Arduino Interrupt

Overview

attachInterrupt(digitalPinToInterrupt(pin),ISR,mode);
: 핀 번호, 아두이노 우노의 경우 2,3 (블루투스 또한 2,3)
: 인터럽트가 발생하면 하드웨어적으로 호출되는 인터럽트 처리 함수
: 인터럽트 발생 조건, LOW, CHANGE, RISING, FALLING 중 하나

전역 인터럽트 플래그는 status register(SREG) 의 I bit에서 유지됩니다.

인터럽트 제어는 인터럽트 context내에서 실행되는 코드에 의해 변경될 수 있는 객체에 정밀한 접근을 할 때 주의가 필요한 경우가 많습니다.
: <Util/atomic.h>를 참조하십시오.

방해 받지 않고 특정 작업을 수행하기 위해 일정 시간 동안 인터럽트가 비활성화 되는 경우가 많습니다.
: 컴파일러 최적화와 관련하여 고려해야 할 사항은 코드 재정렬 문제를 참조하세요.
: avr-libc: Compiler optimization (nongnu.org)

Properties

Header

<avr/io.h>
: 각 MCU에 맞는 레지스터와 핀 번호를 선언하고 있는 헤더
: Arduino uno는 ATMEL MEGA 328P를 사용

Function

cli(), sei()
: 전역 인터럽트 마스크를 셋팅하여 인터럽트들이 불가능하게 만듭니다.
: 이 함수는 실제 하나의 어셈블리 라인 안에서 컴파일됩니다.
: 때문에 함수 오버헤드가 없습니다.

: 하지만 매크로는 또한 추가적인 최적화 손실을 유발 할 수 있는 메모리 배리어를 암시합니다.
: multi –byte objects에 정밀한 접근을 하기 위하여 cli() 과 sei() 를 수동으로 사용하는 것 보단 <util/atomic.h> 에 있는 매크로를 사용하는 걸 고려하십시오.

Identifier

__volatile__
: C 컴파일러에서 최적화(Optimization) 옵션을 켜면 다른 쓰레드에서 변환되어 쓰여져야 하는 값이 바뀌지 않는다.
: 때문에 사용하는 변수라면 volatile을 선언하여 최적화하지 않게 해야 한다.

Detailed Description

Introduction to avr-libc's interrupt handling

인터럽트 코드를 처리하는 방법에 동의하는 컴파일러를 찾는 것은 거의 불가능에 가깝습니다. C 언어 이래로 기계의존적인 방법에서 멀어지려 시도했고 각각의 컴파일러 제작자들은 그 방법을 디자인 하는 것에 집중했습니다.

AVR-GCC 환경에서 vector table 은 이미 지정된 이름이 있는 인터럽트 루틴을 가리키도록 미리 정의되어있습니다. 적절한 이름을 사용함으로써, 당신의 루틴은 해당되는 인터럽트가 발생했을 때 호출됩니다.

벡터 테이블 안으로 패치 하는 것은 문제의 한 부분일 뿐입니다. 컴파일러는 보통 컴파일러가 만들어낸 코드를 실행할 때 레지스터 집합을 사용합니다. 이런 레지스터와 상태 레지스터를 저장하고 복원하는 것이 중요합니다. 이를 수행하는 데 필요한 추가 코드는 인터럽트 기능에 __attribute__((signal)) 태그를 지정하여 활성화 됩니다.

