**임베디드 소프트웨어 Lab 4 실습보고서**

2011104043 컴퓨터공학과 장원엽

1. 강의 내용

이번 시간에는 스위치의 사용법과 인터럽트에 대해서 배웠다. 이전까지 하던 프로그래밍 방식은 polling방식으로, 메인 프로그램에서 상시 사용자의 입력을 기다리는 구조였다. 이러한 방식은 시스템 자원의 효율성을 떨어뜨린다. 이를 해결하기 위해 인터럽트 방식을 사용한다. 인터럽트 방식은 메인 프로그램이 사용자의 입력을 기다리지 않고 자신의 할 일을 수행하다가 사용자의 입력이 생길 경우에 현재 실행중인 작업을 잠시 중단하고 해당 처리 코드로 이동하여 사용자의 입력에 대한 처리를 수행하는 방식이다.

JKit에서 스위치는 PE 포트이며, INT 신호에 연결되어 있다. 때문에 소스코드상에서 스위치가 눌렸음을 알기 위해서는 PINE의 값을 체크하거나 인터럽트 코드를 입력해주어야 한다.

JKit에서 인터럽트를 처리하기 위해서는 몇 가지 레지스터를 미리 설정해주어야만 한다. SREG는 Status REGister의 줄임말로, CPU의 상태를 표시하는 레지스터이다. 8bit로 구성되어 있으며, 7번째 비트를 1로 설정해줌으로써 Global Interrupt를 Enable시킬 수 있다. 이 같은 기능을 하는 함수를 avr/interrupt.h 헤더에서 지원해주는데, 인터럽트의 on, off를 각각 sei(), cli()로 처리할 수 있다.

다음으로 EIMSK(External Interrupt MaSK register)가 있다. JKit에는 총 7가지의 인터럽트가 있으며 이 중 어떤 인터럽트를 enable할 것인지를 설정하는 레지스터이다. 스위치의 경우 1번 스위치와 2번 스위치가 각각 4번 인터럽트와 5번 인터럽트에 해당하기 때문에 00110000의 8bit로 값을 설정해 줄 수 있다.

다음으로 EICRA, EICRB 레지스터가 있다. 이 레지스터들은 각각의 인터럽트가 Low level, 하강 에지, 상승 에지 중 어느 곳에서 발생할 지를 설정해 준다. 각각의 인터럽트마다 2개의 bit가 순차적으로 할당되며 스위치의 경우, EICRB의 0, 1번째 bit(4번 인터럽트)와 2, 3번째 bit(5번째 인터럽트)를 사용한다. 두 값이 00이면 Low level, 10이면 하강 에지, 11이면 상승 에지에서 인터럽트가 발생하게 된다.

1. 실습 코드

#define F\_CPU 16000000UL

#include <avr/io.h> // ATmega128 register 정의

#include <util/delay.h>

int main(void)

{

DDRA = 0xff;

DDRE &= 0xf3;

while (1)

{

PORTA = PINE; // SW1 = PE bit4, SW2 = PE bit5

\_delay\_ms(100);

}

}

위 코드에서 알 수 있다시피, PORTA에 PINE값을 대입하면 해당하는 8bit값을 LED로 확인할 수 있다. 결과 JKit의 초기 설정상, 스위치 1번을 입력 시 00100000이 입력되며, 스위치 2번을 입력 시 00010000이 입력됨을 확인할 수 있었다.

#define F\_CPU 16000000UL

#include <avr/io.h> // ATmega128 register 정의

#include <util/delay.h>

int main(void)

{

DDRA = 0xff;

PORTA = 0x00;

DDRE &= 0xcf;

int led = 0;

int value = 0x00;

while (1)

{

if(led == 0){

if(PINE == 0xef){

led = 1;

value = 0xff;

PORTA = value;

}

if(PINE == 0xdf){

led = 1;

value++;

PORTA = value;

}

}

if(led == 1){

if(PINE == 0xef){

led = 0;

value = 0x00;

PORTA = value;

}

if(PINE == 0xdf){

value++;

PORTA = value;

}

}

}

}

위 코드는 폴링방식으로 LED를 제어하는 코드이다. PINE의 값을 체크하여 스위치가 눌렸는지 확인하며, PORTA값을 변경해 줌으로써 LED를 제어할 수 있다.

