

제 10 장 스위치 조작 및 키패드 입력

이 장에서는 마이크로컨트롤러의 입력 장치로 많이 사용되는 푸쉬 버튼 스위치와 키패드 입력을 받는 방법에 대해 알아본다.

10.1 푸쉬 버튼 조작

6장에서 사용하였던 푸쉬 버튼은 손으로 누르는 동안 동작을 하고 손을 놓으면 내장 스프링에 의해 동작이 복귀하는 스위치이다. 푸쉬 버튼은 손을 놓으면 스위치가 열리는 a-접점, 손을 놓으면 스위치가 닫히는 b-접점 두 가지가 있다. 교재에서는 a-접점을 사용한다.

여기서는 7.5절에서 사용하였던 스위치 입력회로에 하나를 더해 그림 10.1과 같이 세 개의 a-접점 푸쉬 버튼을 스위치로 사용하는 회로를 구성한다.

10.1.1 버튼 조작 프로그램 (I)

다음과 같은 프로그램을 작성하여 보자.

- (1) sw0를 누르면 7-세그먼트에 표시된 수를 1 증가한다.
- (2) sw1을 누르면 7-세그먼트에 표시된 수를 2 증가한다.
- (3) sw2를 누르면 7-세그먼트에 표시된 수를 3 증가한다.
- (4) 여러 개의 버튼이 눌러지면 무시한다.

이를 6장에서 사용한 스위치 입력방법과 9.5절에서 작성한 7-세그먼트 구동프로그램을 사용하여 프로그램 10.1과 같이 작성하여 보자. 프로그램의 흐름은 다음과 같다.

- ☞ 디바운싱을 하면서 버튼을 누른 후 떨 때까지 기다린다.
- ☞ 버튼에 따라 데이터를 증가시키고 7-세그먼트에 10진수로 표시한다.

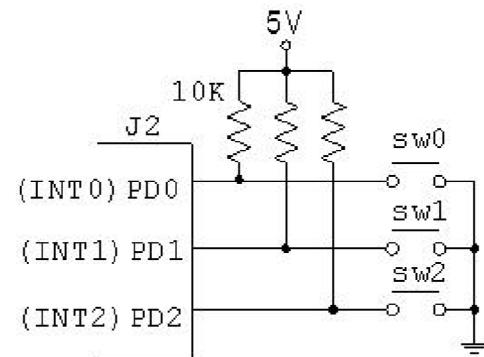


그림 10.1 스위치 회로

```

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include "iseg7.h"
void msec_delay(int n);           // 시간지연함수
int main()
{
    unsigned char sw;             // 입력된 스위치 값
    unsigned short data = 0;
    ISeg7Init();                 // 7-세그먼트 초기화
    sei();                        // 인터럽트허용
    ISeg7DispNum(data, 10);
    while(1)
    {
        // 스위치 입력을 기다림
        while((sw=~PIND & 0x07) == 0);
        msec_delay(30);           // 디바운싱
        while((~PIND & 0x07));   // 버튼 뗄 때를 기다림
        msec_delay(30);           // 디바운싱
        if(sw == 0x01)            // sw0
            data += 1;            // 1을 더한다.
        else if(sw == 0x02) // sw1
            data += 2;            // 2를 더한다.
        else if(sw == 0x04) // sw2
            data += 3;            // 3을 더한다.
        // 동시에 누른 것은 무시
        ISeg7DispNum(data, 10);
    }
}
void msec_delay(int n) // 6.5절에서 작성한 시간지연함수
{ . . . }

```

프로그램 10.1

프로젝트에 iseg7.c를 추가하고 프로그램 10.1을 수행하면 원하는 데로 동작하지만 7-세그먼트의 값이 버튼을 누를 때 증가하지 않고 버튼을 뗄 때 값이 증가한다. 이를 버튼을 누를 때 7-세그먼트의 값이 증가하도록 하여보자.

- ☞ 버튼이 계속 눌러져 있는 상태를 판단하여 처음 눌러졌을 때는 입력을 받아들이고 계속 눌러져 있는 상태일 때는 입력이 없는 것으로 판단한다. 계속 눌려 있는 상태를 파악하기 위해 이전 버튼 상태를 저장하는 변수를 도입하여 작성한다.

프로그램 10.2는 변경된 main() 함수만 표시한 것으로 이를 수행하면 원하는 데로 동작한다.

- ☞ 처음 while() 루프내의 다음 문장은 디바운싱하면서 버튼 입력을 받아들이는 부분이다.

```
sw=~PIND & 0x07; // 버튼 입력을 받는다.
while(1)
{
    msec_delay(30); // 디바운싱 시간지연
    sw1 = ~PIND & 0x07; // 한 번 더 읽는다.
    if(sw == sw1) // 두 입력이 같으면 신호가 안정화
        break; // 된 것으로 판단.
    sw = sw1;
}
```

디바운싱을 위해서 30msec 간격으로 버튼 입력을 받아들인다. 연속된 두 번의 입력이 같을 경우 신호의 채터링이 사라지고 신호가 안정화된 것으로 판단하여 버튼 입력 값으로 받아들인다.

