< 엔코더 읽어들이기 >

엔코더에 관련된 문서를 많이 쓰고 있는데, 이는 좀 조심스럽게 접근하고 있다는 증거겠죠... 게다가 시스템에서 상당히 중요한 부분이라고 생각하고 있기 때문입니다.

1. 기본적인 아이디어

기본적인 아이디어는, 앞 문서중의 '인크리멘탈 엔코더 읽기 방법 연구'에서 결론낸 것을 그대로 이용하기로 합니다. 그 절차를 간단히 적어 봅니다.

1. **입력 포트에 2개의 엔코더 신호선을 결선함** : Port A의 0,1,2,3 번 핀을 이용함
2. **일정시간마다 Port A의 값을 읽어들임** : 충분히 빠른 속도로 리프레쉬함
3. **읽은 값을 적절히 연산하여 원하는 데이터로 가공함** : 비트연산, 논리연산등

여기서 좀 더 생각해 볼 것은, Port A 에 4,5,6,7번 핀이 남으므로 이것을 다른 센서나 스위치 입력으로 이용할 수 있도록 고려해 주어야겠고...

또하나, 리프레쉬 시간을 얼마로 주는 것이 적절할지 생각해 보아야겠습니다. 단순히 볼 때, 33 슬릿 (66 PPR) 분해능 짜리가 최고 100 RPM 으로 돈다고 하면.... 대략 1초에 110번 정도 신호가 변화한다고 볼 수 있겠군요. 이 신호를 안전하게 읽어들일려면, 보통 2배이상의 속도로 읽어 들이는게 일반적이니, **220 Hz** 이상은 되어야겠군요.

음... 그리고, 일정시간마다 읽도록 하는 동작은, 8515의 타이머/카운터 0번을 이용하면 되겠구요.

마지막으로, ANSI C로 비트연산 하는 연산자에 관해 가물가물해지면.....

http://www.koreafluidmechanics.net/main/fm_md_c/cmd_c_03.html

를 일단 참조하도록 합시당..

2. 타이머/카운터0 사용하기

8515에서 내장된 2개의 타이머/카운터 중에 1번은 PWM 발생용으로 이미 사용했으니, 나머지 0번만 남았죠. 이 타이머/카운터 0번을 이용해서, 1/220이하 초마다 한번씩 인터럽트를 발생시키면 되죠? 이런 동작을 설정하기 위한 관련 레지스터를 우선 살펴보겠습니당.

TCCR0 - Timer / Counter Control Register 0

0x00 : 타이머/카운터 0을 정지시킴. Unable 상태로 만듦.

0x01 : 프리스케일링 안함. CK

0x02 : 프리스케일링. CK/8

0x03 : 프리스케일링. CK/64.

0x04 : 프리스케일링. CK/256.

0x05 : 프리스케일링. CK/1024.

0x06 : 외부 펄스 이용함. 핀 T0 (Port B의 0번핀) 사용. Falling Edge에서 트리거링 됨.

0x07 : 외부 펄스 이용함. 핀 T0 (Port B의 0번핀) 사용. Rising Edge에서 트리거링 됨.

네... 이 TCCR0 레지스터는, 보신 바와 같이 카운팅을 위한 클럭 소스를 어떻게 줄 거냐? 라는걸 설정해 주는 기능이 있군요. 이중에 하나 골라잡으면 되겠구요.

TCNT0 - Timer / Counter 0

요 레지스터는, 타이머/카운터 0 의 값을 나타내는 카운터 그 자체로군요. 0x00을 써주면, 리셋되는 것이고...

데이터 시트에 따르면, 이놈은 무조건 증가만 하는 '업 카운터'이며, 쓰기와 읽기 모두 자유롭다고 합니다.

만약 클럭 소스가 주어져 있는 상태에서, 이 레지스터에 어떤 값을 써주면... 그 값부터 카운팅이 클럭에 맞추어 장동으로 착착 올라가게 되니 원하는 카운팅 갯수를 마음대로 정해줄 수 있게 되겠습니다.. 증가하다가, 이 레지스터의 값이 가득차게 되면.... 오버플로우 인터럽트를 호출해서 인터럽트 루틴을 짠! 불러서 실행시켜주면 되고... 그 인터럽트 루틴 맨 마지막 줄에, 이놈에게 다시 원하는 값을 또 써주면... 계속 아무생각없이 동작하게 되겠군요.

