Configuración de interrupciones megaAVR®

El desarrollador de la aplicación debe inicializar cuidadosamente la operación de interrupción de AVR®. Esta página resume los pasos clave de inicialización y uso necesarios para usar interrupciones en una aplicación. Se proporciona más información sobre el uso de interrupciones en la sección **Módulo de interrupción de la biblioteca (http://www.nongnu.org/avr-libc/user-manual/group__avr__interrupts.html)AVR-LIBC (http://www.nongnu.org/avr-libc/)** .

(http://www.nongnu.org/avr-libc/user-manual/group__avr__interrupts.html) (http://www.nongnu.org/avr-libc/)

Paso 1. #incluye encabezados estándar

La aplicación debe incluir archivos de encabezado `avr/io.h` y `avr/interrupt.h` como se muestra a continuación:

```c
#include <avr/io.h>
#include <avr/interrupt.h>
```

El archivo de encabezado `avr/interrupt.h` proporciona varias macros destinadas a simplificar la aplicación de interrupciones en una aplicación, como macros para habilitar/deshabilitar interrupciones globalmente (bit I en el registro de estado), así como una macro para asignar una interrupción. función a un vector de interrupción específico:

- `si( )`
- `CLI( )`
- `ISR(vector_id, atributos)`

Las macros **vector_id** se definen en el archivo de encabezado específico del procesador (incluido a través de `avr/io.h`), así como en la hoja de datos del dispositivo. Su construcción se define a continuación.

Paso 2. Proporcionar rutina de servicio de interrupción

Una función de manejo de interrupciones es diferente a una función ordinaria en que maneja el contexto guardar y restaurar para asegurar que al regresar de la interrupción, se mantenga el contexto del programa. También se usa una secuencia de código diferente para regresar de estas funciones.

Hay varias acciones que el compilador debe realizar para generar una rutina de servicio de interrupción:

- Se debe indicar al compilador que use una forma alternativa de instrucción de retorno (`RETI` vs. `RET`)
- Se debe informar al compilador sobre cualquier opción adicional específica
  - Habilitar el anidamiento de interrupciones
  - Opciones para la generación de código de prólogo/epílogo
- La función debe vincularse a un vector de interrupción específico.

Se proporcionan varios atributos de función de controlador al desarrollador de la aplicación, lo que habilita estas opciones.

- La macro `ISR( )` se proporciona para facilitar la definición de funciones de manejo de interrupciones con atributos

Para todos los vectores de interrupción sin controladores específicos, se instalará un controlador de interrupción predeterminado: **el controlador de interrupción predeterminado restablecerá el dispositivo** . Una aplicación puede anular el controlador predeterminado y proporcionar un controlador de interrupción predeterminado específico de la aplicación utilizando **BADISR_vect** vector_id dentro de la macro `ISR( )` .

Macro ISR( )

El siguiente ejemplo de código muestra cómo usar la macro `ISR()` para definir una función de interrupción:

```c
ISR(vector_id, ISR_[BLOCK|N
{
    /* Hardware auto-clears t
    /* Clear the cause of the
    /* ISR-specific processir
}
```

Los diversos parámetros se describirán ahora con más detalle.

id_vector

Este identificador es una *concatenación* de un **Vector Source ID** y **_vect** . Los ID de fuente de vector se encuentran en la hoja de datos del dispositivo, como se muestra (parcialmente) en el siguiente ejemplo para ATmega328PB:

16.1. Interrupt Vectors in ATmega328PB

Table 16-1 Reset and Interrupt Vectors in ATmega328PB

Vector No	Program Address	Source	Interrupts definition
1	0x0000	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 0
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2_COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2_COMPB	Timer/Coutner2 Compare Match B
10	0x0012	TIMER2_OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1_CAPT	Timer/Counter1 Capture Event
12	0x0016	TIMER1_COMPA	Timer/Counter1 Compare Match A
13	0x0018	TIMER1_COMPB	Timer/Coutner1 Compare Match B
14	0x001A	TIMER1_OVF	Timer/Counter1 Overflow
15	0x001C	TIMER0_COMPA	Timer/Counter0 Compare Match A
16	0x001E	TIMER0_COMPB	Timer/Coutner0 Compare Match B
17	0x0020	TIMER0_OVF	Timer/Counter0 Overflow
18	0x0022	SPI0 STC	SPI1 Serial Transfer Complete
19	0x0024	USART0_RX	USART0 Rx Complete
20	0x0026	USART0_UDRE	USART0, Data Register Empty
21	0x0028	USART0_TX	USART0, Tx Complete
22	0x002A	ADC	ADC Conversion Complete

(/local--files/8avr:interrupts-mega-configuration/vector-source-id-328pb.png)

Los vector_ids mal escritos **aún generarán una función** , sin embargo, no se conectará a la tabla de vectores de interrupción. El compilador generará una advertencia si detecta un nombre sospechoso.

Atributos

`Los atributos ISR( )` proporcionan más instrucciones al compilador sobre cómo configurar la función de interrupción.

ISR_BLOCK

Las interrupciones globales son inicialmente deshabilitadas por el hardware AVR al ingresar al ISR. Esta configuración **no modifica** este estado.

Este atributo es **idéntico a una macro `ISR( )` sin atributo especificado**

ISR_NOBLOCK

ISR se ejecuta con interrupciones globales habilitadas inicialmente. El compilador activa el indicador de habilitación de interrupción lo antes posible dentro de la ISR para garantizar un retraso de procesamiento mínimo para las interrupciones anidadas.

Esto se puede usar para crear ISR anidados, sin embargo, se debe tener cuidado para evitar desbordamientos de pila o para evitar ingresar infinitamente al ISR en aquellos casos en los que el hardware AVR no borre el indicador de interrupción respectivo antes de ingresar al ISR.

ISR_NAKED

ISR se crea sin código de prólogo o epílogo. El código de usuario es responsable de la preservación del estado de la máquina, incluido el registro SREG, así como de colocar un `reti()` al final de la rutina de interrupción.

ISR_ALIASOF(id_vector)

Esto se puede usar para definir vectores adicionales que comparten el mismo controlador. El siguiente ejemplo crea un alias del vector PCINT1 para el controlador PCINT0:

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

Ejemplo de ISR( )

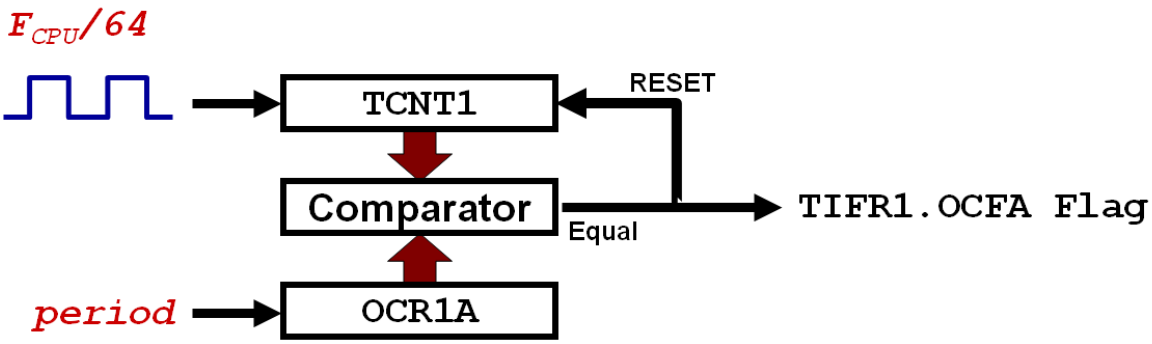
En este ejemplo de código, destacamos los archivos de encabezado requeridos y la definición ISR correcta de una función de controlador para la fuente de interrupción del modo Timer/Counter1 Clear-Timer-On-Compare (CTC). El controlador alterna **LED0** en el **ATmega328PB Xplained Mini (/boards:atavr328)** cada 100 mS:

```
 1  #include <avr/io.h>?
 2  #include <avr/interru
 3
 4  ISR(TIMER1_COMPA_vect
 5  {
 6      PORTB ^= (1 << PO
 7  }
 8
 9  int main(void)
10  {
11      // Initializatior
12
13      // Set LED as out
14      DDRB |= (1 << POR
15      PORTB &= ~(1 << P
dieciséis
17      // Set up Timer/C
18      TCCR1B |= (1 << W
19      OCR1A = 25000;
20
21      TIMSK1 |= (1 << O
22      TCCR1B |= ((1 <<
23
24      // Enable all int
25      sei();
26
27      while(1);
28  }
```