이런 세부사항은 인터럽트 루틴들을 약간 지저분하게 만드는 것처럼 보이지만, 모든 이런 세부사항은 인터럽트 API 에 의해 처리됩니다. 인터럽트 루틴은 ISR()에 의해 정의됩니다. 이 매크로는 루틴을 지정된 주변기기에 대한 인터럽트 핸들러로 등록하고 표시합니다. 아래는 ADC 인터럽트 처리를 정의하는 예제입니다.

```
#include <avr/interrupt.h>
ISR(ADC_vect)
{
    // user code here
}
```

어셈블리 언어로만 작성된 인터럽트 루틴에 대한 설명을 참조하십시오.
: avr-libc: avr-libc and assembler programs (nongnu.org)

Catch-all interrupt vector

만약 예상치 못한 인터럽트가 발생한다면(interrupt is enabled and no handler is installed, which usually indicates a bug), 기본 액션은 리셋 벡터로 이동하여 디바이스를 리셋하는 것입니다. 당신은 *BADISR_vect* 라는 이름으로 지원되는 함수를 ISR()을 아래와 같이 재정의해 사용할 수 있습니다. (*BADISR_vect* 은 실제로 __vectordefault 인 alias 입니다.) 이 문자는 어셈블리 안에서 사용되고 <avr/interrupt.h>에 포함되어있지 않습니다.

```
#include <avr/interrupt.h>
ISR(BADISR_vect)
{
    // user code here
}
```

Nested interrupts

AVR 하드웨어는 SREG 레지스터안의 전역 인터럽트 플래그를 인터럽트 벡터 안에 입력하기 전에 비웁니다. 그러므로 일반적으로 인터럽트는 핸들러가 끝날 때까지 핸들러 안에서 비활성화 상태로 남아있으며, 여기서 RETI 명령은 결국 추가 인터럽트를 다시 활성화합니다. 이러한 이유로 인터럽트 핸들러는 일반적으로 중첩되지 않습니다. 대부분의 인터럽트 핸들러들은 이런 방법을 희망하고 몇몇은 무한한 재귀 인터럽트를 예방하기 위하여 요구합니다.(like UART interrupts,or level-triggered external interrupts).

드문 상황이지만 절대적으로 필요한 것 이상으로 다른 인터럽트가 지연시키지 않기 위해 가능한 한 빨리 전역 인터럽트 플래그를 다시 활성화 하는 것이 바람직할 수 있습니다. 이때 sei() 명령을 사용해 전역 인터럽트를 불가능하게 만들 수 있습니다. 아래와 같은 방법으로 핸들러를 정의함으로써 컴파일러는 인터럽트 핸들러의 시작부분에 SEI 명령어를 바로 삽입하는 명령을 할 수 있습니다.

```
ISR(XXX_vect, ISR_NOBLOCK)
{
  ...
}
```

인터럽트 벡터 XXX_vect 의 올바른 이름은 아래에 설명되어있습니다.

Two vectors sharing the same code

몇몇 상황에서, 두 개의 다른 인터럽트를 사용 해야 하는 액션들은 완전히 동일할 수 있으므로 하나의 ISR 로 충분합니다. 예를 들어, 두 개의 다른 포트들로부터 도착한 pin-change 인터럽트는 논리적으로 실제 발생된 포트로부터 비의존적인 이벤트 신호를 보낼 수 있습니다. ISR 매크로에 대한 ISR_ALIASOF() 속성을 사용하여 인터럽트 벡터 코드를 공유할 수 있습니다.

```
ISR(PCINT0_vect)
{
  ...
  // Code to handle the event.
}
ISR(PCINT1_vect, ISR_ALIASOF(PCINT0_vect));
```

ISR_ALIASOF() 기능을 사용하려면 GCC 4.2 이상(또는 GCC 4.1.x 의 패치 버전)이 필요합니다. 덜 우아하지만 모든 컴파일러 버전에 적용될 수 있는 구현에 대해서는 ISR_ALIAS() 매크로 문서를 참조하십시오.

Empty interrupt service routines

드문 상황에서, 인터럽트 벡터 안에 작동을 위한 어떠한 코드도 필요하지 않을 수 있습니다. 그 벡터는 인터럽트가 발생할 때 BADISR_vect 코드를 실행하지 않기 위해 어떻게든 정의 되어야 합니다.