단순한 MPU 같으니라구.... (^-^)

결론적으로, TCCR0 와 TCNT0 의 값의 조합을 통하여 인터럽트 루틴이 호출되는 주기를 정해줄 수 있겠다는 거로군요. 포켓봇에게 있어서 그건 곧 엔코더 값 읽어들이는 주기가 되겠죠?

자... 포켓봇에게 있어서 목표 주기가 220 Hz였죠.

TCNT0 의 리셋 값은 머리아프니까 그냥 0x00 으로 줍시다. 이놈의 크기가 8비트짜리 카운터니까, 256 클럭동안 카운팅 된다는 소리가 되겠죠... 그것은, TCCR0 레지스터에서 프리스케일링 해 준 주파수에서 256 나눈 값이 바로 주기가 된다는 소리구... 시스템 클럭은 4MHz 이니끼니...

$$T_S \geq \frac{256 \times C_{PRE}}{f_{CK}}$$

이로군요. 그러니까,

$$\frac{1}{220} \geq \frac{256 \times C_{PRE}}{4000000}$$

가 되므로, 필요한 프리스케일링 값은 ... 71 분주 이하가 되어야 하는군요. 조건에 맞는건 64분주 또는 8분주 인데, **64 분주**를 일단 택합니다. 64분주했을 경우, 샘플링 주기는 244.140625 Hz 가 되는군요! 이렇게 샘플링 주기를 가능한 범위 내에서 길게 잡는 이유는, 불필요하게 너무 빈번한 인터럽트를 걸어주는 것이 시스템의 속도를 떨어뜨리고 인터럽트 시간내에 모든 처리를 끝내지 못하게 되는 불상사가 생길 수도 있는 것 등입니다. 나중에 모든 시스템을 구성했을 때, 엔코더값 읽기 뿐 아니라 여러 가지 다른 일도 하게 될테니까요.

그리구... 인터럽트 말인데요. 타이머/카운터 오버플로우를 인터럽트의 트리거로 사용하기 위해 필요한 레지스터는 뭐가 있을지 함 알아봅시당.

TIMSK - Timer / counter Interrupt Mask Register

요 레지스터 하나로 타이머/카운터 0번 및 1번의 인터럽트를 Enable 시켰다가 Disable 시켰다가 합니다. 잠시 각 비트들이 뭐가 있는지 함 살펴보죠.

7	6	5	4	3	2	1	0
TOIE1	OICIE1A	OICIE1B	-	TICIE1	-	TOIE0	-

음... 타이머/카운터 0에 관계되는건 비트 1의 TOIE0 하나 뿐이로군요. 이건 타이머/카운터 0가 오버플로우했을 때 인터럽트가 가능하도록 인가해 주는 설정이죠. 이 비트의 값을 1로 셋팅하면 되겠습니당. 그러면 충실히 인터럽트가 걸리겠죠?

네.

이제 타이머/카운터 0와 인터럽트 걸기의 메커니즘을 대략 훑어 보았으니, AVR-GCC 가지고 코딩하는 걸 생각해 봅시당.

뭐 타이머/카운터 0을 의도한 대로 설정하는 것을 만들어 볼 수 있겠군요. 우선 앞에서 살펴본 대로 두 개의 설정 레지스터를 셋팅 하죠.

```
outp(0x00,TCNT0);       /* 타이머/카운터 0을 일단 리셋 */

outp(0x03,TCCR0);       /* 프리스케일링함... 64분주 */
```

그리구... 카운터가 오버플로우 되었을 때 인터럽트가 걸리도록 TIMSK 레지스터를 셋팅할 필요가 있겠습니다. 폴커 오트씨가 AVR-GCC 배포판의 DEMO 디렉토리 안에 넣어 놓은 예제2 에는 다음과 같이 써 놓았더군요.

```
outp((1<<TOIE0), TIMSK);
```

