

# Tecnicatura Superior en Telecomunicaciones

## Electrónica Microcontrolada-TST-2022

### Grupo # 4

#### ➤ Profesores:

- Jorge Morales
- Gonzalo Vera

#### ➤ Integrantes:

- Daniella Mazzini
- Ivan Exequiel Gomez
- Roxana Vicentelo
- Alfredo Palacios
- Matias Lujan
- Maximo Santillan

#### ➤ Repositorio:

- <https://github.com/EMTSTISPC/Grupo4>

# Descripción general del oscilador megaAVR®

Los microcontroladores microchip megaAVR® de 8 bits tienen varias opciones de fuente de reloj, seleccionables mediante la programación de los **bits de fusibles** (`/8avr:avrfuses`) **CKSEL Flash**. Esta discusión es específica para el MCU ATmega328PB (<http://www.microchip.com/wwwproducts/en/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 de reloj del sistema** se puede cambiar durante el tiempo de ejecución escribiendo en el registro **del preescalador de reloj del sistema** (`/8avr:osc-mega-overview#system-clock-prescaler`) (CLKPR).

Cada fuente de reloj proporciona una opción de retraso después del restablecimiento o encendido del dispositivo para mantener el dispositivo restablecido hasta que se suministre con un V<sub>cc</sub> mínimo. El reloj de la fuente seleccionada se introduce en el generador de reloj AVR® y se enruta a los módulos apropiados.

