Córdoba, 03 de Agosto del 2022

Carrera: **Telecomunicaciones** **Materia:** Electrónica microcontrolada **Grupo:** N°3
Docentes: Jorge Morales – Gonzalo Vera
Alumnos: Carolina Nis - Fernando Vexenat - Rodolfo Paz – Andres Montaño - Juan Diego González Antoniazzi - Leonardo González

Periféricos AVR de 8 bits

Descripción general del oscilador megaAVR

Los microcontroladores de 8 bits megaAVR® de Microchip tienen varias opciones de fuente de reloj, seleccionables a través de la programación de los bits de fusible **CKSEL** **Flash** . Esta discusión es específica de la MCU ATmega328PB . Los bits de fusible pueden seleccionar uno de:

- Oscilador de cristal de baja potencia
- Oscilador de cristal de baja frecuencia
- Oscilador RC interno de 128 kHz
- Oscilador RC interno calibrado, y
- Reloj externo.

La ***fuente del reloj del sistema*** **no se** puede cambiar durante el tiempo de ejecución, ya que se configura a través de la programación de fusibles.

La ***frecuencia del reloj del sistema*** **se** puede cambiar durante el tiempo de ejecución escribiendo en el registro del preescalador del **reloj del sistema** (CLKPR).

Cada fuente de reloj proporciona una opción de retraso después del reinicio o encendido del dispositivo para mantener el dispositivo reiniciado hasta que se suministre con Vcc mínimo. El reloj de la fuente seleccionada se ingresa al generador de reloj AVR® y se enruta a los módulos apropiados.

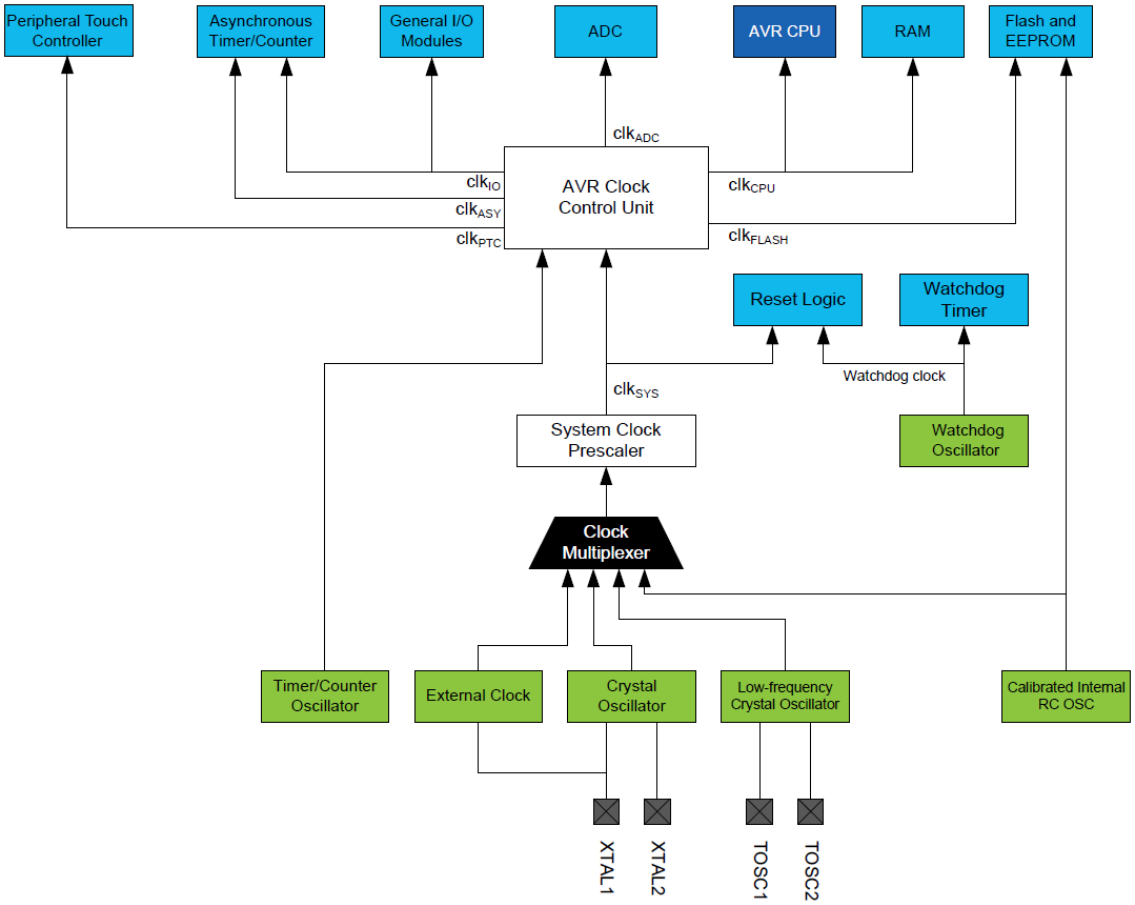
La frecuencia operativa máxima del megaAVR® depende de V $_{cc}$. El software de la aplicación debe garantizar que la frecuencia de la fuente de reloj seleccionada se encuentre dentro del área de operación segura (consulte la sección 33.4 en la **hoja de datos del dispositivo**).

Visión general

La siguiente figura ilustra los principales sistemas de reloj del dispositivo y su distribución. No es necesario que todos los relojes estén activos en un momento dado. Para reducir el consumo de energía, los relojes de los módulos que no se utilizan se pueden detener utilizando diferentes modos de suspensión . Los sistemas de reloj se describen en las siguientes secciones. La frecuencia del reloj del sistema se refiere a la frecuencia generada por el preescalador del reloj del sistema. Todas las salidas de reloj de la unidad de control de reloj AVR funcionan a la misma frecuencia.

Córdoba, 03 de Agosto del 2022

Carrera: **Telecomunicaciones** **Materia:** Electrónica microcontrolada **Grupo:** N°3
Docentes: Jorge Morales – Gonzalo Vera
Alumnos: Carolina Nis - Fernando Vexenat - Rodolfo Paz – Andres Montaño - Juan Diego González Antoniazzi - Leonardo González



Fuentes de reloj

El dispositivo tiene las siguientes opciones de fuente de reloj, seleccionables a través de los bits **CKSEL** Flash Fuse como se muestra a continuación. El reloj de la fuente seleccionada se ingresa al generador de reloj AVR® y se enruta a los módulos apropiados.

