

Resumen de interrupciones de megaAVR®

La familia megaAVR® proporciona varias fuentes de interrupción diferentes, todas las cuales son enmascarables y se dividen en tres categorías:

- **Interrupciones periféricas internas**
 - Asociado con temporizadores, USART, SPI, periféricos ADC
- **Interrupciones de clavijas externas**
 - Asociado con los pines de interrupción externa INT0-INT7
- **Interrupciones de cambio de pin**
 - Asociado con interrupciones externas PCINT0-PCINT2 que ocurren en un cambio de pin de puerto

A los periféricos se les asignan **bits de habilitación de interrupción** individuales en su respectivo **registro de máscara de interrupción** que debe escribirse como uno lógico junto con el **bit I de habilitación de interrupción global** en el **registro de estado** para habilitar la interrupción.

Name: SREG
Offset: 0x5F
Reset: 0x00
Property: When addressing as I/O Register: address offset is 0x3F

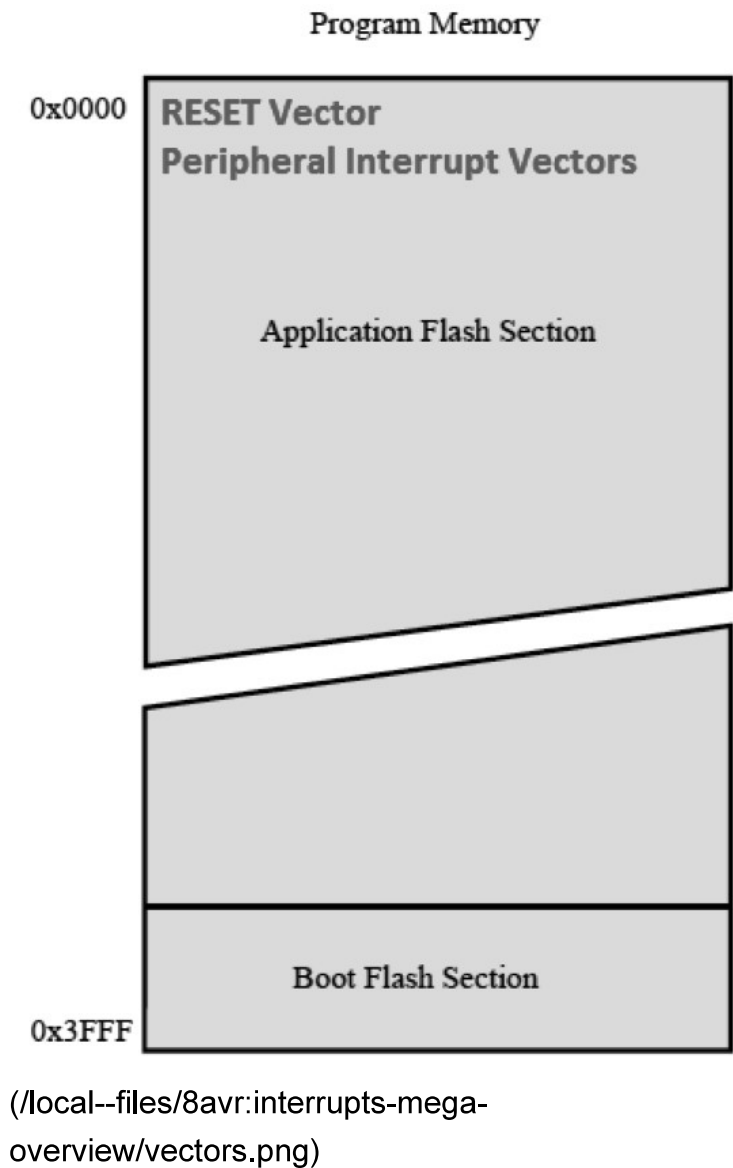
Bit	7	6	5	4	3	2	1	0
	I	T	H	S	V	N	Z	C
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

(/local--files/8avr:interrupts-mega-overview/status-register.png)

Restablecer e interrumpir ubicaciones de vectores

Cada una de las fuentes de reinicio e interrupción tiene un vector de programa separado en el **espacio de memoria** del programa . Las direcciones más bajas en el espacio de la memoria del programa se definen de manera predeterminada como

vectores de reinicio e interrupción, como se muestra:



Reubicación de vectores

El usuario puede reubicar el vector RESET así como la ubicación de inicio de los vectores de Interrupción en la **Sección Flash de Arranque** del espacio de la memoria del programa programando el bit de fusible **BOOTRST** en "0" y configurando el bit **IVSEL** del Registro de Configuración del Microcontrolador (**MCUCR**) en "1". Aquí se muestra la posible ubicación del vector de interrupción y RESET:

BOOTRST	IVSEL	Restablecer dir.	Dirección de inicio del vector de interrupción.
1	0	0x0000	0x0002
1	1	0x0000	Dirección de reinicio de arranque + 0x0002

0	0	Dirección de reinicio de arranque	0x0002
0	1	Dirección de reinicio de arranque	Dirección de reinicio de arranque + 0x0002

La **dirección de reinicio de arranque** se establece mediante bits de fusible BOOTSZ0/BOOTSZ1 como se muestra aquí para ATmega328PB:

Table 32-7 Boot Size Configuration, ATmega328PB

BOOTSZ1	BOOTSZ0	Boot Size	Pages	Application Flash Section	Boot Loader Flash Section	End Application Section	Boot Reset Address (Start Boot Loader Section)
1	1	256 words	4	0x0000 - 0x3EFF	0x3F00 - 0x3FFF	0x3EFF	0x3F00
1	0	512 words	8	0x0000 - 0x3DFF	0x3E00 - 0x3FFF	0x3DFF	0x3E00
0	1	1024 words	16	0x0000 - 0x3BFF	0x3C00 - 0x3FFF	0x3BFF	0x3C00
0	0	2048 words	32	0x0000 - 0x37FF	0x3800 - 0x3FFF	0x37FF	0x3800

(/local--files/8avr:interrupts-mega-overview/bootflash328pb.png)



Los fusibles se programan utilizando un **procedimiento de programación especial (/8avr:avrfuses)** dentro de Atmel Studio 7 u otro programador.



Para evitar cambios no intencionales de las tablas de vectores de interrupción, se debe seguir un procedimiento de escritura especial para cambiar el bit IVSEL:

- Escriba el bit de habilitación de cambio de vector de interrupción (IVCE) a uno.
- Dentro de cuatro ciclos, escriba el valor deseado en IVSEL mientras escribe un cero en IVCE.

Aquí hay un ejemplo de código que muestra cómo modificar el bit IVSEL y reubicar los vectores de interrupción:



```

1  <font></font>?
2  void move_interrupts(void)<font></font>
3  {<font></font>
4      uchar temp;<font></font>
5      /* GET MCUCR */<font></font>
6      temp = MCUCR;<font></font>
7      /* Enable change of Interrupts */
8      MCUCR = temp | (1 << IVCE);
9      /* Move interrupts to Boot Loader Section */
10     MCUCR = temp | (1 << IVSEL);
11 }<font></font>
12 <font></font>

```

Nivel de prioridad

Cada vector tiene un nivel de prioridad predeterminado: cuanto **menor** sea la dirección, **mayor** será el nivel de prioridad. RESET tiene la prioridad más alta, y el siguiente es INT0: la solicitud de interrupción externa 0. El siguiente gráfico muestra la lista de vectores parciales para la MCU 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/Counter2 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/Counter1 Compare Match B
14	0x001A	TIMER1_OVF	Timer/Counter1 Overflow
15	0x001C	TIMER0_COMPA	Timer/Counter0 Compare Match A
16	0x001E	TIMER0_COMPB	Timer/Counter0 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-overview/vectors328pb.png)

Procesamiento de interrupciones

Cuando ocurre una interrupción, el bit I de habilitación de interrupción global se borra y todas las interrupciones se desactivan. El bit I se establece automáticamente cuando se ejecuta una instrucción Return from Interrupt (RETI).



El software del usuario puede escribir uno lógico en el bit I para habilitar **interrupciones anidadas**. Todas las interrupciones habilitadas pueden interrumpir la rutina de interrupción actual.

Hay básicamente dos tipos de interrupciones:

Interrupciones persistentes

Este tipo de interrupción se activará siempre que la condición de interrupción esté presente. Estas interrupciones no necesariamente tienen banderas de interrupción. **Ejemplo: Interrupción de recepción completa de USART** El USART contiene un indicador de recepción completa (RXC) que se establece si hay datos no leídos en el búfer de recepción. Cuando se establece la habilitación de interrupción de recepción completa (RXCIE) en UCSRNb, la interrupción de recepción completa de USART se ejecutará siempre que el indicador RXC esté establecido (siempre que las interrupciones globales estén habilitadas). Cuando se utiliza la recepción de datos impulsada por interrupciones, la rutina de recepción completa debe leer los datos recibidos de UDR para borrar el indicador RXC; de lo contrario, se producirá una nueva interrupción una vez que finalice la rutina de interrupción.