- ☞ 변수 psw는 이전에 받아들인 버튼 입력 값이고 변수 sw는 현재 받아들인 입력 값이다. 두 변수가 같으면 버튼이 계속 눌려 있는 상태이고 두 변수가 서로 다르면 버튼의 상태가 바뀌었음을 나타낸다.
- ☞ 버튼의 상태가 바뀌면, 즉 버튼이 새로이 눌러졌을 때만 7-세그먼트의 값을 변화시킨다.

```

int main()
{
    unsigned char sw1, sw, psw=0; //입력값과 이전 값
    unsigned short data = 0;

    ISeg7Init();           // 7-세그먼트 초기화
    sei();
    ISeg7DispNum(data, 10); // 초기값 표시

    while(1)
    { //=====
        // 디바운싱을 하면서 버튼 입력을 받음
        //=====
        sw=~PIND & 0x07; // 버튼 입력을 받는다.
        while(1)
        {
            msec_delay(30); // 디바운싱 시간지연
            sw1 = ~PIND & 0x07; // 한 번 더 읽는다.
            if(sw == sw1) // 두 입력이 같으면 신호가 안정화
                break; // 된 것으로 판단.
            sw = sw1;
        }
        //=====
        if(psw != sw) // 이전 값과 비교한다.
        { // 누르고 있는 상태가 아님
            if(sw == 0x01) // sw0
                data += 1; // 1을 더한다.
            else if(sw == 0x02) // sw1
                data += 2; // 2를 더한다.
            else if(sw == 0x04) // sw2
                data += 3; // 3을 더한다.
            // 입력이 없거나 동시에 누른 입력은 무시
        }
        psw = sw; // 현재 값을 저장
        ISeg7DispNum(data, 10);
    }
}

```

그림 10.1의 회로에 대한 버튼 입력은 계속적으로 사용할 것이므로 7-세그먼트 구동 프로그램처럼 독립된 소스파일 button.c와 헤더파일 button.h에 함수로서 작성하여 계속적으로 사용할 수 있게 한다. 소스 파일 button.c에서 BtnInput() 함수는 푸쉬 버튼 입력을 받아들이는 함수로서 [프로그램 prac6-1-1.c]에서 사용된다.

- ☞ 헤더파일 작성 시 `#ifndef - #endif` 쌍을 사용하여 헤더파일의 중복 사용을 가능하도록 하였다.(뒤 설명 참조)
- ☞ 이전 값을 저장하는 변수 psw는 BtnInput() 함수가 종료되더라도 값을 유지하여야 하므로 정적변수로 선언한다.
- ☞ main() 함수에서 if-else문은 switch문으로 변경하였다. 많은 종류에 대해 판단을 내려야 할 때는 switch문이 효과적이다.

실습: [프로그램 prac10-1I.c]는 button.c함수의 BtnInput() 함수를 사용해 프로그램 10.2를 다시 작성한 것이다. 실행을 위해 button.c, button.h, iseg7.c, iseg7.h를 프로젝트에 소스파일을 포함시키고 프로그램을 수행한다. 푸쉬 버튼을 누를 때마다 7-세그먼트의 값이 증가하는 것을 볼 수 있다.

과제 : 그림 10.1은 스위치 입력을 받기 위하여 외부풀업을 사용하고 있다. 외부풀업 저항을 제거한 후 내부풀업을 사용할 수 있도록 button.c를 변경하고 [프로그램 prac10-1I.c]를 수행하라.

```
switch(제어표현식)
{
    case label1:
        문장 1;
        break;
    case label2:
        문장2;
        break
    . .
    default:
        default 문장
}
```

☞ **switch** 문

- ▶ 제어표현식의 결과가 case 라벨들의 값 label1, label2, ... 과 비교
- ▶ 일치하는 것이 발견되면 그 case 라벨을 뒤따르는 문장들을 break문을 만날 때까지 실행
- ▶ break를 만나면 switch문을 벋어남.
- ▶ default : case 라벨과 일치하지 않는 경우는 default를 뒤따르는 문장을 실행한다.

☞ 헤더파일에서 선행처리기 **#ifndef**의 사용

- ▶ C-언어에서 선행처리기는 #로 시작한다. 선행처리기는 컴파일러가 프로그램을 컴파일하기 전에 처리하는 문장을 말한다.

- ▶ **#ifndef** 선행처리기의 문법은 다음과 같다.

```
#ifndef 매크로명
```

```
...
```

```
#endif
```

- ▶ **#ifndef**는 **#endif**와 짝을 이루며 “매크로명”이 매크로로 정의되어 있으면 **#ifndef - #endif** 짝 내의 문장을 포함하여 컴파일하고 정의되어있지 않으면 포함하지 않고 컴파일을 수행한다.

- ▶ 헤더파일은 응용프로그램에서 충복 사용되는 경우가 매우 많다. 다음 헤더파일을 고려해보자.

헤더파일 **button.h**

```
#ifndef __BUTTON_H__
#define __BUTTON_H__
#define BTN_SW0 0x01
...
#endif
```

헤더파일 **outfile.h**

```
#include "button.h"
...

```

헤더파일 **outfile.h** 속에서 **button.h** 헤더파일을 첨부하고 있다. 프로그램을 다음과 같이 작성하면 문장 ① ②에 걸쳐 **button.h**가 두 번 첨부된다.

```
#include "outfile.h"      ----- ①
#include "button.h"       ----- ②
```

```
...
```

- ①에서 **button.h**가 첨부될 때 매크로 **__BUTTON_H__**가 정의되지 않았으므로 매크로 **__BUTTON_H__**와 **BTN_SW0**를 정의한다.
- ②에서 **button.h**가 첨부될 때 ①에서 매크로 **__BUTTON_H__**가 이미 정의된 상태이므로 **#ifndef - #endif** 짝내의 문장을 포함시키지 않는다. 따라서 매크로 **BTN_SW0**이 충복해서 정의되지 않는다.

- * **button.h**에서 **#ifndef - #endif**를 사용하지 않으면 ②에서 매크로 **BTN_SW0**이 충복 정의가 되어 에러가 발생하게 된다.