이건, TOIE0 라는 상수가 헤더파일 어딘가에 이미 0x02로 정의되어 있을테니, 1이란 숫자를 0x02비트 밀어내어 TIMSK를 설정해 주는 건데요. 하지만, AVR-GCC 의 "interrupt.h" 파일(="signal.h") 안에, "void timer_enable_int(uint8_t ints)" 라는 함수가 이미 정의되어 있더군요. 이거 불러다 쓰면 더 알아보기 쉽겠구만. 쩝. 그래서 다시 적어보믄...

```
include < signal.h >

   ...

timer_enable_int (0x02);
```

해주면 되겠습니당. 앞의 폴커 오트씨의 예제와 같은 효과가 되겠죠?

자, 이제 타이머/카운터 0이 계속 돌면서 약 244 Hz 주기로 계속 인터럽트 서비스 루틴을 호출할 겁니다. 근데 그 인터럽트 서비스 루틴은 대체 멉니까? 이름을 알아야 써먹든 말든 할텐데...

AVR-GCC에서 제공하는 인터럽트 서비스 루틴 함수는, INTERRUPT() 또는 SIGNAL() 이라는 이름을 가지고 있습니당. 대문자로 이름지어진 것에 주의합시당. 그리고 인자(Argument) 에 인터럽트 벡터를 적어주면 되죠. 이건 당근 상수화 되어 있겠죠? 음... 찾아보니, 타이머/카운터 0 의 오버플로우 인터럽트는 이름이 "SIG_OVERFLOW0" 로군요. 그래서...

```
SIGNAL(SIG_OVERFLOW0)

{

   ...할일들...

   outp(0x00,TCNT0);          /* 담번에 또 오버플로우 시킬꺼니깐, 리셋해줌*/

}
```

요런 인터럽트 서비스 루틴을 사용하면 되겠습니당. 물론 이 인터럽트를 완전히 실행가능하도록 할려면, 먼저 sei() 함수를 써서 인터럽

트 인가 상태로 만들어 놓아야 겠죠?

음... 설명을 막 헷갈리게 해서 죄송합니다. 저도 잘 몰라서... 효과적으로 설명할 수가 없네요... 에구... 일단 넘어갑시당.

3. 비트 연산하기

암튼, 앞에 나온 것처럼 엔코더의 신호를 읽어들였다 칩시다. 이제 이것을 원하는 형태의 데이터로 가공해야 써먹을 수 있겠죠.

우선, 한가지 가정을 해 둡니다. 즉 들어오는 신호는 무조건 1씩 증감하지, 2이상씩 증감하지 않는다...는 것입니다. 아까 그렇게 할려고 220 Hz라고 결정했던 것이구요. 이건 회전속도가 100 RPM 넘어가지 않는 이상 거의 보장되는 가정이라 해 둘 수 있겠습니다. 물론 신호가 미쳐 날뛸땐... 어쩔 수 없겠죠.

이 절차는 다음과 같이 정해 봅니다.

1. **Port A 읽기** : 여기에 결선되어 있으니까.
2. **비트 0,1 과 2,3을 따로 분리하기** : 좌우측 엔코더를 분리해서 다루어야죠.
3. **Gray Code → Binary Code 로 변환** : 이전의 글을 참조바랍니당. 0132 → 0123 순서로 바꾸는거죠?
4. **+1 또는 -1 의 증감 여부를 판별** : 이번 값과 이전 값의 차이로 증감여부를 알 수 있겠군요.
5. **적산(적분)하기** : 어떤 변수에 증감값을 적산해서, 총 이동거리를 알 수 있도록 해야겠죠?