Interrupciones no persistentes

Este tipo de interrupción se desencadena por un evento que establece un **indicador de interrupción**. Para estas interrupciones, el contador de programa se vectoriza al vector de interrupción real para ejecutar la rutina de manejo de interrupciones, y **el hardware borra el indicador de interrupción correspondiente**. Las banderas de interrupción también se pueden borrar escribiendo un uno lógico en la(s) posición(es) del bit de bandera que se va a borrar. Si se produce una condición de interrupción mientras se borra el bit de activación de interrupción correspondiente, el indicador de interrupción se establecerá y se recordará hasta que se habilite la interrupción o el software borre el indicador. De manera similar, si ocurren una o más condiciones de interrupción mientras se borra el bit de habilitación de interrupción global, los indicadores de interrupción correspondientes se establecerán y recordarán hasta que se establezca el bit de habilitación de interrupción global, y luego se ejecutarán por orden de prioridad. **Ejemplo: Timer/Counter0 Overflow Interrupt** Bit-0 del Timer0 Interrupt Flag Register (TIFR0) contiene el indicador de interrupción TOV0. Este indicador se establece cuando se produce un desbordamiento en Timer/Counter0. TOV0 es

borrado por el hardware al ejecutar el vector de manejo de interrupción correspondiente. Alternativamente, TOV0 se borra escribiendo un uno lógico en la bandera. Cuando se establecen el bit I de SREG, TOIE0 (habilitación de interrupción de desbordamiento del temporizador/contador0) y TOV0, se ejecuta la interrupción de desbordamiento del temporizador/contador0.

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:



```
1  #include <avr/io.h>           ?
2  #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:

- `sei()`
- `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:



```
1  ISR(vector_id, ISR_[BLOCK|N2?
2  {
3      /* Hardware auto-clears t
4      /* Clear the cause of the
5      /* ISR-specific processin
6  }
```

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/Counter2 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/Counter1 Compare Match B
14	0x001A	TIMER1_OVF	Timer/Counter1 Overflow
15	0x001C	TIMER0_COMPA	Timer/Counter0 Compare Match A
16	0x001E	TIMER0_COMPB	Timer/Counter0 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:



```
1  ISR(PCINT0_vect)
2  {
3      ...
4      // Code to handle the event
5  }
6  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:



```

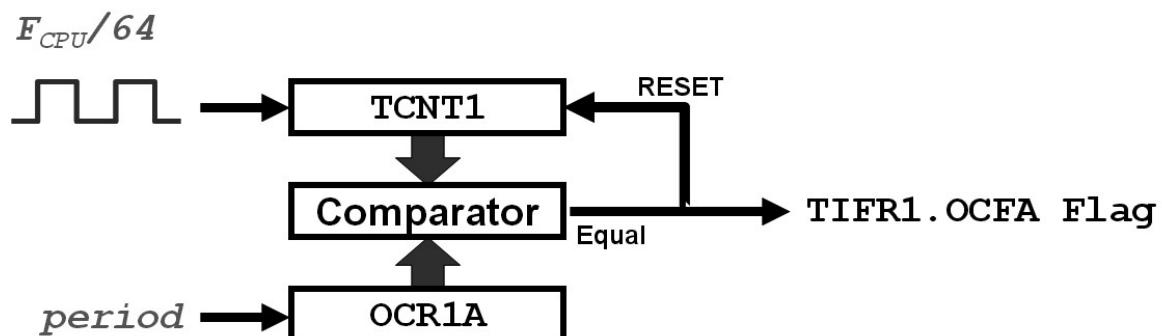
1  #include <avr/io.h>
2  #include <avr/interrupt.h>
3
4  ISR(TIMER1_COMPA_vect)
5  {
6      PORTB ^= (1 << PCRF);
7  }
8
9  int main(void)
10 {
11     // Initialization
12
13     // Set LED as output
14     DDRB |= (1 << PORTB);
15     PORTB &= ~(1 << PORTB);
16
17     // Set up Timer/Counter1
18     TCCR1B |= (1 << WGM12);
19     OCR1A = 25000;
20
21     TIMSK1 |= (1 << OCIF1);
22     TCCR1B |= ((1 << CS12) | (1 << CS11));
23
24     // Enable all interrupts
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):