Device Clocking Option	CKSEL[3:0]
Low Power Crystal Oscillator	1111 - 1000
Low Frequency Crystal Oscillator	0101 - 0100
Internal 128kHz RC Oscillator	0011
Calibrated Internal RC Oscillator	0010
External Clock	0000
Reserved	0001

Fuente de reloj predeterminada

El dispositivo se envía con el oscilador RC interno seleccionado a 8,0 MHz y con el fusible CKDIV8 programado, lo que da como resultado un reloj del sistema de 1,0 MHz. El tiempo de inicio se establece al

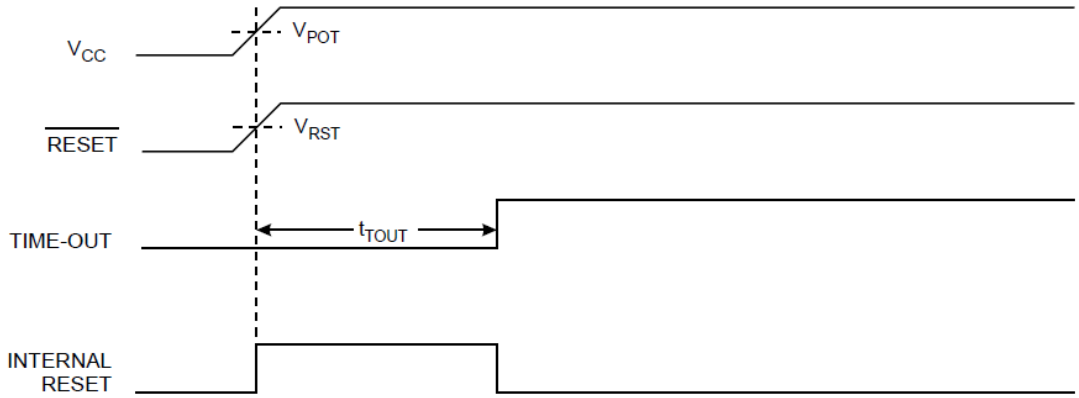
máximo y el período de tiempo de espera está habilitado: CKSEL=0010, SUT=10, CKDIV8=0. Esta configuración predeterminada garantiza que todos los usuarios puedan configurar la fuente de reloj deseada utilizando cualquier interfaz de programación disponible.

Secuencia de inicio del reloj

Cualquier fuente de reloj necesita (i) **un V_{CC} suficiente para comenzar a oscilar** y (ii) **un número mínimo de ciclos de oscilación antes de que pueda considerarse estable** .

Estabilidad Vcc

Para garantizar suficiente V_{CC} , el dispositivo emite un restablecimiento interno con un retraso de tiempo de espera (t_{TOUT}) después de que todas las demás fuentes de restablecimiento liberan el restablecimiento del dispositivo:



El retardo (t_{TOUT}) se cronometra desde el oscilador de vigilancia y el tiempo de retardo se establece mediante los bits de fusible SUTx y CKSELx. Los retardos seleccionables para t_{TOUT} se muestran en la siguiente tabla. Tenga en cuenta que la frecuencia del Watchdog Oscillator depende del voltaje:

Typ. Time-out (V_{CC} = 5.0V)	Typ. Time-out (V_{CC} = 3.0V)
0ms	0ms
4ms	4.3ms
65ms	69ms

V_{CC} no se controla durante el retraso, por lo que se requiere seleccionar un retraso más largo que el tiempo de subida de V_{CC} . Si esto no es posible, se debe utilizar un circuito de detección de Brown-Out (BOD) interno o externo. Un circuito BOD garantizará suficiente V_{CC} antes de liberar el reinicio, y el retardo de tiempo de espera se puede desactivar. No se recomienda deshabilitar el retardo de tiempo de espera sin utilizar un circuito de detección de Brown-Out.

El Fusible CKSEL0 junto con los Fusibles SUT[1:0] seleccionan los tiempos de arranque (ver sección 11.3 en la **hoja de datos del dispositivo**).

Córdoba, 03 de Agosto del 2022

Carrera: **Telecomunicaciones** **Materia:** Electrónica microcontrolada **Grupo:** N°3
Docentes: Jorge Morales – Gonzalo Vera
Alumnos: Carolina Nis - Fernando Vexenat - Rodolfo Paz – Andres Montaño - Juan Diego González Antoniazzi - Leonardo González

Oscilador de cristal de baja frecuencia

El oscilador de cristal de baja frecuencia está optimizado para su uso con un cristal de reloj de 32,768 kHz. El oscilador de cristal de baja frecuencia debe seleccionarse configurando los fusibles CKSEL en '0110' o '0111', y los tiempos de inicio están determinados por el SUT fusibles

Oscilador RC interno calibrado

De forma predeterminada, el oscilador RC interno proporciona un reloj de 8,0 MHz. Aunque depende del voltaje y la temperatura, el usuario puede calibrar este reloj con mucha precisión. El dispositivo se envía con el fusible CKDIV8 programado, lo que proporciona una frecuencia de reloj del sistema de 1 MHz. Este reloj se puede seleccionar como el reloj del sistema programando los fusibles CKSEL en '0010':. Si se selecciona, funcionará sin componentes externos. Durante el reinicio, el hardware carga el valor de calibración preprogramado en el registro OSCCAL y, por lo tanto, calibra automáticamente el oscilador RC.

Consulte la **nota de aplicación AVR053** que describe el procedimiento para volver a calibrar el oscilador RC interno.

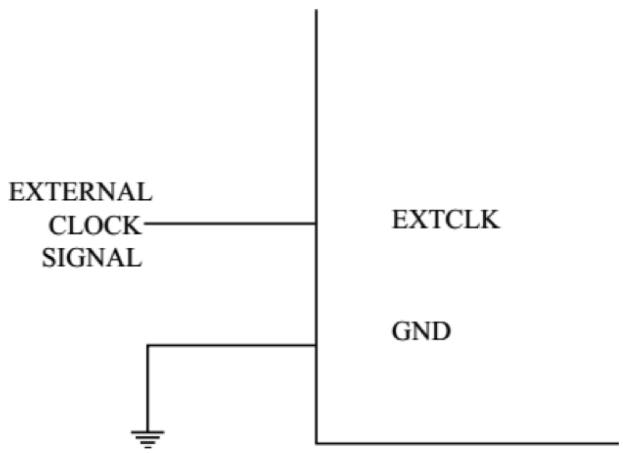
Oscilador interno de 128 kHz

El oscilador interno de 128 kHz es un oscilador de baja potencia que proporciona un reloj de 128 kHz. Este reloj se puede seleccionar como reloj del sistema programando los fusibles CKSEL en '0011'.

Reloj externo

Para controlar el dispositivo desde una fuente de reloj externa, EXTCLK debe controlarse como se muestra en la figura a continuación. Para ejecutar el dispositivo en un reloj externo, los fusibles CKSEL deben programarse en '0000'.

External Clock Drive Configuration



Búfer de salida de reloj

El dispositivo puede generar el reloj del sistema en el pin CLKO. Para habilitar la salida, se debe programar el Fusible CKOUT. Este modo es adecuado cuando el reloj del chip se usa para controlar otros circuitos en el sistema. El reloj también se emitirá durante el reinicio, y el funcionamiento normal del pin de E/S se anulará cuando se programe el fusible. Cualquier fuente de reloj, incluido el oscilador RC interno, se puede seleccionar cuando el reloj se emite en CLKO. Si se utiliza el preescalador de reloj del sistema, lo que se emite es el reloj del sistema dividido.

Temporizador/Contador Oscilador

El dispositivo utiliza el mismo oscilador de cristal para el oscilador de baja frecuencia y el temporizador/contador de osciladores. Consulte Oscilador de cristal de baja frecuencia para obtener detalles sobre los requisitos del oscilador y del cristal.

