로드
         A. ROM 크기 0x80000

```
define symbol __ICFEDIT_region_ROM_start__ = 0x08000000; // TODO
define symbol __ICFEDIT_region_ROM_end__   = 0x0807FFFF;// TODO
```

         ROM의 크기를 80000만큼 설정

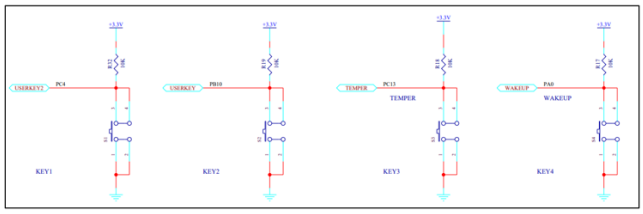
         B. RAM 크기 0x8000

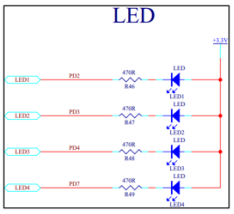
```
define symbol __ICFEDIT_region_RAM_start__ = 0x20000000;// TODO
define symbol __ICFEDIT_region_RAM_end__   = 0x20007FFF;// TODO
```

         RAM의 크기를 8000만큼 설정

② 레지스터 및 주소 설정

1) 릴레이 모듈을 사용하기 위한 INPUT GPIO (PC4, PB10, PC13, PA0)





Port C	0x4001 1000 - 0x4001 13FF
Port B	0x4001 0C00 - 0x4001 0FFF
Port A	0x4001 0800 - 0x4001 0BFF

PORT A,B,C, 의 base address

pc4, pa0 configuration low

address offset 0x00 (0x40011000 + 0x00 , 0x40010800 + 0x00)

pb10, pc13 configuration high

address offset 0x04 (0x40011000 + 0x04, 0x40010C00 + 0x04)

port input data register idr

adress offset 0x08 (위 base address + 0x08)

```
// port C configuration low
#define GPIO_C_CRL (*(volatile unsigned int*)0x40011000) // PC4
// port C configuration high
#define GPIO_C_CRH (*(volatile unsigned int*)0x40011004) // PC13
// port C input data regiser
#define GPIO_C_IDR (*(volatile unsigned int*)0x40011008)

// port B configuration high
#define GPIO_B_CRH (*(volatile unsigned int*)0x40010C04) // PB10
// port B input data regiser
#define GPIO_B_IDR (*(volatile unsigned int*)0x40010C08)

// port A configuration high
#define GPIO_A_CRL (*(volatile unsigned int*)0x40010800) // PA0
// port A input data regiser
#define GPIO_A_IDR (*(volatile unsigned int*)0x40010808)
```

2) 릴레이 모듈을 사용하기 위한 OUTPUT GPIO (PD2, PD3로 설정)

PORT D의 base address 0x40011400 (주석참조)

Bsrr의 address offset 0x10

Bss의 address offset 0x14

Port D	0x4001 1400 - 0x4001 17FF

```
// port D set/reset register
#define GPIO_D_BSRR (*(volatile unsigned int*)0x40011410) // 0x40011400 + 0x10
// port D reset register
#define GPIO_D_BRR (*(volatile unsigned int*)0x40011414) // 0x40011400 + 0x14
```

PORT D의 2,3 의 configuration은 low

CRL의 address offset 0x00

```
#define GPIO_D_CRL (*(volatile unsigned int*)0x40011400)
```

3) RCC_ABP2ENR clock enable

1. Bit 2,3,4,5를 1로 변경. enable port a,b,c,d, 를 clock enable 하기 위함

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

```
// 1. RCC enable
// RCC_ABP2ENR |= (1<<2)|(1<<3)|(1<<4)|(1<<5);
RCC_ABP2ENR |= (0x3C); // IO port d, C, B, A clock enable(bit5, bit4, bit3, bit2)
```

4) GPIO a,b,c 의 configuration 을 pull up, pull down 으로 변경

1. GPIO A0, B10, C4, C13

q

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

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= 8 .. 15)
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= 8 .. 15)
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.

Configuration high 10,13 을 pull up/pull down 10 input mode 00

Configuration low 4,0 을 위와 동일하게 변경 (버튼의 풀업풀다운)

```
GPIO_C_CRL &= 0xFFF0FFFF;
GPIO_C_CRL |= 0x00080000;     //pc4

GPIO_C_CRH &= 0xFF0FFFFF;
GPIO_C_CRH |= 0x00800000;   //pc13

GPIO_B_CRH &= 0xFFFFF0FF;
GPIO_B_CRH |= 0x00000800; //pb10

GPIO_A_CRL &= 0xFFFFFFF0;
GPIO_A_CRL |= 0x00000008;  //pa0
```

2. GPIO D2,D3
   A. 위 reference 에 의하여 d2,d3 을 output 모드로 할 것임으로 해당되는 비트를 11로 pull/push를 00으로 하여 0011=> 3으로 or 연산

```
// 2. GPIOx
GPIO_D_CRL  &= 0xFFFF00FF;     // reset
GPIO_D_CRL  |= 0x00003300;     // output mode
```

3. GPIO D BSRR
   A. 릴레이 1,2를 off상태로 초기화 하기 위하여 reference를 참조

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

BR2 bit 를 1로 하여 pd2를 reset(0x40000)