```
1  #include <avr/io.h>?
2  #include <avr/interrupt.h>
3
4  ISR(TIMER1_COMPA_vect)
5  {
6      PORTB ^= (1 << PORTB);
7  }
8
9  int main(void)
10 {
11     // Initialization
12
13     // Set LED as output
14     DDRB |= (1 << PORTB);
15     PORTB &= ~(1 << PORTB);
16
17     // Set up Timer/Counter 1
18     TCCR1B |= (1 << WGM12);
19     OCR1A = 25000;
20
21     TIMSK1 |= (1 << OCIF1);
22     TCCR1B |= ((1 << CS12) | (1 << CS11));
23
24     // Enable all interrupts
25     sei();
26
27     while(1);
28 }
```



Este es un ejemplo de una interrupción no persistente (/8avr:interrupts-mega-overview#non-persistent-interrupts) . El indicador TIFR1.OCF1A se borra automáticamente por el hardware al ingresar al controlador.



El indicador TIFR1.OCF1A 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



Ejemplo de código de interrupción megaAVR®

🎯 Objetivo


Esta página proporciona un ejemplo de código de **interrupción** básico para la MCU **ATmega328PB** . (<http://www.microchip.com/wwwproducts/en/ATmega328PB>)El proyecto configura el módulo Timer/Counter1 para operar en modo **Clear-Timer-On-Compare** (CTC) y, en una coincidencia de período, genera un evento de interrupción cada 100 mS. El ISR manipula una variable de señal de "marca" que utiliza el bucle principal para alternar LED0 cada 100 mS.

✅ Materiales





Herramientas de ferretería

Herramienta	📖 Sobre
 (http://www.atmel.com/tools/MEGA328PB-XMINI.aspx)	Mini kit de evaluación ATmega328PB Xplained (http://www.atmel.com/tools/MEGA328PB-XMINI.aspx) (https://www.microchipdirect.com)

Herramientas de software

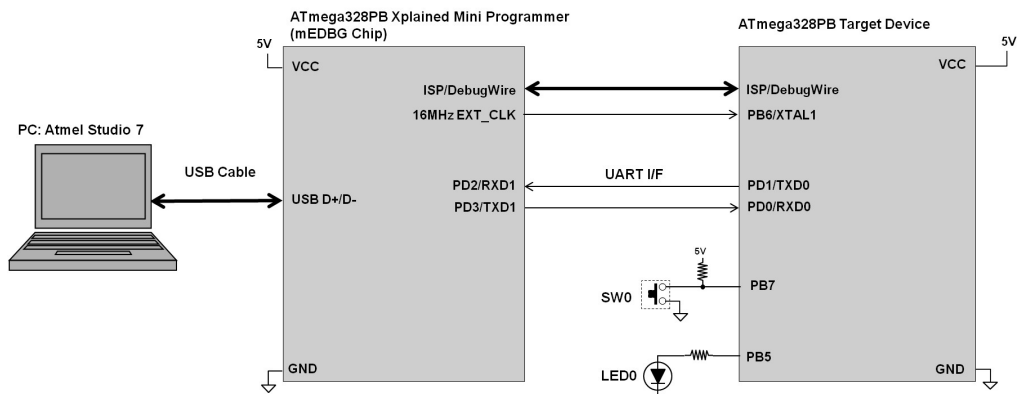
		Instaladores			
Herramienta	📖 Sobre	Ventanas		linux	Mac OS X
Entorno de desarrollo integrado Atmel® Studio		 (/atstudio:start)	 (http://studio.download.atmel.com/7.0.1931/as-installer-7.0.1931-full.exe)		 (/install:atstudio)

Archivos de ejercicios

		Descargar	
Expediente		Ventanas	linux
 Proyecto de ejemplo		 (/local--files/8avr:interrupts-mega-example/8avr-mega-int-usage.zip)	 (/local--files/8avr:interrupts-mega-example/8avr-mega-int-usage)
	Recomendamos extraer el archivo .zip a su carpeta C:\. Debería ver la carpeta C:\MTT\8avr\mega\code-examples\interrupt-example\8avr-mega-int-usage que contiene la solución 8avr-mega-int-usage.atsln		

🔗 Diagrama de conexión

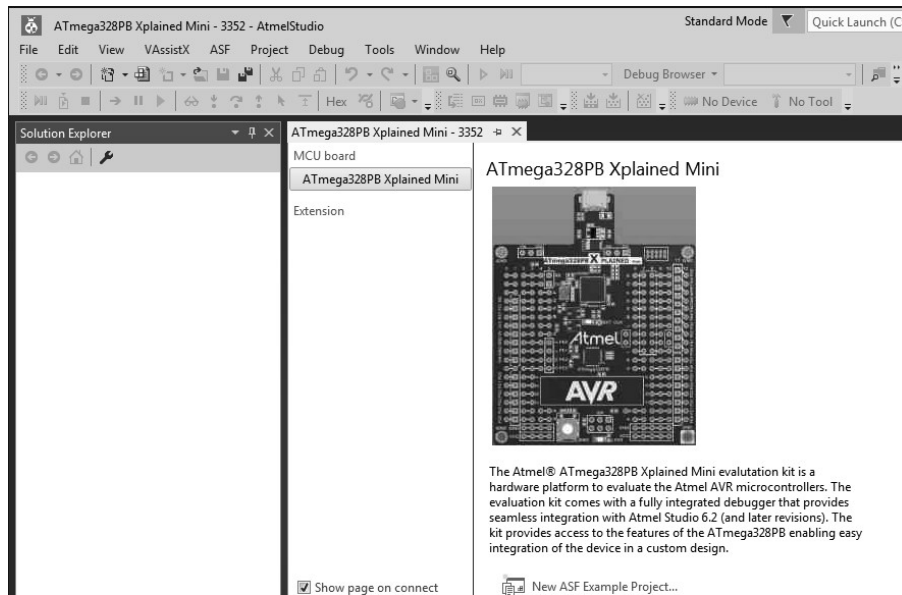
El módulo USART0 en el dispositivo ATmega328PB de destino está conectado a la interfaz USART en el chip mEDBG. El chip mEDBG realiza la conversión USB-serie enumerando como un puerto COM virtual de clase CDC en la PC y presentando los datos USART de destino en esta interfaz. El mEDBG también controla la interfaz de programación/depuración en el dispositivo de destino, además de proporcionar un reloj de 16 MHz cuando la placa Xplained está conectada mediante un cable USB a una PC. El LED0 está conectado al puerto PB5 como se muestra:



(/local--files/8avr:interrupts-mega-example/xplained-mini-connection-diagram-as7.png)

Procedimiento