En este dispositivo, los pines del oscilador del temporizador/contador (TOSC1 y TOSC2) se comparten con EXTCLK. Cuando se utiliza el temporizador/contador de osciladores, el reloj del sistema debe ser cuatro veces la frecuencia del oscilador. Debido a esto y al uso compartido de pines, el temporizador/contador de osciladores solo se puede usar cuando el oscilador RC interno calibrado se selecciona como fuente de reloj del sistema. Se puede aplicar una fuente de reloj externa a TOSC1 si el bit Habilitar entrada de reloj externo en el registro de estado asíncrono (ASSR.EXCLK) se escribe en '1'. Consulte la descripción de la operación asíncrona del temporizador/contador2 para obtener una descripción más detallada sobre la selección de un reloj externo como entrada en lugar de un cristal de reloj de 32,768 kHz.

Prescaler del reloj del sistema

El dispositivo tiene un preescalador de reloj del sistema, y el reloj del sistema se puede dividir configurando el Registro de preescala de reloj (CLKPR). Esta función se puede utilizar para disminuir la frecuencia del reloj del sistema y el consumo de energía cuando el requisito de potencia de procesamiento es bajo. Esto se puede usar con todas las opciones de fuente de reloj y afectará la frecuencia de reloj de la CPU y todos los periféricos síncronos. $clk_{I/O}$, clk_{ADC} , clk_{CPU} y clk_{FLASH} se dividen por un factor como se muestra en la descripción de CLKPR:

Name: CLKPR
Offset: 0x61
Reset: Refer to the bit description
Property: -

Bit	7	6	5	4	3	2	1	0
	CLKPCE				CLKPSn	CLKPSn	CLKPSn	CLKPSn
Access	R/W				R/W	R/W	R/W	R/W
Reset	0				x	x	x	x

Bit 7 – CLKPCE: Clock Prescaler Change Enable
The CLKPCE bit must be written to logic one to enable change of the CLKPS bits. The CLKPCE bit is only updated when the other bits in CLKPR are simultaneously written to zero. CLKPCE is cleared by hardware four cycles after it is written or when CLKPS bits are written. Rewriting the CLKPCE bit within this time-out period does neither extend the time-out period, nor clear the CLKPCE bit.

Bits 3:0 – CLKPSn: Clock Prescaler Select n [n = 3:0]
These bits define the division factor between the selected clock source and the internal system clock. These bits can be written run-time to vary the clock frequency to suit the application requirements. As the divider divides the master clock input to the MCU, the speed of all synchronous peripherals is reduced when a division factor is used. The division factors are given in the table below.

CLKPS[3:0]	Clock Division Factor
0000	1
0001	2
0010	4
0011	8
0100	16
0101	32
0110	64
0111	128
1000	256
1001	Reserved
1010	Reserved
1011	Reserved
1100	Reserved
1101	Reserved
1110	Reserved
1111	Reserved

Escribiendo a CLKPR

Al cambiar entre las configuraciones del preescalador, el Preescalador del reloj del sistema asegura que no ocurran fallas en el sistema del reloj. También asegura que ninguna frecuencia intermedia sea superior a la frecuencia de reloj correspondiente a la configuración anterior, ni a la frecuencia de reloj correspondiente a la nueva configuración. El contador de ondas que implementa el preescalador se ejecuta a la frecuencia del reloj indiviso, que puede ser más rápido que la frecuencia del reloj de la CPU. Por lo tanto, no es posible determinar el estado del preescalador; incluso si fuera legible, el tiempo exacto que se tarda en cambiar de una división de reloj a otra no se puede predecir con exactitud. Desde el momento en que se escriben los valores de los bits de selección del preescalador de reloj (CLKPS[3:0]), transcurren entre T1 + T2 y T1 + 2 * T2 antes de que se active la nueva frecuencia de reloj. En este intervalo, se producen dos flancos de reloj activos. Aquí, T1 es el período de reloj anterior y T2 es el período correspondiente a la nueva configuración del preescalador. Para evitar cambios involuntarios de la frecuencia del reloj, se debe seguir un procedimiento de escritura especial para cambiar los bits CLKPS:

1. Escriba el bit de habilitación de cambio de preescalador de reloj (CLKPCE) en '1' y todos los demás bits en CLKPR en cero: CLKPR=0x80.
2. Dentro de cuatro ciclos, escriba el valor deseado en CLKPS[3:0] mientras escribe un cero en CLKPCE: CLKPR=0x0N

Las interrupciones deben desactivarse al cambiar la configuración del preescalador para asegurarse de que el procedimiento de escritura no se interrumpa.

Ejemplo de código

La siguiente función se puede utilizar para actualizar dinámicamente CLKPR como se requiere anteriormente. Tenga en cuenta el uso de las funciones `cli()` y `sei()` para garantizar que el procedimiento de escritura CLKPR no se interrumpa.

```
#include <stdint.h>  // Std integral type definitions
#include <avr/io.h>  // SFR definitions
#include <avr/interrupt.h> // ISR macros

void clkPrescaleSet(uint8_t divisionFactor){
    cli();                          // disable interrupts
    CLKPR = (1 << CLKPCE);          // enable change of the CLKPSx bits
    CLKPR = divisionFactor;         // update the CLKPSx bits
    sei();                          // re-enable interrupts
}
```

Para ver esta función en uso, visite el proyecto de ejemplo del **oscilador megaAVR®**

Córdoba, 03 de Agosto del 2022

Carrera: **Telecomunicaciones** **Materia:** Electrónica microcontrolada **Grupo:** N°3
Docentes: Jorge Morales – Gonzalo Vera
Alumnos: Carolina Nis - Fernando Vexenat - Rodolfo Paz – Andres Montaño - Juan Diego González Antoniazzi - Leonardo González

CLKDIV8 Fusible y CLKPR

El fusible CKDIV8 determina el valor inicial de los bits CLKPS. Si CKDIV8 no está programado, los bits CLKPS se restablecerán a "0000". Si se programa CKDIV8, los bits CLKPS se restablecen a "0011", dando un factor de división de 8 al inicio. Esta función debe utilizarse si la fuente de reloj seleccionada tiene una frecuencia superior a la frecuencia máxima del dispositivo en las condiciones de funcionamiento actuales. Tenga en cuenta que se puede escribir cualquier valor en los bits CLKPS independientemente de la configuración del fusible CKDIV8. El software de la aplicación debe garantizar que se elija un factor de división suficiente si la fuente de reloj seleccionada tiene una frecuencia superior a la frecuencia máxima del dispositivo en las condiciones de funcionamiento actuales. El dispositivo se envía con el fusible CKDIV8 programado.

Descripción general de AVR® USART

Los microcontroladores Microchip AVR® de 8 bits contienen un periférico de comunicación altamente flexible conocido como **USART** (receptor y transmisor serie universal síncrono y asíncrono). Este periférico se puede utilizar para comunicarse con una amplia variedad de otros componentes, incluidos otros microcontroladores, módulos inalámbricos, pantallas LCD, módulos GPS, etc. El periférico USART puede funcionar en uno de dos modos principales: sincrónico o asincrónico. Este módulo se centra en el **modo de funcionamiento asíncrono** .