이걸 바탕으로 일단 유사코드(Pseudo code)를 적어봅니다.

```
SIGNAL(SIG_OVERFLOW0)
{
    sensor = inp(PINA);        /* Port A 읽기 */
    e_current_L = 0b00000011 & sensor;        /* 좌측 엔코더 값만 분리 */
```

```
        e_current_R = (0b00001100 & sensor) >> 2;        /* 우측 엔코더 값만 분리 */
        e_current_L ^= e_current_L>>1;        /* 좌측 엔코더 코드 변환 : 1비트 우측으로 쉬프트 시킨 것과 XOR */
        e_current_R ^= e_current_R>>1;        /* 우측 엔코더 코드 변환 : 1비트 우측으로 쉬프트 시킨 것과 XOR */
        e_flag_L = e_current_L - e_past_L;        /* 좌측 엔코더 증감여부 판별 */
        e_flag_R = e_current_R - e_past_R;        /* 우측 엔코더 증감여부 판별 */
        if (e_flag_L == 3) {e_flag_L = -1;}        /* 좌측 엔코더 예외 처리 */
        else if (e_flag_L == -3) {e_flag_L = 1;}
        if (e_flag_R == 3) {e_flag_R = -1;}        /* 우측 엔코더 예외 처리 */
        else if (e_flag_R == -3) {e_flag_R = 1;}
        e_L += e_flag_L;        /* 좌측 엔코더값 적분 */
        e_R += e_flag_R;        /* 우측 엔코더값 적분 */
        e_past_L = e_current_L;        /* 좌측 엔코더 지난번 읽은값으로 저장하기 */
        e_past_R = e_current_R;        /* 우측 엔코더 지난번 읽은값으로 저장하기 */
        outp(0x00,TCNT0);
    }
```

음... 제가 써놓고도 무슨 말인지 잘 모르겠군요.... 헛헛헛... 이대로 하면 동작 될 것 같긴 한데, 변수가 남발되고 노는 비트들이 많군요. 행수도 많으니 클럭도 많이 잡아먹고... 머리 좀 더 굴려서 더 콤펙트하게 할 수 있을 것 같기도 한데... 지금은 별로 아이디어가 없군요.

일단 함 해 봅니다.

4. 소스 코드

이상의 조각조각 나누어진 것들을 모아서 테스트 가능한 소스 코드를 짜 보았습니다.

Port A의 0,1,2,3 번 핀에 엔코더의 1A,1B,2A,2B 신호가 각각 들어가고... 좌측 엔코더 값은 Port B로, 우측 엔코더 값은 Port D로 출력됩니다. 각 핀에 LED를 달아서 확인해 볼 수 있겠습니다.

```
/* 2001. 12.  김동호... 엔코더 읽기 테스트 */

#include <io.h>

#include <interrupt.h>

#include <signal.h>


// 에러코드

#define GOOD    0


unsigned char sensor=0;


// 엔코더 관련 변수들

unsigned char e_current_L=0, e_current_R=0;  /* 최근에 읽은 값 */

unsigned char e_past_L=0, e_past_R=0;        /* 지난번 읽은 값 */
```

```
char e_flag_L=0, e_flag_R=0;                /* 증감 판별 값 */
char e_L=0, e_R=0;                          /* 적분 값 */


// 각 포트 입출력 설정
int port_init(void)
{
    outp(0x00,DDRA);        /* Port A 읽기 상태로 */
    outp(0xff,DDRB);        /* Port B 쓰기 상태로 */
    outp(0xff,DDRD);        /* Port D 쓰기 상태로 */
    return GOOD;
}


// 타이머 초기화
int timer_init(void)
{
    timer_enable_int(0x02);  /* TIMSK 설정, 타이머/카운터 0의 오버플로우 인터럽트 인가 */
    outp(0x00,TCNT0);        /* 타이머/카운터 0을 일단 리셋 */
    outp(0x03,TCCR0);        /* 프리스케일링함... 64분주 */
    return GOOD;
```

```
}


// 엔코더 읽기
int e_read(void)
{
    e_current_L = 0x03 & sensor;        /* 좌측 엔코더 값만 분리 */
    e_current_R = (0x0C & sensor) >> 2; /* 우측 엔코더 값만 분리 */
    e_current_L ^= e_current_L>>1;      /* 좌측 엔코더 코드 변환 : 1비트 우측으로 쉬프트 시킨 것과 XOR */
    e_current_R ^= e_current_R>>1;      /* 우측 엔코더 코드 변환 : 1비트 우측으로 쉬프트 시킨 것과 XOR */
    e_flag_L = (char)(e_current_L - e_past_L);  /* 좌측 엔코더 증감여부 판별 */
    e_flag_R = (char)(e_current_R - e_past_R);  /* 우측 엔코더 증감여부 판별 */
    if (e_flag_L == 3) {e_flag_L = -1;} /* 좌측 엔코더 예외 처리 */
    else if (e_flag_L == -3) {e_flag_L = 1;}
    if (e_flag_R == 3) {e_flag_R = -1;} /* 우측 엔코더 예외 처리 */
    else if (e_flag_R == -3) {e_flag_R = 1;}
    e_L += e_flag_L;                    /* 좌측 엔코더값 적분 */
    e_R += e_flag_R;                    /* 우측 엔코더값 적분 */
    e_past_L = e_current_L;             /* 좌측 엔코더 지난번 읽은값으로 저장하기 */
    e_past_R = e_current_R;             /* 우측 엔코더 지난번 읽은값으로 저장하기 */
```

