에디로봇아카데미

임베디드 마스터 Lv1 과정

Ch6.

제 3기

2022. 2. 26

여건영

CONTENTS

* 수업 review

* pipeline

Interrupt 활용하기

* interrupt를 활용하여 button Switch의 상태 변화에 따른 장치 제어해보기 (예 : LED)

// interrupt 란 ?

- 디바이스가 입력/출력/에러의 상황에 대한 프로세서의 처리를 요구하는 것.
- 마이크로프로세서(CPU)가 프로그램을 실행하고 있을 때, 입출력 HW 등의 장치나 또는 예외상황이 발생하여 처리가 필요할 경우 이를 CPU에 알려 처리를 할 수 있도록 하는 것

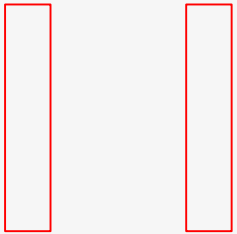
// ISR (Interrupt Service Routine)

- CPU는 인터럽트를 감지하면 지금 실행중인 코드를 중단하고 해당 인터럽트 처리 코드로 점프하여 일을 수행, 이러한 인터럽트 처리를 위한 루틴을 ISR, Interrupt Service Routine 이라고 함
- 원래 System이 하던 일이 있음에도 잠시 짬을 내서 급한 것을 먼저 처리해 주는 거니까, 아주 짧고 간결하고 굵게 코드를 구성하자

- ISR() 함수와 다른 함수 (예 : main(), adc_value_alloc()) 들과 벼수를 공유할 경우
  → 반드시 전역변수 사용
  위 변수가 ISR() 내부에서 값을 갱신된다면? Volatile 키워드를 붙여 선언한다.

Interrupt 활용하기

* 입력 : PIND.5 - button Switch
  출력 : PORTB.5 - LED
  // Switch의 상태 변화에 따라 LED가 켜지고 꺼짐



Interrupt 활용하기

* 코드분석

// DDB4에 1을 넣음으로써

PORTB.4을 출력으로 사용

// DDD5에 0을 넣음으로써

PIND.5을 입력으로 사용

Interrupt 활용하기

* 코드분석

// PCIE2 : Pin Change Interrupt Enable 2
- PCIE2 bit가 1, 상태 레지스터(SREG)의 I-bit(Global Interrupt Flag)가 1로 set되면
  PCIE2가 Enable 된다.
  어떤 PCINT16 ~ 23 중 하나가 enable 되고, 그의 상태가 변한다면 Interrupt를 발생 시킨다.

  해당 Interrupt 요청은 PCI2 인터럽트 벡터에서 실행된다.
  PCINT16 ~ 23 pin들은 각각 PCMSK2 레지스터에 의해 enable 될 수 있다.

*** PCIE1은 PCINT8 ~ 14
  PCIE0은 PCINT0 ~ 7 로 위에 자리에 대응 될 수 있

Interrupt 활용하기

* 코드분석

// 각 PCINT16 ~ 23 bit는 핀 상태 변경 감 인터럽트가
어떤 해당 I/O(여기서 PIND.5)에서 활성화되는지 여부를 선택한다.

PCINT0 ~ 23 pin을 enable 함으로서 해당 pin의 상태 변화를 감지하여 Interrupt 활성화할 수 있으
clear함으로써 disable 할 수 있다.

Interrupt 활용하기

* 코드분석

즉 입력으로 설정한 PIND.5의 상태가 변화할 때마다
Interrupt 요청이 되며
ISR을 수행한다.

ISR 내부에서 PORTB.4를 toggle(XOR) 시키며 LED를 제어함

각 ISR의 Vector name – Interrupt definition은
아래를 사이트를 참조함
https://ece-classes.usc.edu/ee459/library/documents/avr_intr_vectors/

Pipeline 내용 보충

struct_ex2.c

1. set_both_link 함수를 실행 하기 전, 구조체 test1, tset2의 정보 확인
2. set_both_link 함수 내에서 구조체 target1, target2 정보 확인
3. set_both_link 함수 실행 후, 구조체 test1, tset2의 정보 변화 확인

struct_ex3.c

* set_both_link(&tset1, &test2) 함수를 사용하여 test1.link와 test2.link에 각 상대 구조체의 주소값을 넣어보자

* 결과는? 변한 것을 볼 수 있다.
  test1, test2의 주소값을 받아 접근한다.
  set_both_link 함수는 struct 타입 데이터의 주소값을 저장하는 매개변수를 입력 받는다.
  여기서 target1, target2는 그냥 주소값을 갖고있는 변수 한 칸이다.
  포인터 변수를 통해 해당 주소로 직접 접근하여 구조체 데이터를 바꿀 수 있었다.

  아래와 같이 확인해보자.

struct_ex3.c

1. 구조체 test1, test2 data 값 출력
2. set_link 함수 실행 후, 구조체 test.link에 각 구조체 자신의 주소값이 들어갔는지 확인하고
   그 주소값을 통해 해당 구조체 내부에 data를 출력하여 본다.
3. set_both_link 함수 실행 후, 구조체 test.link에 각 상대 구조체의 주소값이 들어갔는지 확인하고
   그 주소값을 통해 해당 구조체 내부에 data를 출력하여 본다.

struct_ex4.c

* set_both_link(&test1 ,&test) 함수 실행 후, 각 구조체.link에 상대 구조체의 시작 주소값이 들어간다.
  // test1 = { 3, test2 시작주소}, test2 = { 7, test1 시작주소}

  While 문 동작에서, 처음 한번만 test1.link, 0xbe959910(test2 시작주소)가 가리키는 data 7이 출력되고,
  다시 test1.link 안으로 test2.link 값 0xbe959900(test1 시작주소) 값이 들어간다.
  // 현재 test1 = { 3, test1 시작주소}

  반복을 실행하며, 현재 test1.link(test1 시작주소)가 가리키는 data 값 3이 출력되며
  다시 test1.link 안으로 test1.link가 가리키는 link 값(test1 시작주소)이 들어간다.
  // test1 = { 3, test1 시작주소}
  반복 실행..

struct_ex5.c

* test1 = { 3, test1 시작주소}, test2 = { 7, test2 시작주소}
  tmp = { test1 시작주소 }
  set_both_link(&test1, &test2) 실행 후,
  test1 = { 3, test2 시작주소}, test2 = { 7, test1 시작주소}

  While 문 동작에서, tmp가 가리키는 data 3이 출력되고,
  tmp에 tmp가 가리키는 link, test2 시작주소 가 들어간다. // tmp = { test2 시작주소 }

  다음, tmp가 가리키는 data 7이 출력되고,
  tmp에 tmp가 가리키는 link, test1 시작주소 가 들어간다. // tmp = { test1 시작주소 }
  반복 수행...

pof.c

pof2.c

pof3.c

* arbiter()를 단순히 변수라고 생각하여
  arbiter() => p로 치환하여,
  Int (* p)(void);

  코드를 작성하였을 때,
  Compile error 부분
  그리고 그 이유에 대해 좀더 고민해 보겠습니다;