Configuración megaAVR® USART

En esta sección, cubriremos los pasos básicos de codificación necesarios para configurar/usar el módulo USART en un MCU megaAVR®, con un enfoque en el dispositivo **ATmega328PB .**

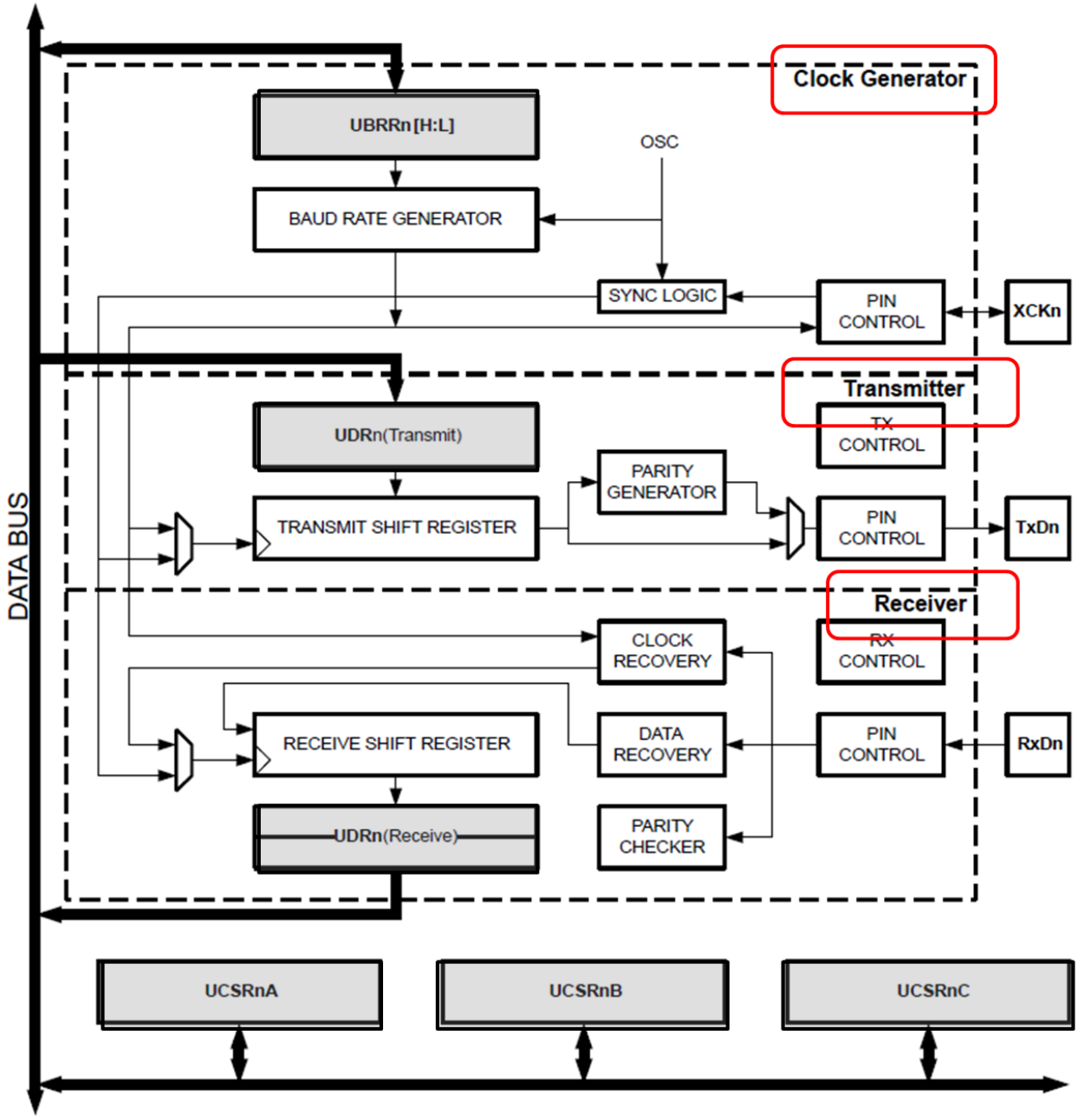
Visión general

El módulo USART consta de tres secciones principales, como se muestra en el siguiente diagrama: **generador de reloj** , **transmisor** y **receptor** .

Córdoba, 03 de Agosto del 2022

Carrera: **Telecomunicaciones** **Materia:** Electrónica microcontrolada **Grupo:** N°3
Docentes: Jorge Morales – Gonzalo Vera
Alumnos: Carolina Nis - Fernando Vexenat - Rodolfo Paz – Andres Montaño - Juan Diego González Antoniazzi - Leonardo González

Figure 24-1. USART Block Diagram



Los registros clave (resaltados en gris) incluyen:

- Registros de control y estado (**UCSRnA** , **UCSRnB** , **UCSRnC**) compartidos por las tres secciones.
- Registro de datos **UDRn** compartido por las secciones Transmisor y Receptor.
- Registros de control de velocidad en baudios **UBRRn[H:L]** utilizados por el generador de reloj.

" **n** " en el nombre del registro/bit identifica la instancia de hardware USART específica (0, 1, 2) a la que está asociado el registro/bit. Por ejemplo **, UCSR0A** se refiere a **USART0** Control & Status Register **A**

Córdoba, 03 de Agosto del 2022

Carrera: Telecomunicaciones **Materia:** Electrónica microcontrolada **Grupo:** N°3
Docentes: Jorge Morales – Gonzalo Vera
Alumnos: Carolina Nis - Fernando Vexenat - Rodolfo Paz – Andres Montaño - Juan Diego González Antoniazzi - Leonardo González

Usando el USART (Resumen)

Para la operación sondeada básica, se deben realizar los siguientes pasos mínimos:

1. Elija una tasa de baudios y programe los **registros UBRRn[H:L]** en consecuencia.
2. Habilite las secciones de transmisión y recepción en serie de USART.
3. Si está transmitiendo, espere hasta que el registro de desplazamiento de transmisión esté vacío (sondee en **UCSRnA.UDREn**), luego cargue su byte de datos en **UDRn** .
4. Si recibe, espere hasta que se establezca el bit de recepción de datos del receptor (sondee en **UCSRnA.RXCn**), luego lea los datos de **UDRn** . La lectura de UDRn borra automáticamente el bit y prepara el hardware para recibir el siguiente byte.

Inicialización

El USART debe inicializarse antes de que pueda tener lugar cualquier comunicación. El proceso de inicialización normalmente consiste en:

- Configuración de la velocidad en baudios,
- Configuración del formato de cuadro y
- Habilitación del Transmisor o del Receptor según el uso.