[프로그램 prac10-11.c]

```

//=====
// 실습 10.1.1 : 푸쉬 버튼 입력
//=====

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include "iseg7.h"          // 7-세그먼트 구동
#include "button.h"         // 푸쉬 버튼 입력

void msec_delay(int n); // 시간지연함수

int main()
{
    unsigned char btn; // 버튼 입력 값
    unsigned short data = 0;

    ISeg7Init();          // 7-세그먼트 초기화
    BtnInit();            // 스위치 입력 초기화
    sei();                // 인터럽트 허용

    ISeg7DispNum(data, 10);
    while(1)
    {
        btn = BtnInput(); // 버튼 입력을 받는다.
        switch(btn)
        {
            case BTN_SW0:           // sw0
                data +=1;
                break;
            case BTN_SW1:           // sw1
                data += 2;
                break;
            case BTN_SW2:           // sw2
                data += 3;
                break;
            default:                // 입력이 없거나,

```

```

        break;           // 여러 스위치가 눌려진 경우 제외
    }
    ISeg7DispNum(data, 10);
}
void msec_delay(int n)           // 시간지연함수
{
    for(; n >0; n--)           // 1msec 시간지연을 n회 반복
        _delay_ms(1);          // 1msec 시간지연
}

```

[헤더파일 button.h]

```

//=====
// 버튼 입력 헤더파일 button.h
//=====

#ifndef __BUTTON_H__
#define __BUTTON_H__

// 입력 버튼 정의
//
#define BTN_SW0      0x01      // sw0을 누름
#define BTN_SW1      0x02      // sw1을 누름
#define BTN_SW2      0x04      // sw2를 누름
#define BTN_SW01     (BTN_SW0 | BTN_SW1)
#define BTN_SW02     (BTN_SW0 | BTN_SW2)
#define BTN_SW12     (BTN_SW1 | BTN_SW2)
#define BTN_SW012    (BTN_SW0 | BTN_SW1 | BTN_SW2)
#define NO_BTN       0x00      // 입력이 없음
#define BTN_MASK     (BTN_SW0 | BTN_SW1 | BTN_SW2)

// 함수 정의
void BtnInit(void);
unsigned char BtnInput(void);
unsigned char BtnInput_Press(unsigned char *pressed)

#endif

```

[소스파일 button.c]

```

#include <avr/io.h>
#include <util/delay.h>
#include "button.h"

#define BTN_IMPORT PIND          // 버튼 입력 포트
#define BTN_DDR    DDRD          // 방향 포트

static unsigned char     psw=0;    // 이전 버튼 값

void BtnInit(void)
{
    BTN_DDR &= ~BTN_MASK; // 버튼 입력 핀을 입력으로 설정
}

//=====
// 버튼 입력을 받는다. (계속 누르고 있을 때는 입력이 없는 것으로
// 판단한다.)
// 리턴 값 : 0      - 입력이 없음(계속 누르고 있는 상태포함)
//           이외    - 버튼 입력(헤더파일 참조)
//=====

unsigned char BtnInput(void)
{
    unsigned char sw, sw1;

    sw = ~BTN_IMPORT & BTN_MASK; // 버튼 입력을 받는다.

    while(1) // 연속 두 번 읽은 값이 같을 때까지 루프를 돈다.
    {
        // 디바운싱 시간지연
        _delay_ms(10); _delay_ms(10); _delay_ms(10);
        sw1 = ~BTN_IMPORT & BTN_MASK; // 한 번 더 읽는다.

        if(sw == sw1)             // 두 번 읽은 값이 같으면 신호가
            break;                // 안정화된 것으로 판단한다.
        sw = sw1;
    }
}

```

```

    if(sw == psw) return 0; // 이전 값과 같으므로 눌러진 상태
    psw = sw;                // 이전 값을 저장한다.
    return sw;                // 스위치 입력 값을 리턴
}

//-----
// 버튼을 계속 누르고 있는지 판단하는 시간
// PRESS_THRESHOLD * 디바운싱지연 이상 누르고 있으면 버튼을 계속
// 누른 것으로 판단한다.
//-----

#define PRESS_THRESHOLD      16

//=====
// 버튼 입력을 받는다. (계속 누르고 있는 입력도 받아 들일수 있음)
//   - 짧은 시간 누르면 입력이 없는 것으로 판단
//   - 기준시간 이상 누르면 입력을 계속 하는 것으로 판단
// 출력 :
//   pressed - 0 : 계속 누르는 것이 아님
//                 1 : 계속 누르고 있음
// 리턴 값 : 0      - 입력이 없음
//           이외    - 버튼 입력(헤더파일 참조)
//=====

// 
unsigned char BtnInput_Press(unsigned char* pressed)
{
    static unsigned char    press_cnt=0;
    unsigned char sw, sw1;

    sw = ~BTN_IMPORT & BTN_MASK; // 버튼 입력을 받는다.
    while(1) // 연속 두 번 읽은 값이 같을 때까지 루프를 돈다.
    {
        // 디바운싱 시간지연
        _delay_ms(10); _delay_ms(10); _delay_ms(10);

        sw1 = ~BTN_IMPORT & BTN_MASK; // 한번 더 읽는다.

```

```

        if(sw == sw1) // 두 번 읽은 값이 같으면 신호가
                      // 안정화된 것으로 판단한다.
        sw = sw1;
    }

*pressed = 0;

if(sw == psw)
{
    press_cnt++; // 누른 시간 증가
    if(press_cnt > PRESS_THRESHOLD)
    {
        if(sw != 0)
        {
            *pressed = 1; // 계속 누르고 있는 상태임
            _delay_ms(10); _delay_ms(10);
            _delay_ms(10); _delay_ms(10);
            _delay_ms(10); _delay_ms(10);
            _delay_ms(10);
        }
    }
    else
    { // 짧은 시간 누르고 있으므로 입력이 없는 것으로 판단
        return 0;
    }
}
else
{
    // 새 버튼이 눌러졌거나, 버튼을 뗐음.
    press_cnt = 0; // 누른 시간을 리셋한다.
}

psw = sw; // 이전 값을 저장한다.
return sw; // 버튼 입력 값을 리턴
}

```

10.1.2 버튼조작 프로그램 (II)

10.1.1절에서 사용한 BtnInput() 함수는 버튼이 계속 눌러진 상태에서는 입력을 받아들이지 않는다. 이를 다음과 같이 변경하여 보자.