Paso 3. Configurar el periférico

A continuación, debe configurar el periférico para generar eventos de solicitud de interrupción.

Por ejemplo, el ATmega328PB contiene varios módulos periféricos de temporizador/contador. Cada módulo tiene un modo llamado **Clear Timer on Compare** (CTC) que, cuando se inicializa correctamente, activará periódicamente una señal de **indicador de coincidencia de comparación de salida del** temporizador 1 en el registro de indicador de interrupción TC1 (TIFR1), como se muestra:



(/local--files/8avr:interrupts-mega-configuration/timer1-ctc-int-flag.png)

En este ejemplo, inicializaremos Timer/Counter1 en modo CTC para generar solicitudes de interrupción cada 100 mS, dada una entrada preescala de 250 kHz (16 MHz/64):

```c
#include <avr/io.h>?
#include <avr/interru

ISR(TIMER1_COMPA_vect
{
    PORTB ^= (1 << PO
}

int main(void)
{
    // Initializatior

    // Set LED as out
    DDRB |= (1 << POR
    PORTB &= ~(1 << P
dieciséis
    // Set up Timer/C
    TCCR1B |= (1 << W
    OCR1A = 25000;

    TIMSK1 |= (1 << O
    TCCR1B |= ((1 <<

    // Enable all int
    sei();

    while(1);
}
```

Este es un ejemplo de una interrupción no persistente (/8avr:interrupts-mega-overview#non-persistent-interrupts) . El indicador TIFR1.OCFA se borra automáticamente por el hardware al ingresar al controlador.

El indicador TIFR1.OCFA también se puede borrar manualmente escribiendo un "1" lógico en la ubicación del bit.

Paso 4. Habilitar todas las interrupciones

Finalmente, debemos habilitar globalmente todas las interrupciones periféricas habilitadas configurando el **bit I de habilitación de interrupción global** en el **registro de estado (SREG)** . La biblioteca de interrupciones AVR-LIBC proporciona dos funciones de macro útiles para esto:

- `sei()` para habilitar interrupciones globalmente
- `cli( )` para deshabilitar las interrupciones globalmente

```
 1  #include <avr/io.h>?
 2  #include <avr/interru
 3
 4  ISR(TIMER1_COMPA_vect
 5  {
 6      PORTB ^= (1 << PO
 7  }
 8
 9  int main(void)
10  {
11      // Initializatior
12
13      // Set LED as out
14      DDRB |= (1 << POR
15      PORTB &= ~(1 << P
dieciséis
17      // Set up Timer/C
18      TCCR1B |= (1 << W
19      OCR1A = 25000;
20
21      TIMSK1 |= (1 << O
22      TCCR1B |= ((1 <<
23
24      // Enable all int
25      sei();
26
27      while(1);
28  }
```

Aprende más

Resumen de interrupciones de megaAVR

Más información > (/8avr:interrupts-mega-overview)

Consideraciones especiales

Más información > (/8avr:interrupts-special-considerations)

Ejemplo de interrupción de megaAVR

Más información > (/8avr:interrupts-mega-example)