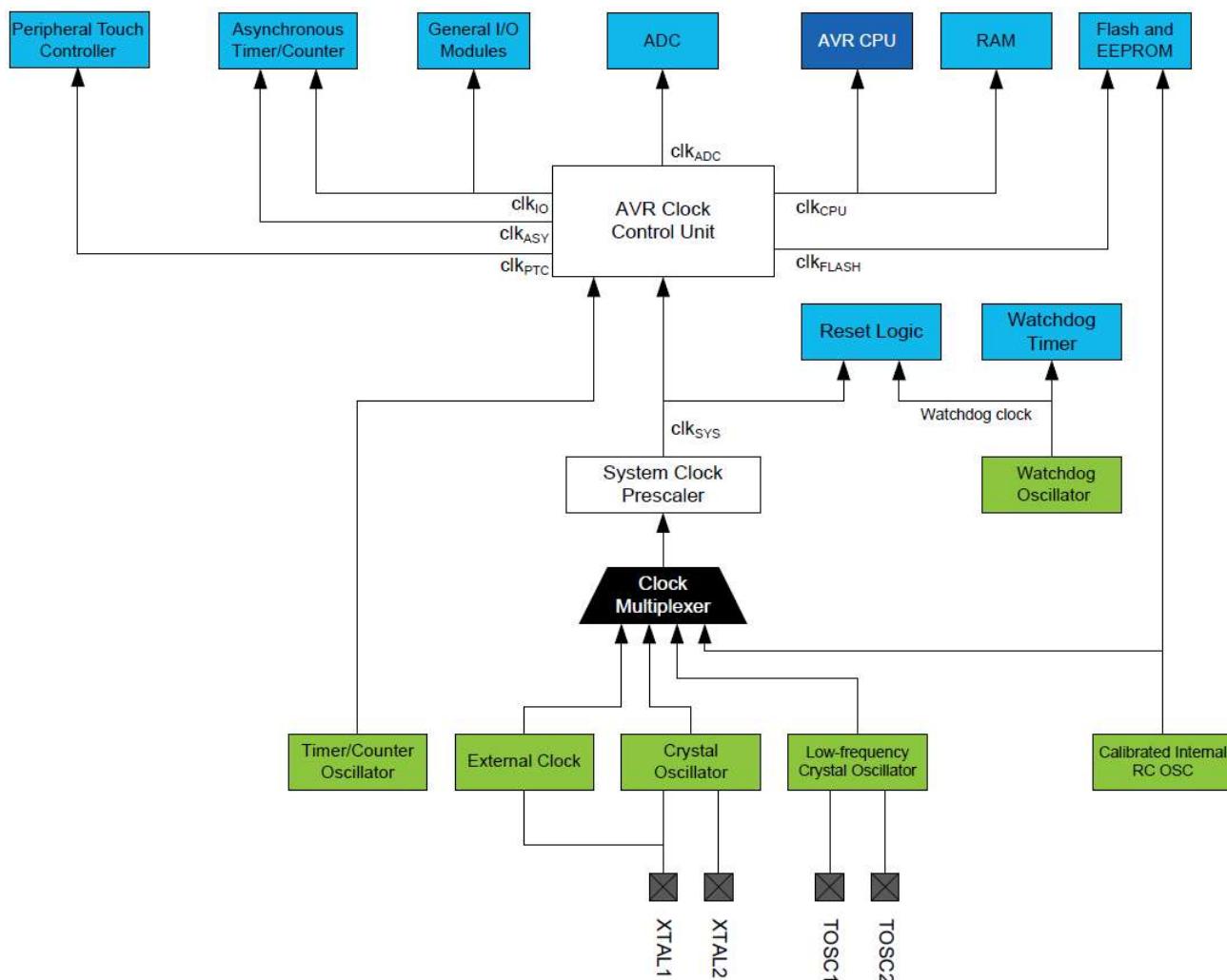


La **frecuencia máxima de funcionamiento de megaAVR® depende de V<sub>cc</sub>**. El software de aplicación debe asegurarse de que la frecuencia de la fuente de reloj seleccionada se encuentra dentro del área de operación segura (ver sección 33.4

en la hoja de datos del dispositivo (<http://ww1.microchip.com/downloads/en/DeviceDoc/40001906A.pdf>).

## Visión general

La siguiente figura ilustra los principales sistemas de reloj en el dispositivo y su distribución. No es necesario que todos los relojes estén activos a una hora determinada. Con el fin de reducir el consumo de energía, los relojes de los módulos que no se utilizan se pueden detener mediante el uso de diferentes modos de suspensión (/8avr:avrsleep). Los sistemas de reloj se describen en las siguientes secciones. La frecuencia de reloj del sistema se refiere a la frecuencia generada a partir del preescalador de reloj del sistema. Todas las salidas de reloj de la unidad de control de reloj AVR funcionan a la misma frecuencia.



(/local--files/8avr:osc-mega-overview/atmega328pb-sys-clk-distribution.png)

## Fuentes de reloj

El dispositivo tiene las siguientes opciones de fuente de reloj, seleccionables a través de bits **CKSEL** Flash Fuse como se muestra a continuación. El reloj de la fuente seleccionada se introduce en el 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

(/local--files/8avr:osc-mega-overview/atmega328pb-clk-sources.png)

## Origen de reloj predeterminado

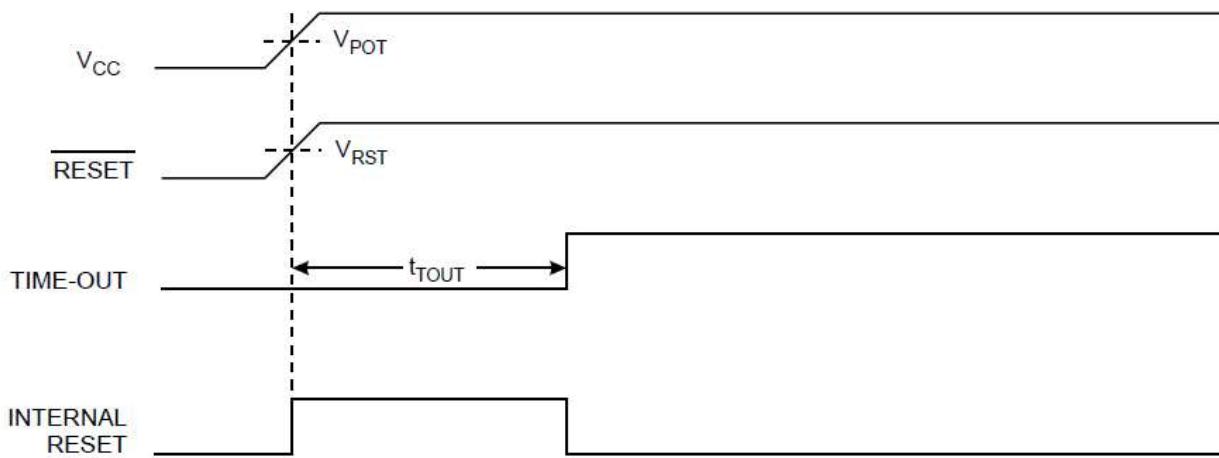
El dispositivo se envía con el oscilador RC interno seleccionado a 8.0 MHz y con el fusible CKDIV8 programado, lo que resulta en un reloj del sistema de 1.0 MHz. El tiempo de inicio se establece en máximo y el período de tiempo de espera está habilitado: CKSEL=0010, SUT=10, CKDIV8=0. Esta configuración predeterminada garantiza que todos los usuarios puedan realizar la configuración de origen de reloj deseada utilizando cualquier interfaz de programación disponible.