- ☞ 버튼을 0.5초미만 누르고 있으면 처음 버튼을 누를 때를 제외하고 버튼 입력이 없는 것으로 간주한다.
- ☞ 0.5초 이상 누르고 있으면 계속적으로 버튼을 누르는 것으로 간주하여 100msec마다 누른 스위치에 해당하는 증분값으로 7-세그먼트의 값을 증가시킨다.

이를 위해 BtnInput() 함수의 기능에 다음을 추가하여야 한다.

- ☞ 버튼을 0.5초미만 누르고 있으면 처음 버튼을 누를 때만 스위치 값을 리턴하고 아니면 0을 반환하도록 한다.
- ☞ 버튼을 0.5초 이상 누르고 있으면 버튼 값을 반환하도록 하고 디바운싱을 포함하여 100msec 지연한 후 함수를 리턴한다.
- ☞ 반환된 버튼 값이 처음 버튼을 눌러서 입력된 값인지 0.5초 이상 계속 누르고 있어 입력된 값인지 구분할 수 있도록 인수 출력으로 알려 주도록 한다.

앞 절에서 작성한 button.c파일의 BtnInput_Press() 함수는 위의 기능을 추가한 함수이다. BtnInput_Press() 함수를 설명하면

- ☞ BtnInput_Press() 함수는 버튼이 계속 눌려있는지를 나타내는 플래그를 인수로서 출력한다. 정보를 함수의 인수로 출력할 때는 변수의 포인터를 사용하여야 한다. BtnInput_Press() 함수에서 인수인 pressed 변수는 unsigned char형 변수의 포인터이다.(뒤 설명참조)
- ☞ 정적변수 press_cnt 변수는 이전 버튼 입력값과 현재 버튼 입력값이 같으면 증가하여 얼마동안 계속 누르고 있는지 알려준다. 함수가 호출될 때 마다 30msec시간지연으로 디바운싱 하므로 누르고 있는 시간

은 press_cnt \times 30msec로 계산된다.

- ☞ press_cnt가 16미만이면 약 0.5초 미만을 누른 것이므로 버튼 입력이 없는 것으로 판단하고 0을 반환한다.
- ☞ press_cnt가 16이상이면 계속 누르고 있다는 것을 표시하는 플래그인 *pressed변수를 1로 설정하고 70msec 시간 지연 후(디바운싱을 포함하여 100msec 시간지연 구현) 현재 버튼 값을 반환한다.
- ☞ press_cnt는 이전 버튼 값과 현재 버튼 값이 다르면 새 버튼이 눌러졌거나 버튼을 뗐으므로 0으로 리셋하여 다시 카운트를 시작한다.

[프로그램 prac10-1III.c]는 BtnInput_Press() 함수를 사용하여 버튼0과 버튼1을 1초 이상 누르고 있으면 100msec마다 7-세그먼트의 값이 증가하는 프로그램이다. 버튼2는 1초 이상 누르고 있어도 증가하지 않는다.

실습: [프로그램 prac10-1III.c]를 수행하라.

과제 :

1. BtnInput_Press() 함수를 0.5초 이상 2초 미만 누르면 100msec시간 지연, 2초 이상이면 50msec시간 지연 후(디바운싱 시간지연 포함) 리턴을 하도록 프로그램을 변경하고, [프로그램 prac10-1III.c]를 수행하여 본다.
2. [프로그램 prac8-4.c]에서는 외부인터럽트를 사용하여 버튼입력을 받아 들였다. prac8-4.c를 button.c를 사용하여 LED의 밝기를 조절하는 프로그램으로 변경한 후 LED의 밝기를 조절하여 본다.(계속 누르는 입력도 받아들이도록 할 것)

[프로그램 prac10-III.c]

```

//=====
// 실습 10.1.2 : 푸쉬 버튼 입력 예제 - 계속 누르는 입력을 허용
//=====

//
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include "iseg7.h"          // 7-세그먼트 구동
#include "button.h"         // 버튼 입력

void msec_delay(int n); // 시간지연함수

int main()
{
    unsigned char btn;      // 버튼 입력 값
    unsigned char pressed; // 버튼을 계속 누르는 것인지 판단
    unsigned short data = 0;

    ISeg7Init();           // 7-세그먼트 초기화
    BtnInit();              // 스위치 입력 초기화
    sei();                  // 인터럽트 허용
    ISeg7DispNum(data, 10);

    while(1)
    {
        btn = BtnInput_Press(&pressed);

        switch(btn)
        {
            case BTN_SW0:           // 스위치 0
                data +=1;
                break;
            case BTN_SW1:           // 스위치 1
                data += 2;
                break;
            case BTN_SW2:           // 스위치 2
                data += 4;
                break;
        }
        ISeg7DispNum(data, 10);
    }
}

```

```
        if(pressed) break;
        data += 3;
        break;
    default:           // 입력이 없거나,
                      // 여러 버튼이 눌러진 경우 제외
    }
    ISeg7DispNum(data, 10);
}
}

void msec_delay(int n)          // 시간지연함수
{
    for(; n >0; n--)          // 1msec 시간지연을 n회 반복
        _delay_ms(1);          // 1msec 시간지연
}
```