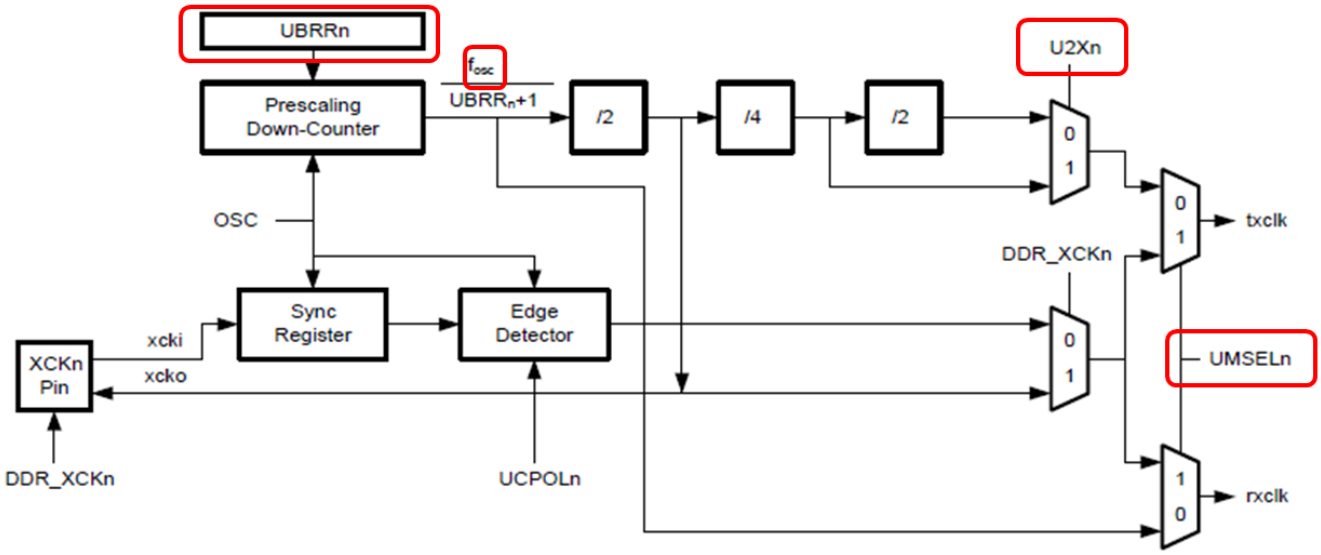
Configuración de la tasa de baudios

La generación de reloj interno se utiliza para el modo de funcionamiento asíncrono. La lógica de generación de reloj genera el reloj base para el transmisor y el receptor (los registros clave y los bits de control están resaltados):

Córdoba, 03 de Agosto del 2022

Carrera: **Telecomunicaciones** **Materia:** Electrónica microcontrolada **Grupo:** N°3
Docentes: Jorge Morales – Gonzalo Vera
Alumnos: Carolina Nis - Fernando Vexenat - Rodolfo Paz – Andres Montaño - Juan Diego González Antoniazzi - Leonardo González

Figure 24-2. Clock Generation Logic, Block Diagram



Signal description:

- txclk: Transmitter clock (internal signal).
- rxclk: Receiver base clock (internal signal).
- xcki: Input from XCKn pin (internal signal). Used for synchronous slave operation.
- xcko: Clock output to XCKn pin (internal signal). Used for synchronous master operation.
- f_{osc}: System clock frequency.

Selección de modo USART (UMSELn)

La ecuación de velocidad en baudios utilizada por el módulo se establece en función del modo de funcionamiento. Para la operación en modo asíncrono, los bits de selección de modo USART en el registro C de control y estado de USART (**UCSRnC.UMSELn[1:0]**) se utilizan para seleccionar **la operación asíncrona** (**UMSEL[1:0] = 00**) como se muestra:

Name: UCSR0C, UCSR1C
Offset: 0xC2 + n*0x08 [n=0..1]
Reset: 0x06
Property: -

Bit	7	6	5	4	3	2	1	0
	UMSEL[1:0]		UPM[1:0]		USBS	UCSZ1 / UDORD	UCSZ0 / UCPHA	UCPOL
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	1	1	0

Con el modo de velocidad doble configurado, el receptor solo usará la mitad del número de muestras (reducidas de 16 a 8) para el muestreo de datos y la recuperación del reloj y, por lo tanto, se requiere una configuración de velocidad en baudios y un reloj del sistema más precisos cuando se usa este modo.

Registro de velocidad en baudios (UBRRn)

El registro de tasa de baudios USART (**UBRRn**) y el contador descendente conectado a él funcionan como un preescalador programable o generador de tasa de baudios. El contador regresivo, que se ejecuta en el reloj del sistema (f_{osc}), se carga con el valor UBRRn cada vez que el contador llega a cero o cuando se escribe el registro UBRRnL. Se genera un reloj cada vez que el contador llega a cero. Este reloj es la salida del reloj del generador de velocidad en baudios ($= f_{osc}/(UBRRn+1)$). El transmisor divide la salida del reloj del generador de velocidad en baudios por 2, 8 o 16 según el modo. La salida del generador de velocidad en baudios es utilizada directamente por el reloj del receptor y las unidades de recuperación de datos. Sin embargo, las unidades de recuperación usan una máquina de estado que usa 2, 8 o 16 estados según el modo establecido por el estado de los bits UMSEL, U2Xn y DDR_XCK. La siguiente tabla contiene ecuaciones para calcular la tasa de baudios (en bits por segundo) y para calcular el valor UBRRn para cada modo de operación utilizando una fuente de reloj generada internamente.

Table 24-1. Equations for Calculating Baud Rate Register Setting

Operating Mode	Equation for Calculating Baud Rate(1)	Equation for Calculating UBRRn Value
Asynchronous Normal mode (U2Xn = 0)	$\text{BAUD} = \frac{f_{\text{osc}}}{16(\mathbf{UBRR}n + 1)}$	$\mathbf{UBRR}n = \frac{f_{\text{osc}}}{16\text{BAUD}} - 1$
Asynchronous Double Speed mode (U2Xn = 1)	$\text{BAUD} = \frac{f_{\text{osc}}}{8(\mathbf{UBRR}n + 1)}$	$\mathbf{UBRR}n = \frac{f_{\text{osc}}}{8\text{BAUD}} - 1$

La biblioteca **AVR-LIBC Setbaud** contiene macros útiles para calcular los valores correctos para escribir en los registros UBRRnH y UBRRnL. Consulte el ejemplo de código de inicialización a continuación.