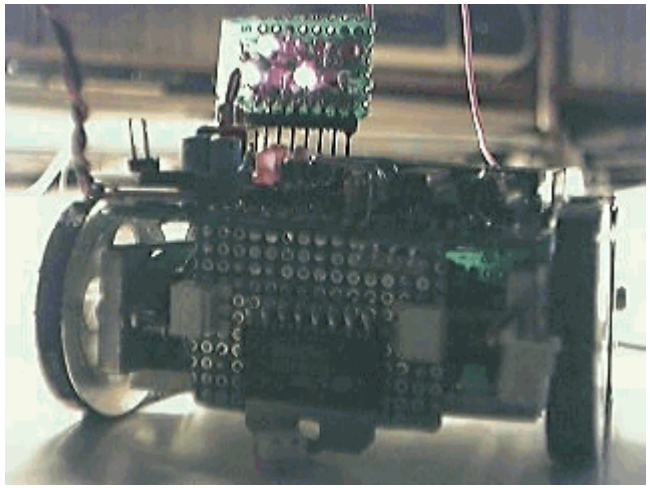
```
    return GOOD;
}

// 인터럽트 서비스 루틴
SIGNAL(SIG_OVERFLOW0)
{
    sensor = inp(PINA);     /* Port A 읽기 */
    e_read();
    outp(e_L,PORTB);        /* 좌측 엔코더 적분값 출력 */
    outp(e_R,PORTD);        /* 우측 엔코더 적분값 출력 */
    outp(0x00,TCNT0);
}

int main(void)
{
    port_init();
    timer_init();
    sei();
    for (;;) {}
```

```
    }
```

5. 테스트



Encoder Counting Test

위의 프로그램을 콤파일해서 심어넣고 돌려보니... 엔코더값이 깨끗하게 잘 읽히고 있습니다. 확인은, LED를 달아붙여 했는데 사진에서 불이 밝혀진 부분을 볼 수 있습니다. 발통을 손으로 돌리면, LED 불빛이 차례대로 챠르륵 적산되어 올라갑니다. LED는 어떻게 붙였냐구요? 헐헐... 8515의 다리 윗부분에다가 임시로 납땜해서 붙였죠... 테스트 끝나면 또 납을 녹여서 떼어내구... 좀 무식한...

암튼 이거 한 번만에 되니 기분이 좋군요. 한 번만에 된게 이번 포켓봇 진행하는 중 처음일까???... 정말 그런 것 같네요... ㅠ_ㅠ;

6. 결론

이제, 모터도 마음대로... 엔코더도 마음대로 읽을 수 있으니... 본격적으로 피드백 제어를 할 수 있으리라 기대됩니당. 이를 구성하기 위한 기본적인 토대는 마침내 갖추었으니까요. 다만, 엔코더의 분해능이 부족하고 정확한 물성치들도 모르는 상태이므로, 정량적인 설계 과정 보다는 시행착오에 의존한 노가다성 작업이 될 공산이 크죠.

암튼, 연말에 이런 재미없는 글 읽어주시는 분들께 감사드리구요.

제 설명 능력이 부족한 관계로, 이해가 잘 안되시는 부분이나 잘못된 점이 있다면 언제든지 비평 바랍니다.

대학교 1~2학년 정도의 학생이 충분히 이해할 수 있는 수준에 맞추려고 노력하고 있습니다.