이는 예를 들어 컨트롤러를 sleep_mode() 에서 벗어나게 할 목적으로만 활성화된 인터럽트의 경우일 수 있습니다.

이 처리는 인터럽트를 EMPTY_INTERRUPT() 매크로를 사용해 정의함으로써 사용가능합니다.

```
INTERRUPT(ADC_vect);
```

Manually defined ISRs

어떤 상황에서는 컴파일러가 생성한 ISR 의 프롤로그와 에필로그가 작업에

적합하지 않을 수 있으며 특히 인터럽트 처리 속도를 높이기 위해 수동으로 정의된 ISR 을 고려할 수 있습니다.

이에 대한 한 가지 솔루션은 전체 ISR 을 별도의(어셈블리) 파일에 수동 어셈블리 코드로 구현하는 것입니다. 그런 식으로 구현하는 방법의 예는 C 및 어셈블리 소스 파일 결합을 참조하세요.

: avr-libc: Combining C and assembly source files (nongnu.org)

또 다른 솔루션은 여전히 C 언어로 ISR 을 구현하지만 프롤로그와 에필로그를 생성하는 컴파일러의 작업을 인수하는 것입니다. 이것은 ISR() 매크로에 대한 ISR_NAKED 속성을 사용하여 수행할 수 있습니다.

컴파일러는 프롤로그나 에필로그로 아무 것도 생성하지 않으므로 실제 구현에서 최종 reti()를 제공해야 합니다. SREG 는 ISR 코드가 수정하는 경우 수동으로 저장해야 하며 컴파일러에서 암시하는 __zero_reg__가 항상 0 이라는 가정은 틀릴 수 있습니다(예: MUL 명령어 직후 인터럽트할 때).

```
ISR(TIMER1_OVF_vect, ISR_NAKED)
{
  PORTB |= _BV(0);  // results in SBI which does not affect SREG
  reti();
}
```

Choosing the vector : Interrupt vector names

인터럽트는 다음 표의 기호 중 하나를 제공하여 선택됩니다.

현재 벡터의 이름을 지정하기 위한 두 가지 다른 스타일이 있습니다. 한 형식은 SIG_로 시작하는 이름을 사용하고 그 뒤에 비교적 장황하지만 인터럽트 벡터를 설명하는 임의로 선택한 이름이 옵니다. 이것은 버전 1.2.x 까지 avr-libc 에서 유일하게 사용 가능한 스타일이었습니다.

avr-libc 버전 1.4.0 부터 두 번째 스타일의 인터럽트 벡터 이름이 추가되었습니다. 여기서 벡터 설명에 대한 짧은 문구 뒤에 _vect 가 옵니다. 짧은 문구는 각 장치의 데이터시트(Atmel 의 XML 파일)에 설명된 대로 벡터 이름과 일치하며 공백은 밑줄로 대체되고 영숫자가 아닌 다른 문자는 삭제됩니다. 접미사 _vect 를 사용하는 것은 유사한 명명 규칙을 사용하는 AVR 에 사용할 수 있는 다른 C 컴파일러로의 이식성을 개선하기 위한 것입니다.

역사적인 이름 지정 스타일은 향후 릴리스에서 더 이상 사용되지 않을 수 있으므로 새 프로젝트에는 권장되지 않습니다.

메모
ISR() 매크로는 실제로 전달된 인수의 철자를 검사할 수 없습니다. 따라서 ISR() 호출에서 아래 이름 중 하나의 철자를 틀리면 인터럽트 함수로 사용할 수 있지만 실제로 인터럽트 벡터 테이블에 연결되지 않는 함수가 생성됩니다. 컴파일러는 의심스럽게 보이는 ISR() 함수의 이름(즉, 매크로 교체 후 "__vector_"로 시작하지 않는 이름)을 감지하면 경고를 생성합니다.

Code

<avr/interrput.h>

```

```

Reference

avr-libc: <avr/interrupt.h>: Interrupts (nongnu.org)