BR3 bit 를 1로 하여 pd3를 reset(0x80000)

```
// 3. set/reset register
GPIO_D_BSRR |= (0x40000 | 0x80000); //  릴레이 1 ,2  off상태로 초기화  pd2 pd3 reset bit 1
```

4. 모터의 회전과 정지
   A. 정방향 회전을 위하여 회로를 정방향으로 구성한 output 포트에 set bit 를 1 로 활성화 (pd2)
   B. 역방향 회전을 위하여 회로를 역방향으로 구성한 output 포트에 set bit 를 1로 활성화 (pd3)
   C. 모터의 정지를 위하여 회로를 정방향으로 구성한 output 포트의 reset bit 를 1로 비활성화 (pd2)
   D. 모터의 정지를 위하여 회로를 역방향으로 구성한 output 포트

의 reset bit를 1로 비활성화 (pd3)

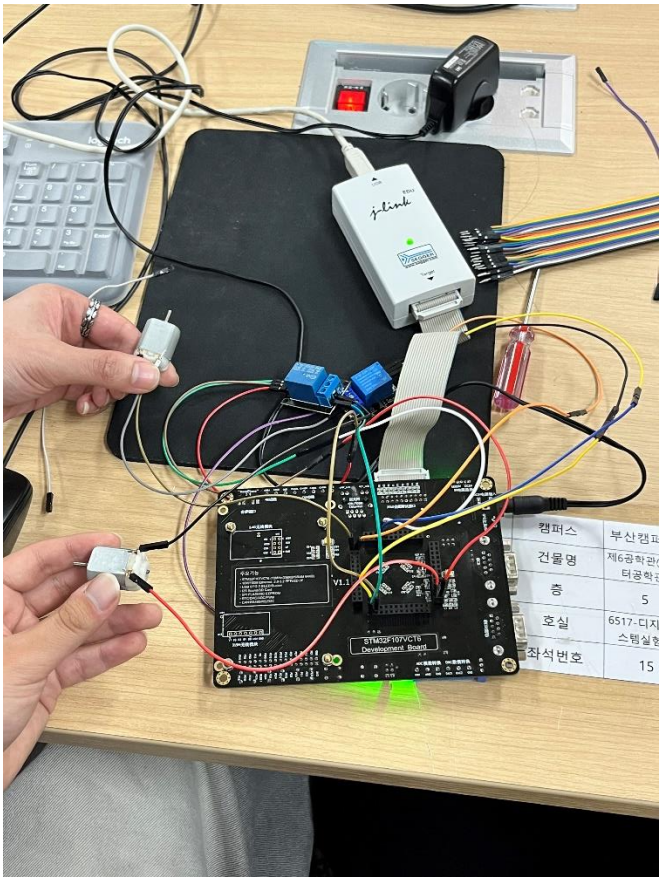
```
while(1){
  if(!(GPIO_C_IDR& (1<<4))){ // Button1 눌렸는지 확인
    GPIO_D_BSRR |= OUT1;     // 정방향 릴레이 1 on (pd2)

  }
  if(!(GPIO_B_IDR& (1<<10))){ // Button2 눌렸는지 확인
    GPIO_D_BSRR |= OUT2;     // 역방향 릴레이 2 on (pd3)

  }
  if(!(GPIO_C_IDR& (1<<13))){ // Button3 눌렸는지 확인
    GPIO_D_BSRR |= OUT1;     // 정방향 릴레이 1 on (pd2)
    delay();                 // 2초 딜레이
    GPIO_D_BRR |= OUT1;      // 정방향 릴레이 1 off (pd2)
    GPIO_D_BSRR |= OUT2;     // 역방향 릴레이 2 on (pd3)
    delay();                 // 2초 딜레이
    GPIO_D_BRR |= OUT2;      // 역방향 릴레이 2 off (pd3)

  }
  if(!(GPIO_A_IDR& (1<<0))){  // Button4 눌렸는지 확인
    GPIO_D_BRR |= OUT1;      // 정방향 릴레이 1 off (pd2)
    GPIO_D_BRR |= OUT2;      // 역방향 릴레이 2 off (pd3)


  }
}
```

```
#define OUT1 (1<<2)
#define OUT2 (1<<3)
```

5. 릴레이 모듈 회로 연결
   A. 릴레이 모듈을 사용하기 위해 릴레이모듈의 각 부분을 보드에 연결 하였다.

   1) 릴레이 모듈의 입력전압은 회로의 3.3v에 연결한다
   2) GND 는 회로의 GND와 연결한다
   3) 제어신호 INPUT은 회로에서 출력포트에 해당하는 PD2, PD3 와 각각 연결한다
   4) 버튼이 눌렸을 때 동작하도록 할 것이므로 NO 단자에 3.3V를 연결한다
   5) 공통단자는 모터의 한쪽과 연결한다
   6) 모터의 나머지 한쪽은 보드의 그라운드에 연결한다.
   7) 모터의 역회전을 위해 모터의 서로 반대방향에 각각 공통단자와 그라운드를 연결한다.

   B. 회로 사진 첨부



6. 폴링방식의 원리와 문제점
   A. 폴링방식은 하드웨어의 변화를 지속적으로 읽어들여 변화를 알아채는 방식이다.
   B. 폴링방식은 신호를 판단하기 위해 지속적으로 확인해야 한다.
   C. 다른 일을 하는 중에 신호를 읽을 수 없는 단점이 있다.