También se proporcionan tablas en la hoja de datos del dispositivo que contienen valores UBRRn para tasas de baudios comunes, dadas varias frecuencias de oscilador:

Table 24-9. Examples of UBRRn Settings for Commonly Used Oscillator Frequencies

Baud Rate [bps]	f_{osc} = 16.0000MHz				f_{osc} = 18.4320MHz				f_{osc} = 20.0000MHz			
	U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1		U2Xn = 0		U2Xn = 1	
	UBRRn	Error	UBRRn	Error	UBRRn	Error	UBRRn	Error	UBRRn	Error	UBRRn	Error
2400	416	-0.1%	832	0.0%	479	0.0%	959	0.0%	520	0.0%	1041	0.0%
4800	207	0.2%	416	-0.1%	239	0.0%	479	0.0%	259	0.2%	520	0.0%
9600	103	0.2%	207	0.2%	119	0.0%	239	0.0%	129	0.2%	259	0.2%
14.4k	68	0.6%	138	-0.1%	79	0.0%	159	0.0%	86	-0.2%	173	-0.2%
19.2k	51	0.2%	103	0.2%	59	0.0%	119	0.0%	64	0.2%	129	0.2%
28.8k	34	-0.8%	68	0.6%	39	0.0%	79	0.0%	42	0.9%	86	-0.2%
38.4k	25	0.2%	51	0.2%	29	0.0%	59	0.0%	32	-1.4%	64	0.2%
57.6k	16	2.1%	34	-0.8%	19	0.0%	39	0.0%	21	-1.4%	42	0.9%
76.8k	12	0.2%	25	0.2%	14	0.0%	29	0.0%	15	1.7%	32	-1.4%
115.2k	8	-3.5%	16	2.1%	9	0.0%	19	0.0%	10	-1.4%	21	-1.4%

Para los cálculos de frecuencia en baudios, generalmente se acepta que los porcentajes de error de menos de ± 2% son aceptables.

Configuración del formato de marco

El registro C de control y estado de USART (**UCSRnC**) se utiliza para configurar el formato de la trama de comunicación UART: paridad, número de bits de parada y número de bits de datos. Los ajustes para el formato de cuadro típico “8N1” son los siguientes:

- **UPM[1:0] = 00** para Sin paridad
- **USBS = 0** para 1 bit de parada
- **UCSZ1[1:0]** = 11 para 8 Bits

Name: UCSR0C, UCSR1C
Offset: 0xC2 + n*0x08 [n=0..1]
Reset: 0x06
Property: -

Bit	7	6	5	4	3	2	1	0
	UMSEL[1:0]		UPM[1:0]		USBS	UCSZ1 / UDORD	UCSZ0 / UCPHA	UCPOL
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	1	1	0

Habilitación del transmisor

El transmisor USART se habilita configurando el bit de **habilitación de transmisión (TXEN)** en el registro **UCSRnB** :

Name: UCSR0B, UCSR1B
Offset: 0xC1 + n*0x08 [n=0..1]
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8
Access	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
Reset	0	0	0	0	0	0	0	0

Cuando el transmisor está habilitado, la operación normal del puerto del pin TxDn es anulada por el USART y se le asigna la función de salida en serie del transmisor.

La velocidad en baudios, el modo de operación y el formato de trama deben configurarse una vez antes de realizar cualquier transmisión.

Habilitación del receptor

El receptor USART se habilita escribiendo el bit de **habilitación de recepción (RXEN)** en el registro **UCSRnB** a '1':

Name: UCSR0B, UCSR1B
Offset: 0xC1 + n*0x08 [n=0..1]
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8
Access	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
Reset	0	0	0	0	0	0	0	0

Cuando el Receptor está habilitado, el USART anula la operación normal del puerto del pin RxDn y se le asigna la función como entrada en serie del Receptor.

La velocidad en baudios, el modo de operación y el formato de trama deben configurarse una vez antes de realizar cualquier transmisión.

Ejemplo de código

El siguiente ejemplo de código de inicialización de USART utiliza la biblioteca de **utilidades setbaud** en AVR-LIBC. Esta biblioteca proporciona macros que usan el preprocesador c para calcular los valores apropiados para **UBBRn** . **Entradas** Este archivo de encabezado requiere que los valores de entrada ya estén definidos para **F_CPU** y **BAUD** . Además, la macro BAUD_TOL definirá la tolerancia de velocidad en baudios (en porcentaje) que es aceptable durante los cálculos. El valor de BAUD_TOL por defecto será +/-

2%. **Salidas** Suponiendo que los BAUD solicitados son válidos para la F_CPU dada, entonces la macro **UBRR_VALUE** se establece en el valor del preescalador requerido. Se proporcionan dos macros adicionales para los bytes alto y bajo del preescalador, respectivamente: **UBRRL_VALUE** se establece en el byte inferior de UBRR_VALUE y **UBRRH_VALUE** se establece en el byte superior. Se definirá una macro adicional **USE_2X** . Su valor se establece en 1 si la tasa de BAUDIOS deseada dentro de la tolerancia dada solo se puede lograr al establecer el bit **U2Xn** en la configuración de UART. Se definirá a 0 si no se necesita **U2Xn** .

```
1<font></font>
2#define F_CPU 16000000UL            // required for setbaud & other libraries<fo
3#define BAUD 38400UL                // desired baud<font></font>
4#define BAUD_TOL 2                  // desired baud rate tolerance (+/- %)<font>
5<font></font>
6#include <avr/io.h><font></font>
7#include <util/setbaud.h><font></font>
8<font></font>
9void USART0_Init(void){<font></font>
10<font></font>
11    // Set the BAUD rate<font></font>
12<font></font>
13    UBRR0H = UBRRH_VALUE;<font></font>
14    UBRR0L = UBRRL_VALUE;<font></font>
15    #if USE_2X                        // USE_2X defined by setbaud.h based on inp
dieciséis    UCSR0A |= (1 << U2X0);<font></font>
17    #else<font></font>
18    UCSR0A &= ~(1 << U2X0);<font></font>
19    #endif<font></font>
20<font></font>
21    // Set the Mode & Frame Parameters<font></font>
22<font></font>
23    UCSR0C = 0x06;                    // Asynchronous, 8-data, No parity, 1-stop<
24<font></font>
25    // Enable USART0 Transmitter and Receiver<font></font>
26<font></font>
27    UCSR0B = (1 << TXEN0) | (1 << RXEN0);<font></font>
28<font></font>
29}<font></font>
30<font></font>
```

La biblioteca setbaud genera mensajes de advertencia durante la compilación si los parámetros de entrada generan una configuración de tasa BAUD que producirá una tasa de baudios fuera del BAUD_TOL deseado

Córdoba, 03 de Agosto del 2022

Carrera: Telecomunicaciones **Materia:** Electrónica microcontrolada **Grupo:** N°3
Docentes: Jorge Morales – Gonzalo Vera
Alumnos: Carolina Nis - Fernando Vexenat - Rodolfo Paz – Andres Montaño - Juan Diego González Antoniazzi - Leonardo González

Transmisión de datos

Transmitir

Una transmisión de datos se inicia cargando el búfer de transmisión con los datos a transmitir. La CPU puede cargar el búfer de transmisión escribiendo en el registro **UDRn** . Para la operación de sondeo, el firmware debe monitorear el indicador de registro de datos vacío (**UCSRnA.UDREn**) antes de cargar **UDRn** . Los datos almacenados en el búfer de transmisión se moverán al registro de desplazamiento cuando el registro de desplazamiento esté listo para enviar una nueva trama. El registro de desplazamiento se carga con nuevos datos si está en estado inactivo (sin transmisión en curso) o inmediatamente después de que se transmita el último bit de parada de la trama anterior. Cuando el registro de desplazamiento se carga con nuevos datos, transferirá un cuadro completo a la velocidad dada por el registro de baudios.