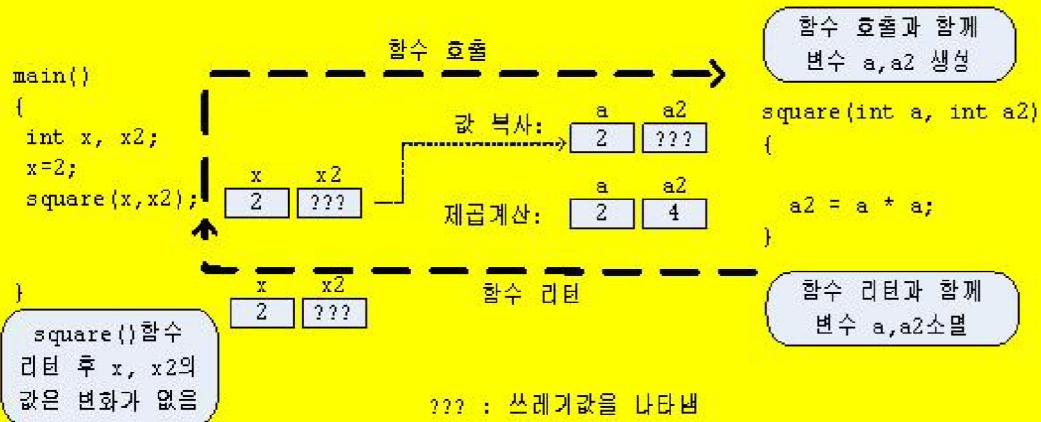
☞ 함수의 인수를 사용한 데이터 출력

- ▶ 함수는 여러 개의 인수를 가질 수 있으며 인수를 통하여 데이터를 입력받거나 함수를 호출한 곳에 데이터를 출력할 수 있다.
- ▶ 예제로서 함수의 인수사용법을 알아보자. 입력을 받아 입력의 제곱을 계산하여 인수로 출력하는 함수 `square()`를 다음과 같이 작성하여보자.

```
int main()
{
    int x, x2;
    x=2;
    square(x, x2);
    ...
}
```

```
void square(int a, int a2)
{
    a2 = a * a;
}
```

- `main()`에서는 2^2 을 계산하기 위해 `square()`함수를 호출하였고 호출 후 변수 `x2`에 $2^2=4$ 가 저장되기를 바라지만 결과는 그렇지 못하다. 다음 그림은 함수호출 과정을 보여준다.



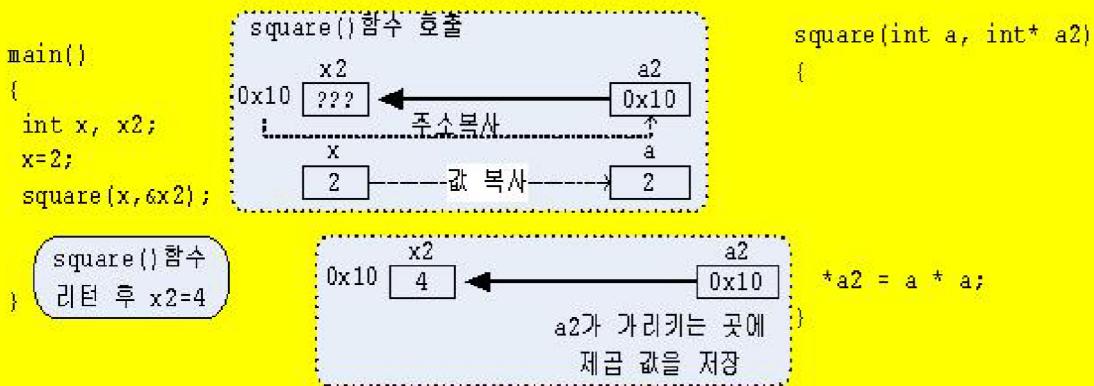
- `square()`함수를 호출할 때 변수 `a`, `a2`가 생성되고 `main()`함수의 `x`, `x2`의 값이 각각 `a`와 `a2`에 복사된다.
- `x2`와 `a2`는 다른 변수이므로 `a2`에 제곱이 계산되더라도 `x2`값은 변하지 않는다. 이 상태에서 함수가 리턴을 하면 `square()`함수 내에서 생성된 변수 `a`와 `a2`는 소멸된다. 따라서 함수 리턴 후 `main()`의 변수 `x2`는 여전히 쓰레기 값을 갖는다.

- ▶ 앞의 예에서 계산결과가 호출함수인 main()에 전달되지 못하는 것은 square()내의 변수를 변경하더라도 main()내의 변수가 영향을 받지 않기 때문이다. 함수내의 변수 값을 호출한 함수에 인수로서 전달할 때는 변수의 포인터를 사용하여 다음과 같이 작성하여야 한다.

```
int main()
{
    int x, x2;
    x=2;
    square(x, &x2);
    ...
}
```

```
| void square(int a, int *a2)
| {
|     *a2 = a * a;
| }
```

- ▶ 제곱 결과를 출력하는 인수는 포인터로 설정하였다. 다음 그림은 square()의 계산결과가 main()에 전달되는 과정을 보여준다.



- 함수를 호출할 때 square() 함수의 변수 `a2`에는 `x2`의 주소가 저장된다. `a2`는 포인터이므로 `a2`는 변수 `x2`를 가리킨다.
- 제곱 계산 후 결과는 `a2`가 아니라 `a2`가 가리키는 곳에 저장된다. 즉 `a2`가 가리키는 `main()` 함수의 변수 `x2`에 제곱결과가 저장된다. 따라서 `square()` 함수의 리턴 후에 `x2`는 `x`의 제곱결과를 갖는다.