## Secuencia de inicio del reloj

Cualquier fuente de reloj necesita (i) una V suficiente<sub>Cc</sub> para empezar a oscilar y (ii) un número mínimo de ciclos oscilantes antes de que pueda considerarse estable.

### Estabilidad de Vcc

Para garantizar una V suficiente<sub>Cc</sub>, el dispositivo emite un restablecimiento interno con un retraso de tiempo de espera ( $t_{PRESUMIR}$ ) después de que el restablecimiento del dispositivo sea liberado por todas las demás fuentes de restablecimiento:



(/local--files/8avr:osc-mega-overview/atmega328pb-tout-delay.png)

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

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

(/local--files/8avr:osc-mega-overview/atmega328pb-tout-values.png)



$V_{CC}$  no se supervisa durante el retraso, por lo que es necesario seleccionar un retraso superior al  $V_{CC}$  tiempo de ascenso. Si esto no es posible, se debe utilizar un circuito interno o externo de detección de apagado (DBO). Un circuito BOD asegurará suficiente  $V_{CC}$  antes de que se libere el restablecimiento, y el retraso de tiempo de espera se puede deshabilitar. No se recomienda deshabilitar el retraso de tiempo de espera sin utilizar un circuito de detección de salida marrón.

## Estabilidad del oscilador

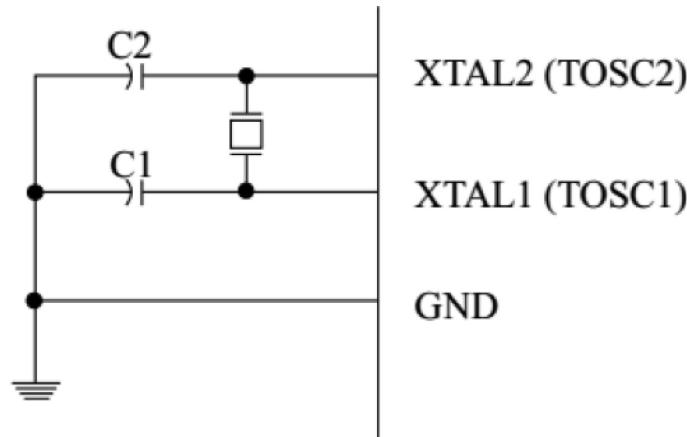
Se requiere que el oscilador oscile durante un número mínimo de ciclos antes de que el reloj se considere estable. Un contador de ondulación interno monitorea el reloj de salida del oscilador y mantiene activo el restablecimiento interno durante un número determinado de ciclos de reloj. El restablecimiento se libera y el dispositivo comenzará a ejecutarse. El tiempo de arranque del oscilador recomendado depende del tipo de reloj y varía de 6 ciclos para un reloj aplicado externamente a 32K ciclos para un cristal de baja frecuencia.



Consulte la sección 11 de la hoja de datos del dispositivo (<http://ww1.microchip.com/downloads/en/DeviceDoc/40001906A.pdf>), que especifica el número de ciclos de retardo CK para cada tipo de fuente de reloj y la configuración del fusible SUTx.

## Oscilador de cristal de baja potencia

Los pines XTAL1 y XTAL2 son entrada y salida, respectivamente, de un amplificador inversor que se puede configurar para su uso como oscilador en chip, como se muestra en la figura a continuación. Se puede utilizar un cristal de cuarzo o un resonador de cerámica:



(/local--files/8avr:osc-mega-overview/atmega328pb-xtal-connection.png)

El oscilador de baja potencia puede funcionar en tres modos diferentes, cada uno optimizado para un rango de frecuencia específico. El modo de funcionamiento es seleccionado por los fusibles CKSEL[3:1], como se muestra en la siguiente tabla:

Frequency Range [MHz]	CKSEL[3:1] <sup>(2)</sup>	Range for total capacitance of C1 and C2 [pF] <sup>(4)</sup>
0.4 - 0.9	100 <sup>(3)</sup>	–
0.9 - 3.0	101	12 - 22
3.0 - 8.0	110	12 - 22
8.0 - 16.0	111	12 - 22

(/local--files/8avr:osc-mega-overview/atmega328pb-xtal-modes.png)

El fusible CKSEL0 junto con los fusibles SUT[1:0] seleccionan los tiempos de arranque (consulte la sección 11.3 de la hoja de datos del dispositivo (<http://ww1.microchip.com/downloads/en/DeviceDoc/40001906A.pdf>)).