Name: UDR
Offset: 0xC6 + n*0x08 [n=0..1]
Reset: 0x00
Property: -

Bit	7	6	5	4	3	2	1	0
	TXB / RXB[7:0]							
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

El indicador de interrupción de transmisión completa (**USCRnA.TXCn**) se establece y se puede generar una interrupción de TX opcional (si está habilitada) cuando se ha desplazado todo el marco en el registro de desplazamiento. El bit indicador **UCSRnA.TXCn** se borra automáticamente cuando se ejecuta una interrupción de transmisión completa, o se puede borrar escribiendo un uno en su ubicación de bit.

Recibir

El receptor inicia la recepción de datos cuando detecta un bit de inicio válido. Cada bit que sigue al bit de inicio se muestreará a la velocidad en baudios o al reloj XCKn, y se desplazará al registro de desplazamiento de recepción hasta que se reciba el primer bit de parada de una trama. El búfer de recepción se puede leer leyendo el registro **UDRn** . La recepción completa de un byte se puede verificar sondeando el bit RXCn en **el registro UCSRnA** .

El indicador de interrupción de recepción completa (**RXCn**) se establece y se puede generar una interrupción de RX opcional (si está habilitada) cuando el cuadro completo en el registro de desplazamiento se ha copiado en el registro **UDRn** . Esta es una interrupción *persistente* , es decir, el firmware debe leer los datos recibidos de **UDRn** para borrar el indicador **RXCn**

Ejemplo de código

Las siguientes API de bloqueo simple envían y reciben un byte de datos a través de USART0.

```
1<font></font>
2void USART0_Transmit(unsigned char data){<font></font>
3<font></font>
4    // Wait for empty transmit buffer<font></font>
5    while(!(UCSR0A & (1 << UDRE0)));<font></font>
6<font></font>
7    // Put data into buffer, sends the data<font></font>
8    UDR0 = data;<font></font>
9<font></font>
10}<font></font>
11<font></font>
12unsigned char USART0_Receive(void){<font></font>
13<font></font>
14    // Wait for data to be received<font></font>
15    while(!(UCSR0A & (1 << RXC0)));<font></font>
dieciséis<font></font>
17    // Get and return received data from buffer<font></font>
18    return UDR0;<font></font>
19<font></font>
20}<font></font>
21<font></font>
```

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

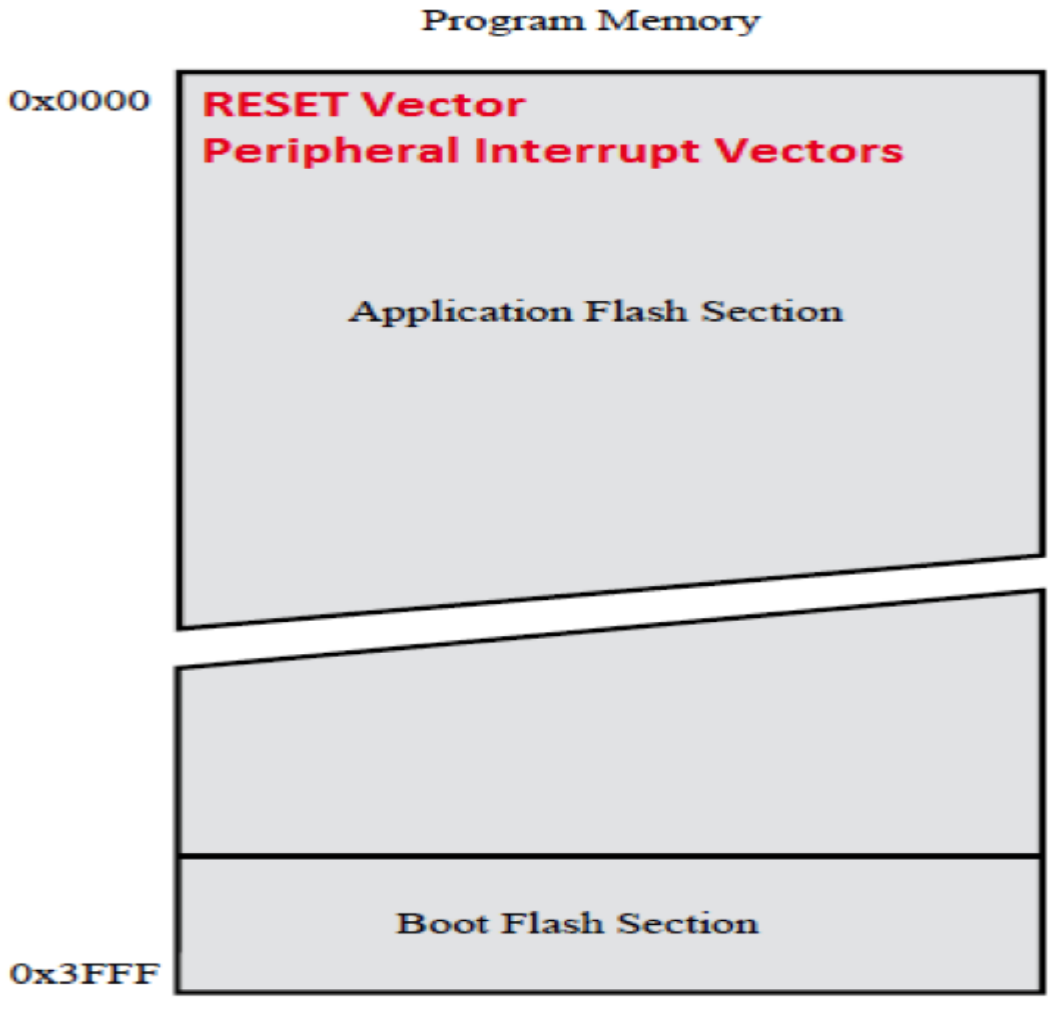
A los periféricos se les asignan **bits de habilitación de interrupción** individuales en su respectivo **registro de máscara de interrupción** que debe escribirse como uno lógico junto con el **bit I de habilitación de interrupción global** en el **registro** de estado para habilitar la interrupción.

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

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

Los fusibles se programan utilizando un **procedimiento de programación especial** 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:

```
void move_interrupts(void)
{
  uchar temp;
  /* GET MCUCR */
  temp = MCUCR;
  /* Enable change of Interrupt Vectors */
  MCUCR = temp | (1 << IVCE);
  /* Move interrupts to Boot Flash section */
  MCUCR = temp | (1 << IVSEL);
}
```

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/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

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.

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 AVR-LIBC** .