10.2 키패드 입력 받기

키패드는 그림 10.2와 같이 푸쉬 버튼을 행렬로 배열한 것이다. 그럼 10.3은 키패드의 회로를 보여준다. 여기서는 포트 F의 하위니블을 열을 따라 푸쉬 버튼과 연결하였고 행을 따라 풀업을 하여 포트 F의 상위니블과 연결하였다. 그럼 10.3과 같이 PF0~PF3은 출력포트이고 PF4~PF7은 입력포트로 설정된다.

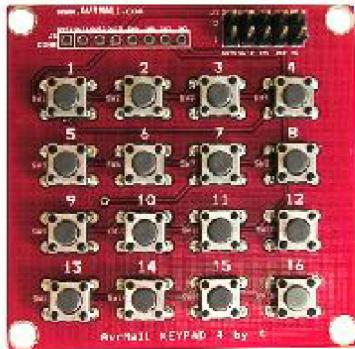


그림 10.2 키패드

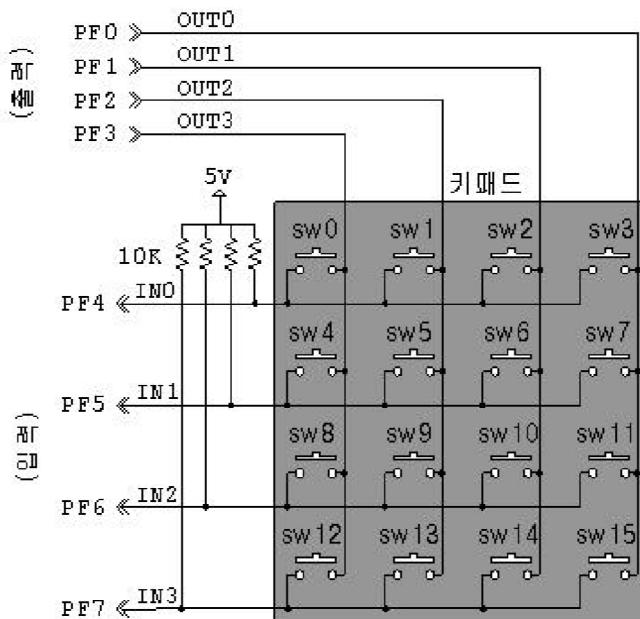


그림 10.3 키패드 연결회로도

그림 10.4와 같이 출력 포트에 PF3:0=0b1101을 출력하여 보자. 그럼 10.4에서 SW5과 SW10이 눌러진 상태이다. 이 때 전류 경로는 그림의 굵은 선을 따라 형성되어 입력포트 PF6에는 0이 읽히고 나머지 입력포트는 1이 읽힌다. 그림을 다시 살펴보면 0이 출력되지 않은 열에 위치한 버튼(예: SW1, SW5)은 개폐에 관계없이 입력 측의 데이터에는 영향을 미치지 못한다. 그러나 0이 출력되는 열에 위치한 버튼은 SW10과 같이 누른 상태가 되면 해당 행에 해당하는 입력비트가 0이 되고 SW6과 같이 떨어진 상태가 되면 해당하는 입력비트에 1이 읽힌다. 즉 PF1에 0을 출력함으로써 PF1과 연결된 열의 버튼 상태를 감시할 수 있다.