#define F\_CPU 16000000UL

#include <avr/io.h>

#include <avr/interrupt.h>

#include <util/delay.h>

volatile static int num = 0;

// switch 1

SIGNAL(SIG\_INTERRUPT4){

cli( );

if(PORTA == 0x00){

PORTA = 0xff;

}

else{

PORTA = 0x00;

}

\_delay\_ms(100);

sei( );

}

// switch 2

SIGNAL(SIG\_INTERRUPT5){

cli( );

num++;

\_delay\_ms(100);

sei( );

}

void main(){

static unsigned char FND\_DATA[ ]= {

0x3f, // 0

0x06, // 1

0x5b, // 2

0x4f, // 3

0x66, // 4

0x6d, // 5

0x7d, // 6

0x27, // 7

0x7f, // 8

0x6f, // 9

0x77, // A

0x7c, // B

0x39, // C

0x5e, // D

0x79, // E

0x71, // F

0x80, // .

0x40, // -

0x08 // \_

};

DDRA = 0xff;

PORTA = 0x00;

DDRE = 0xcf;

DDRC = 0xff;

EICRA = 0x00;

EICRB = 0x0a;

EIMSK = 0x30;

SREG |= 0x80;

//4321

while(1){

PORTC = FND\_DATA[num%10];

PORTG = 0x01;

PORTC = FND\_DATA[(num/10)%10];

PORTG = 0x02;

PORTC = FND\_DATA[(num/100)%10];

PORTG = 0x04;

PORTC = FND\_DATA[(num/1000)%10];

PORTG = 0x08;

}

}

위 코드는 인터럽트 방식을 이용하여 스위치의 입력을 처리하는 코드이다. 스위치 1번은 인터럽트 4번에서, 스위치 2번은 인터럽트 5번에서 처리한다. 4번 인터럽트에서는 Led를 처리하며, 5번 인터럽트에서는 FND에 표시할 숫자를 1씩 증가시켜 주는 기능을 수행한다.

#define F\_CPU 16000000UL

#include <avr/io.h>

#include <avr/interrupt.h>

#include <util/delay.h>

volatile int onOff = 0;

volatile int num = 0;

volatile int display = 0;

volatile int start = 0;

SIGNAL(SIG\_INTERRUPT4){

cli( );

if(onOff == 0){

onOff = 1;

start = 1;

}

else{

onOff = 0;

display = num;

}

\_delay\_ms(100);

sei( );

}

SIGNAL(SIG\_INTERRUPT5){

cli( );

num = 0;

display = 0;

start = 0;

onOff = 0;

\_delay\_ms(100);

sei( );

}

void main(){

static unsigned char FND\_DATA[ ]= {

0x3f, // 0

0x06, // 1

0x5b, // 2

0x4f, // 3

0x66, // 4

0x6d, // 5

0x7d, // 6

0x27, // 7

0x7f, // 8

0x6f, // 9

0x77, // A

0x7c, // B

0x39, // C

0x5e, // D

0x79, // E

0x71, // F

0x80, // .

0x40, // -

0x08 // \_

};

DDRE = 0xcf;

DDRC = 0xff;

EICRA = 0x00;

EICRB = 0x0a;

EIMSK = 0x30;

SREG |= 0x80;

while(1){

if(onOff == 1){

PORTC = FND\_DATA[num%10];

PORTG = 0x01;

\_delay\_ms(2);

PORTC = FND\_DATA[(num/10)%10];

PORTG = 0x02;

\_delay\_ms(3);

PORTC = FND\_DATA[(num/100)%10];

PORTG = 0x04;

\_delay\_ms(2);

PORTC = FND\_DATA[(num/1000)%10];

PORTG = 0x08;

\_delay\_ms(3);

}

else{

PORTC = FND\_DATA[display%10];

PORTG = 0x01;

\_delay\_ms(2);

PORTC = FND\_DATA[(display/10)%10];

PORTG = 0x02;

\_delay\_ms(3);

PORTC = FND\_DATA[(display/100)%10];

PORTG = 0x04;

\_delay\_ms(2);

PORTC = FND\_DATA[(display/1000)%10];

PORTG = 0x08;

\_delay\_ms(3);

}

if(start == 1){

num++;

}

}

}

위 코드는 실습 응용 2, JKit 실습자료 5장 더 해보기 (실습 6의 스톱워치를 기본으로, 이 눌려지면 그 순간 시간이 정지한 상태로 디스플레이 하지만, 실제로는 스톱워치 가 시간을 계속 카운트하고 있어서, 이 다시 눌려지면 현재 시간부터 디스플레이 되도록 하는 프로그램을 작성하라.)의 코드이다. 실습 6번에서는 사용하지 않았던 몇 가지 flag들을 넣어 현재 상태를 체크하고 요구사항에 알맞게 동작하도록 하였다.