Córdoba, 03 de Agosto del 2022

Carrera: **Telecomunicaciones** **Materia:** Electrónica microcontrolada **Grupo:** N°3
Docentes: Jorge Morales – Gonzalo Vera
Alumnos: Carolina Nis - Fernando Vexenat - Rodolfo Paz – Andres Montaño - Juan Diego González Antoniazzi - Leonardo González

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:

- `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:

```
1<font></font>
2ISR(vector_id, ISR_[BLOCK|NOBLOCK|NAKED|ALIASOF])<font></font>
3{<font></font>
4    /* Hardware auto-clears the interrupt flag (most interrupt sources) */<font></font>
5    /* Clear the cause of the interrupt (required by some interrupt sources) */<font></f
6    /* ISR-specific processing */<font></font>
7}        <font></font>
8<font></font>
```

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

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<font></font>
2ISR(PCINT0_vect)<font></font>
3{<font></font>
4  ...<font></font>
5  // Code to handle the event.<font></font>
6}<font></font>
7ISR(PCINT1_vect, ISR_ALIASOF(PCINT0_vect));<font></font>
8<font></font>
```

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** cada 100 mS:

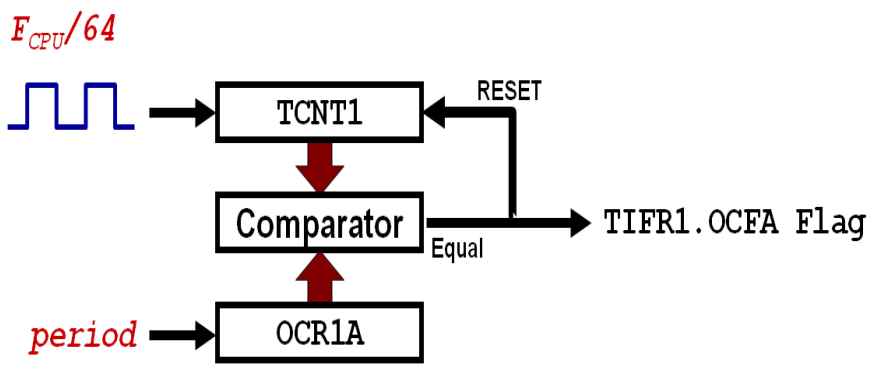
```
1#include <avr/io.h>
2#include <avr/interrupt.h>
3
4ISR(TIMER1_COMPA_vect, ISR_BLOCK)
5{
```

```
        6     PORTB ^= (1 << PORTB5);    // Toggle LED0
        7}
        8
        9int main(void)
       10{
       11    // Initialization
       12
       13    // Set LED as output
       14    DDRB |= (1 << PORTB5);     // Configure PB5 as digital output
       15    PORTB &= ~(1 << PORTB5);   // Set initial level for PB5
dieciséis
       17    // Set up Timer/Counter1
       18    TCCR1B |= (1 << WGM12 );   // Configure timer 1 for CTC mode
       19    OCR1A = 25000;             // Set CTC compare value to 10Hz (100mS)
       20                               // at 16MHz AVR clock, with a prescaler of 64
       21    TIMSK1 |= (1 << OCIE1A );  // Enable CTC interrupt
       22    TCCR1B |= ((1 << CS10 ) | (1 << CS11 )); // Start Timer/Counter1 at F_CPU/64
       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:



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<font></font>
 2#include <avr/io.h><font></font>
 3#include <avr/interrupt.h><font></font>
 4<font></font>
 5ISR(TIMER1_COMPA_vect, ISR_BLOCK)<font></font>
 6{<font></font>
 7    PORTB ^= (1 << PORTB5);   // Toggle LED0<font></font>
 8}<font></font>
 9<font></font>
10int main(void)<font></font>
11{<font></font>
12    // Initialization<font></font>
13<font></font>
14    // Set LED as output<font></font>
15    DDRB |= (1 << PORTB5);     // Configure PB5 as digital output        <font><
dieciséis    PORTB &= ~(1 << PORTB5);   // Set initial level for PB5<font></font>
17<font></font>
18    // Set up Timer/Counter1<font></font>
19    TCCR1B |= (1 << WGM12 );   // Configure timer 1 for CTC mode<font></font>
20    OCR1A = 25000;             // Set CTC compare value to 10Hz (100mS)<font></f
21                               // at 16MHz AVR clock, with a prescaler of 64<fon
22    TIMSK1 |= (1 << OCIE1A );  // Enable CTC interrupt<font></font>
23    TCCR1B |= ((1 << CS10 ) | (1 << CS11 )); // Start Timer/Counter1 at F_CPU/64
24<font></font>
25    // Enable all interrupts<font></font>
26    sei(); <font></font>
27<font></font>
28    while(1);<font></font>
29}<font></font>
30<font></font>
```

Este es un ejemplo de una interrupción no persistente . 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<font></font>
2#include <avr/io.h><font></font>
3#include <avr/interrupt.h><font></font>
4<font></font>
5ISR(TIMER1_COMPA_vect, ISR_BLOCK)<font></font>
6{<font></font>
7    PORTB ^= (1 << PORTB5);   // Toggle LED0<font></font>
8}<font></font>
9<font></font>
10int main(void)<font></font>
11{<font></font>
12    // Initialization<font></font>
13<font></font>
14    // Set LED as output<font></font>
15    DDRB |= (1 << PORTB5);     // Configure PB5 as digital output        <font><
dieciséis    PORTB &= ~(1 << PORTB5);   // Set initial level for PB5<font></font>
17<font></font>
18    // Set up Timer/Counter1<font></font>
19    TCCR1B |= (1 << WGM12 );   // Configure timer 1 for CTC mode<font></font>
20    OCR1A = 25000;             // Set CTC compare value to 10Hz (100mS)<font></f
21                               // at 16MHz AVR clock, with a prescaler of 64<for
22    TIMSK1 |= (1 << OCIE1A );  // Enable CTC interrupt<font></font>
23    TCCR1B |= ((1 << CS10 ) | (1 << CS11 )); // Start Timer/Counter1 at F_CPU/64
24<font></font>
25    // Enable all interrupts<font></font>
26    sei(); <font></font>
27<font></font>
28    while(1);<font></font>
29}<font></font>
30<font></font>
```

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>
2uint8_t flag;<font></font>
3...<font></font>
4ISR(SOME_vect) {<font></font>
```

```
 5    flag = 1;<font></font>
 6}<font></font>
 7...<font></font>
 8while(flag == 0) {<font></font>
 9...<font></font>
10}<font></font>
11<font></font>
```

el compilador generalmente accederá a "bandera" solo una vez y optimizará los accesos adicionales por completo, 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>
 2volatile uint8_t flag;<font></font>
 3...<font></font>
 4ISR(SOME_vect) {<font></font>
 5    flag = 1;<font></font>
 6}<font></font>
 7...<font></font>
 8while(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 dondequiera que se actualice o lea la variable, 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>
        6volatile uint16_t ctr;<font></font>
        7<font></font>
        8ISR(TIMER1_OVF_vect)<font></font>
        9{<font></font>
       10  ctr--;<font></font>
       11}<font></font>
       12...<font></font>
       13int<font></font>
       14main(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 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
 7volatile uint16_t ctr;
 8
 9ISR(TIMER1_OVF_vect)
10{
11  ctr--;
12}
13...
14int main(void)
15{
dieciséis   uint_16 ctr_copy;
17   ...
18   ctr = 0x0200;
19   start_timer();
20   do
21   {
22     ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
23     {
24       ctr_copy = ctr;
25     }
26   } while(ctr != 0);
27     // wait
28       ;
29   ...
30}
31
```

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.