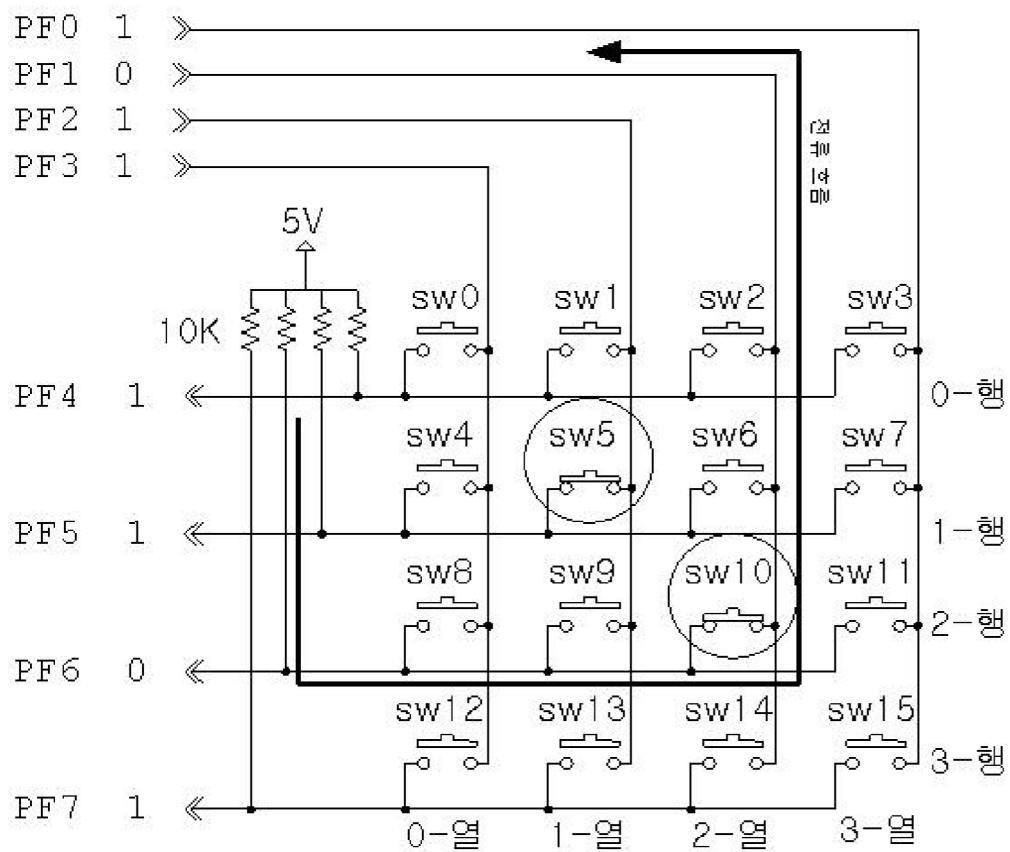


그림 10.4 출력과 키의 상태에 따른 입력값

표 10.1 버튼 코드값

버튼	입력값	출력값	코드값	버튼	입력값	출력값	코드값
	D7:D4	D3:D0			D7:D4	D3:D0	
SW0	~0x10	~0x08	0x18	SW8	~0x40	~0x08	0x48
SW1	~0x10	~0x04	0x14	SW9	~0x40	~0x04	0x44
SW2	~0x10	~0x02	0x12	SW10	~0x40	~0x02	0x42
SW3	~0x10	~0x01	0x11	SW11	~0x40	~0x01	0x41
SW4	~0x20	~0x08	0x28	SW12	~0x80	~0x08	0x88
SW5	~0x20	~0x04	0x24	SW13	~0x80	~0x04	0x84
SW6	~0x20	~0x02	0x22	SW14	~0x80	~0x02	0x82
SW7	~0x20	~0x01	0x21	SW15	~0x80	~0x01	0x81

* 코드값은 입력값과 출력값을 각각 비트별 NOT을 취하고 두 값을 더하여 얻는다.

출력 포트에서 0을 출력하는 팁을 PF0->PF1->PF2->PF3로 순환하면서 그 때의 입력을 읽으면 각 열에서 어느 행의 버튼이 눌렸는지 판단할 수 있다. 순환하면서 키를 검사하는 것을 키 스캔(Key Scan)이라 한다. 출력데이터 PF3:0와 입력데이터 PF7:4를 조합하면 전체 16개 버튼 중 어느 것이 눌렸는지 판단할 수 있다.

표 10.1은 각 버튼 상태를 검사하기 위한 출력 값과 버튼이 눌렸을 때의 입력값을 나타낸다. 표 10.1에서는 비트별 NOT값으로 표현하였다. 각 버튼의 코드 값은 입력값과 출력값을 각각 비트별 NOT를 취하여 두 값 더하여 얻었다. 키패드를 스캔하여 표 10.1의 코드값을 얻는 것은 프로그램 10.3과 같이 작성 할 수 있다.

```

out = 0x01;           // 3-열부터 스캔
for(i=0; i<4; i++)   // 스캔 시작
{
    PORTF = ~out;      // 출력 포트에 스캔 출력을 낸다.
    asm("nop": :);    // 동기화를 위해 1사이클 지연
    in = (~PINF) & 0xF0; // 입력된 값의 상위 4비트를 취한다.
    if(in)             // 키 입력이 있음
    {
        in += out;     // 출력과 입력을 조합하여 코드생성
        break;          // 출력이 있으므로 루프 탈출
    }
    out <<= 1;          // 다음 스캔코드로 변경
}

```

프로그램 10.3

- ☞ 변수 out는 출력포트에 출력할 데이터의 비트별 NOT을 한 값이다. 루프를 돌 때마다 원쪽으로 1비트를 이동하여 다음 열을 스캔한다.
- ☞ 변수 in은 입력포트로 읽은 값 중 상위니브만을 취한 값이다.
- ☞ in에 0이 아닌 값이 읽히면 버튼이 눌러 졌음을 의미하므로 출력과 입력을 조합(in += out)를 하고 루프를 빠져 나온다.
- ☞ 여기서는 입력이 파악되면 바로 루프를 빠져나오므로 한 행에서 두 개 이상의 버튼이 눌려 있을 때 처음 발견되는 버튼만 입력된다. 그러나

한 열에 두 개 이상의 버튼이 눌려 있으면 두 버튼이 눌려 있는 값으로 입력된다. 한 열에서도 단 하나의 버튼 입력을 갖도록 하려면 그에 대한 처리를 부가적으로 해주어야 한다.

- ☞ 같은 포트 출력과 포트 입력 사이에는 1사이클의 지연을 반드시 주어야 한다.

[프로그램 keypad.c]은 프로그램 10.3을 바탕으로 스캔을 하면서 키пад입력을 받는 프로그램을 완성한 것이다.

- ☞ 초기화 함수 KeyInit()에서 입출력 포트의 방향을 설정한다.
- ☞ 함수 key_scan()은 스캔을 하면서 현재 눌러진 버튼의 코드값을 반환한다.
- ☞ 디바운싱을 위하여 key_scan()은 시간지연과 함께 두 번을 읽고 두 번 읽은 값이 같으면 입력을 받아들인다.
- ☞ 입력함수 KeyInput()에서는 같은 버튼을 계속 누르고 있는 경우 입력이 되지 않는 것으로 판단한다. 이를 위해 정적변수 pin을 도입하여 이전 코드값을 저장한다.
- ☞ 헤더파일에는 코드값을 표 10.1의 버튼 명으로 매크로를 정의하였다.
- ☞ 함수 key_scan()은 파일 keypad.c에서만 사용되므로 정적함수로 작성하였다.

☞ 정적 함수

정적변수와 유사하게 함수가 정의된 파일에서만 함수가 통용되고 외부에는 함수의 존재를 숨길 때 사용할 수 있다.

함수의 타입 앞에 static을 쓰면 정적함수가 된다.

```
예) static int func(int x, int y)
{
    . .
}
```

[프로그램 lab10-2.c]는 키패드 입력을 받아 다음 동작을 한다.

- (1) sw1을 누르면 7-세그먼트에 표시된 수를 1 증가한다.
- (2) sw2를 누르면 7-세그먼트에 표시된 수를 2 증가한다.
- (3) sw3을 누르면 7-세그먼트에 표시된 수를 3 증가한다.
- (4) 여러 개의 스위치가 눌러지면 무시한다.