#define F\_CPU 16000000UL

#include <avr/io.h>

#include <util/delay.h>

volatile int num = 0;

void main(){

static unsigned char FND\_DATA[ ]= {

0x3f, // 0

0x06, // 1

0x5b, // 2

0x4f, // 3

0x66, // 4

0x6d, // 5

0x7d, // 6

0x27, // 7

0x7f, // 8

0x6f, // 9

0x77, // A

0x7c, // B

0x39, // C

0x5e, // D

0x79, // E

0x71, // F

0x80, // .

0x40, // -

0x08 // \_

};

DDRA = 0xff;

PORTA = 0x00;

DDRE = 0xcf;

DDRC = 0xff;

//4321

// 스위치 1번이 led4, 2번이 led3

// 스위치 1번 눌렸을 때 00100000

// 스위치 2번 눌렸을 때 00010000

while(1){

if(PORTA == 0x00){

if(PINE == 0x20 ){

PORTA = 0xff;

\_delay\_ms(100);

}

}

else{

if(PINE == 0x20){

PORTA = 0x00;

\_delay\_ms(100);

}

}

PORTC = FND\_DATA[num%10];

PORTG = 0x01;

PORTC = FND\_DATA[(num/10)%10];

PORTG = 0x02;

PORTC = FND\_DATA[(num/100)%10];

PORTG = 0x04;

PORTC = FND\_DATA[(num/1000)%10];

PORTG = 0x08;

if(PINE == 0x10){

num++;

\_delay\_ms(100);

}

}

}

위 코드는 실습 응용 3(실습 응용 1을 interrupt 대신 polling으로 구현)에 대한 코드이다. 기존에 인터럽트 방식으로 처리했던 것을, 메인 함수에서 while문 내부에 집어넣어 처리하였다. While문 내부에서는 PINE의 값을 체크하여 스위치가 눌렸는지를 항시 확인하고 있다.

#define F\_CPU 16000000UL

#include <avr/io.h>

#include <util/delay.h>

volatile int num = 0;

volatile int onOff = 0;

volatile int tempNum = 0;

void main(){

static unsigned char FND\_DATA[ ]= {

0x3f, // 0

0x06, // 1

0x5b, // 2

0x4f, // 3

0x66, // 4

0x6d, // 5

0x7d, // 6

0x27, // 7

0x7f, // 8

0x6f, // 9

0x77, // A

0x7c, // B

0x39, // C

0x5e, // D

0x79, // E

0x71, // F

0x80, // .

0x40, // -

0x08 // \_

};

DDRE = 0xcf;

DDRC = 0xff;

while(1){

if(onOff == 1){

PORTC = FND\_DATA[num%10];

PORTG = 0x01;

\_delay\_ms(2);

PORTC = FND\_DATA[(num/10)%10];

PORTG = 0x02;

\_delay\_ms(3);

PORTC = FND\_DATA[(num/100)%10];

PORTG = 0x04;

\_delay\_ms(2);

PORTC = FND\_DATA[(num/1000)%10];

PORTG = 0x08;

\_delay\_ms(3);

}

else{

PORTC = FND\_DATA[tempNum%10];

PORTG = 0x01;

\_delay\_ms(2);

PORTC = FND\_DATA[(tempNum/10)%10];

PORTG = 0x02;

\_delay\_ms(3);

PORTC = FND\_DATA[(tempNum/100)%10];

PORTG = 0x04;

\_delay\_ms(2);

PORTC = FND\_DATA[(tempNum/1000)%10];

PORTG = 0x08;

\_delay\_ms(3);

}

if(onOff == 0){

if(PINE == 0x20){

onOff = 1;

\_delay\_ms(100);

}

}

else{

if(PINE == 0x20){

onOff = 0;

tempNum = num;

\_delay\_ms(100);

}

}

if(PINE == 0x10){

num = 0;

onOff = 0;

tempNum = 0;

\_delay\_ms(100);

}

num++;

}

}

위 코드는 실습 응용 3번(실습 응용 2를 interrupt 대신 polling으로 구현)에 대한 코드이다. 마찬가지로 원래는 인터럽트로 처리했던 코드들을 while문 내부에 집어넣어 PINE의 값을 항시 체크하는 구조이다.