Conecte la mini placa Xplained ATmega328PB a su computadora usando un cable USB A-a-MicroB. Inicie Atmel Studio 7. Si la placa se ha enumerado correctamente, debería ver la imagen de la placa en Studio como se muestra:

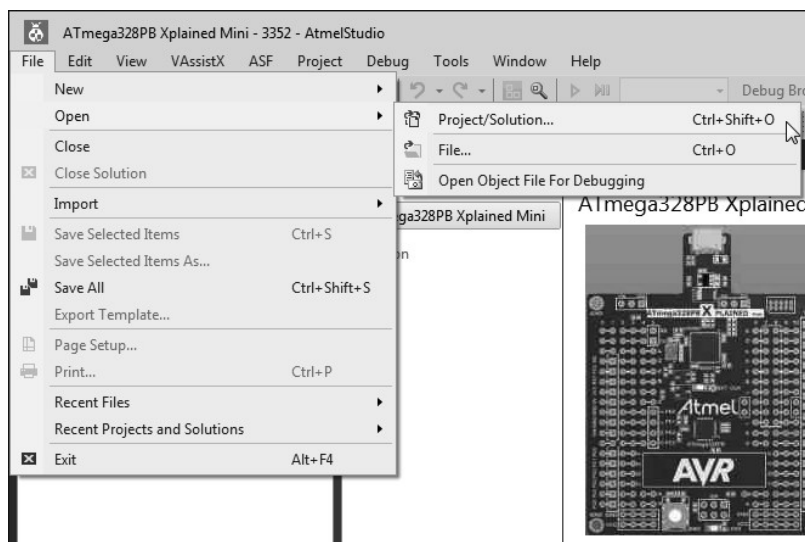


(/local--files/8avr:interrupts-mega-example/xplained-mini-enumeration-success.png)

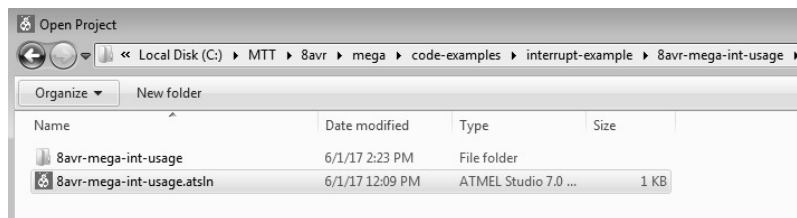


La placa se identifica por los últimos cuatro dígitos de su número de serie (consulte la etiqueta en la parte inferior de la placa). En el ejemplo anterior, los últimos cuatro dígitos son "3352"

1 Abra la solución



(/local--files/8avr:interrupts-mega-example/as7-open-solution.png)

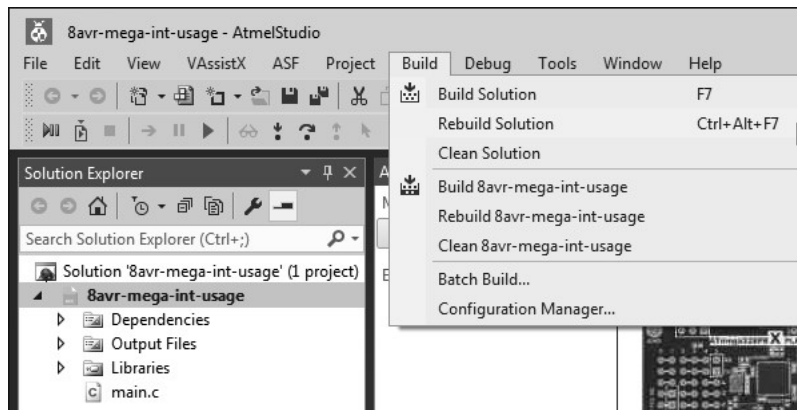


(/local--files/8avr:interrupts-mega-example/as7-open-solution-detail.png)



Para comprender cómo se configuraron y habilitaron las interrupciones en este ejemplo (archivo main.c), consulte la página **Configuración de interrupciones del megaAVR® (/8avr:interrupts-mega-configuration)** .

2 Reconstruir la solución



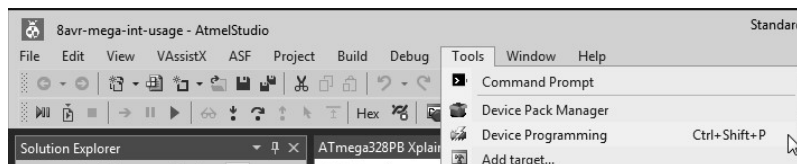
(/local--files/8avr:interrupts-mega-example/as7-rebuild-solution.png)

3 programa los fusibles

Hay varios ajustes de configuración de hardware clave que deben configurarse. Los siguientes **ajustes (/8avr:avrfuses)** de fusibles deben programarse en el dispositivo:

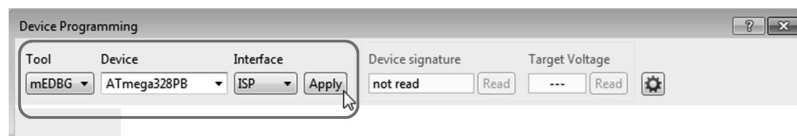
- ALTO: 0xDF
- BAJO: 0xC0
- EXT: 0xFC

Ingresa al cuadro de diálogo Programación del dispositivo como se muestra:



(/local--files/8avr:interrupts-mega-example/as7-program-fuses-1.png)

En el cuadro de diálogo Programación de dispositivos, seleccione la **herramienta** , **el dispositivo** y la **interfaz** como se muestra, luego presione **Aplicar** :



(/local--files/8avr:interrupts-mega-example/as7-program-fuses-2.png)

Para verificar una conexión, seleccione **Leer** y verifique que se encuentre una **firma de dispositivo** :



(/local--files/8avr:interrupts-mega-example/as7-program-fuses-3.png)

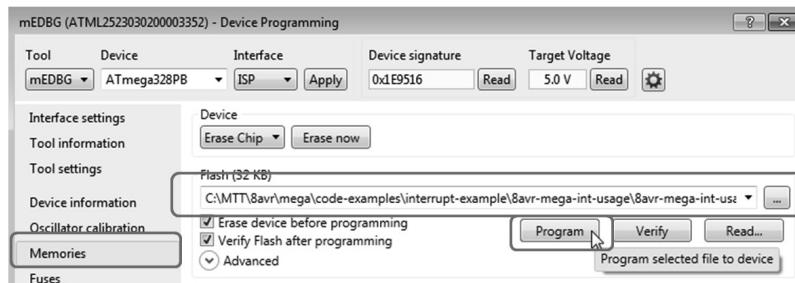
Seleccione la subsección **Fusibles** , ingrese los **3 valores de byte del fusible** arriba, luego presione **Programar** como se muestra:



(/local--files/8avr:interrupts-mega-example/as7-program-fuses-4.png)

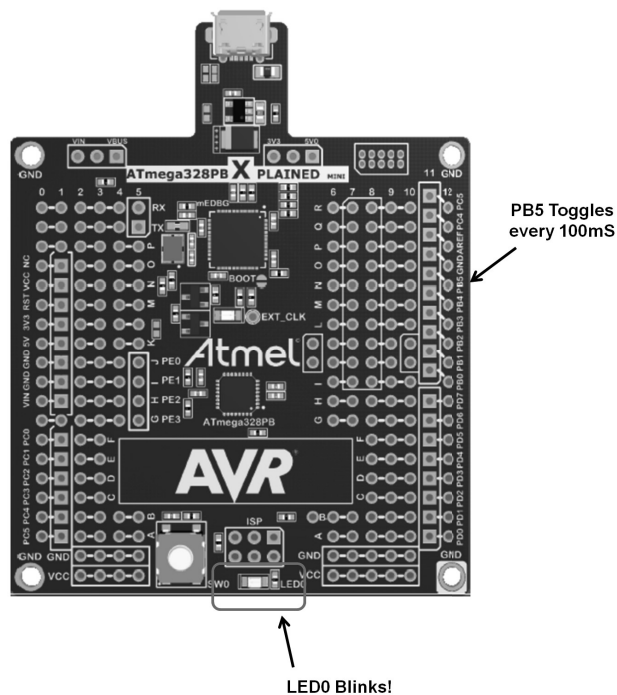
4 Programa el archivo hexadecimal

Mientras aún está en el cuadro de diálogo Programación del dispositivo, seleccione "Memorias" como se muestra. La ruta al archivo hexadecimal de la solución ya debería aparecer en el cuadro de diálogo. Presione Programa como se muestra:



(/local--files/8avr:interrupts-mega-example/as7-program-hex-1.png)

✶ Resultados



(/local--files/8avr:interrupts-mega-example/interrupts-mega-example-results.png)

💡 Conclusiones

Consideraciones Especiales

Esta página cubre algunas consideraciones especiales a tener en cuenta cuando se trabaja con interrupciones en MCU AVR.

Compartir datos con el ISR

Las variables compartidas entre el ISR y el programa principal deben declararse como **volátiles** y tener un alcance **global** . Al compilar usando el optimizador, en un bucle como el siguiente:



```
1  <font></font>
2  uint8_t flag;<font></font>
3  ...<font></font>
4  ISR(SOME_vect) {<font></font>
5      flag = 1;<font></font>
6  }<font></font>
7  ...<font></font>
8  while(flag == 0) {<font></font>
9      ...<font></font>
10 }<font></font>
11 <font></font>
```

el compilador normalmente accederá a "bandera" solo una vez y optimizará los accesos adicionales completamente, ya que su análisis de ruta de código muestra que nada dentro del ciclo podría cambiar el valor de "bandera" de todos modos. Para decirle al compilador que esta variable podría cambiarse fuera del alcance de su análisis de ruta de código (por ejemplo, dentro de una rutina de servicio de interrupción), la variable debe declararse así:




```
1  <font></font>
2  volatile uint8_t flag;<font>
3  ...<font></font>
4  ISR(SOME_vect) {<font></font>
5      flag = 1;<font></font>
6  }<font></font>
7  ...<font></font>
8  while(flag == 0) {<font></font>
9      ...<font></font>
10 }<font></font>
11 <font></font>
```

Cuando la variable se declara **volátil** como se indicó anteriormente, el compilador se asegura de que siempre que la variable se actualice o lea, siempre escribirá los cambios en la memoria SRAM y leerá la variable desde SRAM.

Operaciones de datos atómicos

Para que una operación sea considerada **atómica** , debe garantizar el acceso **ininterrumpido** de una determinada variable. Muchos lenguajes ensambladores brindan esto en ciertos niveles, es decir, prueba y configuración de bits, sin embargo, no existe **ninguna disposición** para proporcionar automáticamente la atomicidad de todos los tipos de variables en el lenguaje ANSI C.



Las expresiones y declaraciones ANSI-C **no son atómicas** .

Este problema puede ser problemático (en ciertas situaciones) cuando **las variables de varios bytes se comparten** con un ISR. Si bien declarar una variable de este tipo como **volátil** garantiza que el compilador no optimizará los accesos a ella, no garantiza el acceso **atómico** a ella. Considere el siguiente ejemplo de código:



```
1  <font></font>      ?
2  #include <stdint.h><font></font>
3  #include <avr/io.h><font></font>
4  #include <avr/interrupt.h><font></font>
5  <font></font>
6  volatile uint16_t ctr;<font></font>
7  <font></font>
8  ISR(TIMER1_OVF_vect)<font></font>
9  {<font></font>
10     ctr--;<font></font>
11 }<font></font>
12 ...<font></font>
13 int<font></font>
14 main(void)<font></font>
15 {<font></font>
dieciséis    ...<font></font>
17     ctr = 0x0200;<font></font>
18     start_timer();<font></font>
19     while(ctr != 0)<font></font>
20         // wait<font></font>
21         ;<font></font>
22     ...<font></font>
23 }<font></font>
24 <font></font>
```

Existe la posibilidad de que el contexto principal salga de su ciclo `while()` cuando la variable `ctr` alcance el valor `0x00FF`. Esto sucede porque el compilador no puede acceder de forma nativa a una variable de 16 bits de forma atómica en una CPU de 8 bits. Entonces, cuando `ctr` está, por ejemplo, en `0x0100`, el compilador luego prueba el byte bajo para 0, lo que tiene éxito. Luego procede a probar el byte alto, pero en ese momento se activa el ISR y el contexto principal se interrumpe. El ISR disminuirá la variable de `0x0100` a `0x00FF`, luego continúa el contexto principal. Ahora prueba el byte alto de la variable que (ahora) también es 0, por lo que concluye que la variable ha llegado a 0 y finaliza el ciclo.

Macros de acceso atómico

La biblioteca atómica (http://www.nongnu.org/avr-libc/user-manual/group__util__atomic.html) AVR-LIBC proporciona las macros `ATOMIC_BLOCK` que insertan la protección de interrupción adecuada cuando se desea acceso atómico. Estas macros operan a través de la manipulación automática del bit de **estado de interrupción global (I) del registro SREG**. Las rutas de salida de ambos tipos de bloques se gestionan automáticamente sin necesidad de consideraciones especiales, es decir, el estado de interrupción se restaurará al mismo valor que tenía al entrar en el bloque respectivo. Usando las macros de este archivo de encabezado, el código anterior se puede reescribir como:



```

1  <font></font>      ?
2  #include <stdint.h><font></font>
3  #include <avr/io.h><font></font>
4  #include <avr/interrupt.h><font></font>
5  #include <util/atomic.h><font></font>
6  <font></font>
7  volatile uint16_t ctr;
8  <font></font>
9  ISR(TIMER1_OVF_vect)<font></font>
10 {<font></font>
11     ctr--;<font></font>
12 }<font></font>
13 ...<font></font>
14 int main(void)<font></font>
15 {<font></font>
dieciséis     uint_16 ctr_copy;<font></font>
17     ...<font></font>
18     ctr = 0x0200;<font></font>
19     start_timer();<font></font>
20     do<font></font>
21     {<font></font>
22         ATOMIC_BLOCK(ATOMIC_RESTORESTATE)<font></font>
23         {<font></font>
24             ctr_copy = ctr;<font></font>
25         }<font></font>
26     } while(ctr != 0);
27     // wait<font></font>
28     ;<font></font>
29     ...<font></font>
30 }<font></font>
31 <font></font>

```

La macro **ATOMIC_BLOCK** instalará la protección de interrupción adecuada antes de acceder a la variable **ctr** , por lo que se garantiza que se probará de manera consistente. En este caso, el parámetro **ATOMIC_RESTORESTATE** hace que **ATOMIC_BLOCK** restaure el estado anterior del registro SREG, guardado antes de que se deshabilitara el bit indicador de estado de interrupción global. El efecto neto de esto es hacer que el contenido de **ATOMIC_BLOCK** sea atómico garantizado, sin cambiar el estado del indicador de estado de interrupción global cuando se completa la ejecución del bloque.

Aprende más



Resumen de interrupciones de megaAVR®

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



Configuración de interrupciones megaAVR®