실습 프로그램 prac10-2.c와 keypad.c를 프로젝트에 포함시키고 키패드에서 sw1, sw2, sw3를 누르면서 프로그램을 수행을 한다.

* **JTAG** 인터페이스는 포트F를 사용하므로 실습을 수행하려면 **JTAG**를 불능시켜야 한다. 실습 수행이 안 되는 경우 5.4.6절에 따라 **JTAG**를 불능시킨다.

과제 :

1. 그럼 10.3에서 풀업저항을 제거하고 prac10-2.c를 수행하여도 정상적으로 동작을 한다. 이유를 설명하라.
2. 10.1.2절의 버튼입력과 같이 키패드의 버튼을 0.5초 이상 누르면 계속 입력하는 것으로 받아들여 100msec마다 누른 키에 해당하는 증분으로 수를 증가시키도록 prac10-2.c와 keypad.c를 변경하라.
3. [프로그램 prac10-4.c]를 키패드의 입력을 받아 LED의 밝기를 조절하는 프로그램으로 변경하라.(계속 누르는 입력도 받아들이도록 할 것)

[프로그램 prac10-2.c]

```

//=====
// 실습 10.2 : 키패드 입력
//=====

//
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include "iseg7.h"          // 7-세그먼트 구동
#include "keypad.h"         // 키패드 입력

void msec_delay(int n); // 시간지연함수

int main()
{
    unsigned char key;      // 입력 값
    unsigned short data = 0;

    ISeg7Init(); // 7-세그먼트 초기화
    KeyInit();   // 키패드 초기화
    sei();       // 인터럽트 허용

    ISeg7DispNum(data, 10);
    while(1)
    {
        key = KeyInput();

        switch(key)
        {
            case SW1:           // 스위치 1
                data +=1;
                break;
            case SW2:           // 스위치 2
                data += 2;
                break;
            case SW3:           // 스위치 3
                data += 3;
        }
    }
}

```

```
        break;
    default:
        break;
    }
    ISeg7DispNum(data, 10);
}

void msec_delay(int n)           // 시간지연함수
{
    for(; n >0; n--)
        _delay_ms(1);           // 1msec 시간지연을 n회 반복
}
```

[헤더파일 keypad.h]

```
#ifndef __KEYPAD_H__
#define __KEYPAD_H__

// 키패드 버튼 코드 값
#define SW0      0x18    // 0-번째 행
#define SW1      0x14
#define SW2      0x12
#define SW3      0x11
#define SW4      0x28    // 1-번째 행
#define SW5      0x24
#define SW6      0x22
#define SW7      0x21
#define SW8      0x48    // 2-번째 행
#define SW9      0x44
#define SW10     0x42
#define SW11     0x41
#define SW12     0x88    // 3-번째 행
#define SW13     0x84
#define SW14     0x82
#define SW15     0x81
#define NO_KEY   0x00

// 함수 정의
void KeyInit(void);
unsigned char KeyInput(void);

#endif
```

[소스파일 keypad.c]

```

//=====
//  Keypad Matrix 인터페이스
//
//  포트 핀      D7  D6  D5  D4  D3  D2  D1  D0
//  키 패드      I3  I2  I1  I0  O3  O2  O1  O0
//
//      In : 키패드 입력 핀
//      On : 키패드 출력 핀
//=====

#include <avr/io.h>
#include <util/delay.h>
#include <stdio.h>

#include "keypad.h"

#define N_COL    4          // 스캔 할 키패드 열 수

#define KEY_OUT PORTF      // keypad 출력 포트
#define KEY_IN  PINF       // keypad 입력 포트
#define KEY_DIR DDRF       // keypad 입출력 방향 레지스터

static unsigned char key_scan(void);
static unsigned char pin = 0;      // 이전 코드값

void KeyInit()
{
    KEY_OUT = 0xF0;    // 출력 초기 값으로 0을 출력
    KEY_DIR = 0x0F;    // 하위니블 -> 출력, 상위니블->입력
}

//=====
//  키패드의 스캔 코드 값을 반환한다.
//
//      리턴 값 : 0이 아닐 때 : 스캔 코드 값
//                  0일 때      : 입력 값이 없음
//=====

```

```

unsigned char KeyInput(void)
{
    unsigned char in, in1;

    in = key_scan(); // 키 스캔을 하여 코드값을 읽는다.

    while(1)
    {
        // 디바운싱 시간지연
        _delay_ms(10); _delay_ms(10); _delay_ms(10);
        in1 = key_scan(); // 한 번 더 읽는다.
        if(in==in1) break;
        in = in1;
    }

    if(!(in & 0xF0)) // 눌러진 키가 없음
    {
        pin =0;
        return 0;
    }

    if(pin == in) // 같은 키를 계속 누르고 있음
        return 0; // 새로운 키 입력이 없는 것으로 리턴

    pin = in; // 키 값을 저장 함.

    return in; // 코드를 반환한다.
}

//-----
// 스캔 코드값을 리턴함
//-----
static unsigned char key_scan(void)
{
    unsigned char out, i, in;

    out = 0x01; // 3-열부터 스캔

```

```
for(i=0; i<N_COL; i++)
{
    KEY_OUT = ~out;           // 출력 포트에 스캔 출력을 낸다.
    asm("nop");              // 1사이클 지연

    in = (~KEY_IN) & 0xF0;      // 입력된 값의
                                // 상위 4비트를 취한다.
    if(in)                    // 키 입력이 있음
    {
        in += out;            // 출력과 입력을 조합하여 코드 생성
        break;                 // 출력이 있으므로 루프 탈출
    }
    out <<= 1;                  // 다음 스캔 코드값으로 변경
}

return in;                      // 스캔 코드 값 반환
}
```