## 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 arranque son determinados por los fusibles SUT.

## Oscilador RC interno calibrado

De forma predeterminada, el oscilador RC interno proporciona un reloj de 8,0 MHz. Aunque el voltaje y la temperatura dependen, este reloj puede ser calibrado con mucha precisión por el usuario. El dispositivo se envía con el fusible CKDIV8 programado, 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 a '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 la aplicación AVR053 ([http://ww1.microchip.com/downloads/en/AppNotes/Atmel-2555-Internal-RC-Oscillator-Calibration-for-tinyAVR-and-megaAVR-Devices\\_ApplicationNote\\_AVR053.pdf](http://ww1.microchip.com/downloads/en/AppNotes/Atmel-2555-Internal-RC-Oscillator-Calibration-for-tinyAVR-and-megaAVR-Devices_ApplicationNote_AVR053.pdf)), que describe el procedimiento para volver a calibrar el oscilador RC interno.

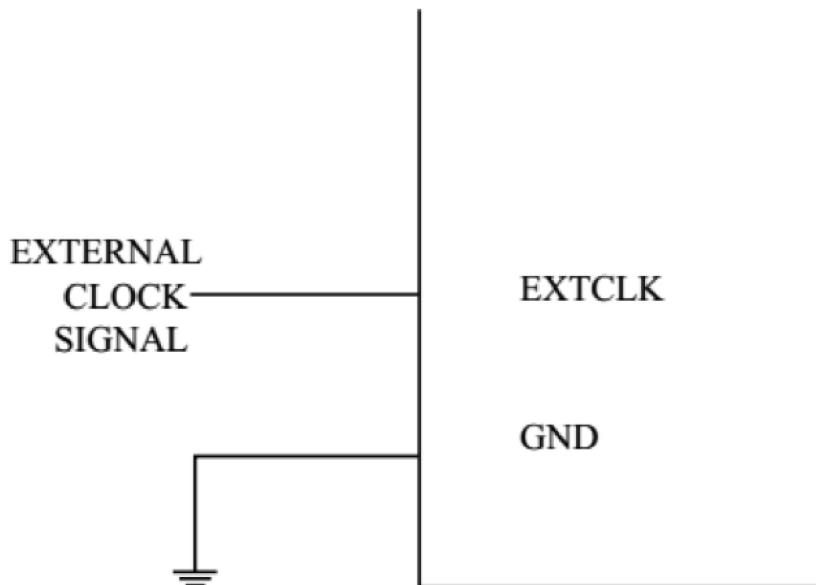
## Oscilador interno de 128 kHz

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 el reloj del sistema programando los fusibles CKSEL a '0011'.

## Reloj externo

Para manejar el dispositivo desde una fuente de reloj externa, EXTCLK debe ser conducido 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



(/local--files/8avr:osc-mega-overview/atmega328pb-ext-clk-connection.png)

## Búfer de salida de reloj

El dispositivo puede emitir 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 utiliza para conducir 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, es el reloj del sistema dividido el que se emite.

## Temporizador/Contador Oscilador

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

En este dispositivo, los pines del temporizador/oscilador de contador (TOSC1 y TOSC2) se comparten con EXTCLK. Cuando se utiliza el temporizador / oscilador de contador, el reloj del sistema debe ser cuatro veces la frecuencia del oscilador. Debido a esto y al uso compartido de pines, el temporizador / oscilador de contador solo se puede usar cuando se selecciona el oscilador RC interno calibrado como fuente de reloj del sistema. La aplicación de una fuente de reloj externa a TOSC1 se puede realizar si el bit Habilitar entrada de reloj externo en el Registro de estado asincrónico (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 del reloj externo como entrada en lugar de un cristal de reloj de 32.768 kHz.

## Preescalador de 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 característica se puede utilizar para disminuir la frecuencia de 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.  $\text{Clk}_{\text{E/S}}$ ,  $\text{Clk}_{\text{Adc}}$ ,  $\text{Clk}_{\text{CPU}}$ , y  $\text{clk}_{\text{FLASH}}$  se dividen por un factor como se muestra en la descripción del 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.

(/local--files/8avr:osc-mega-overview/atmega328pb-clkpr-1.png)

<b>CLKPS[3:0]</b>	<b>Clock Division Factor</b>
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

(/local--files/8avr:osc-mega-overview/atmega328pb-clkpr-2.png)

## Escribiendo a CLKPR

Al cambiar entre la configuración del preescalador, el preescalador del reloj del sistema garantiza que no se produzcan fallos en el sistema de reloj. También garantiza 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 ondulación que implementa el preescalador se ejecuta a la frecuencia del reloj indiviso, que puede ser más rápido que la frecuencia de reloj de la CPU. Por lo tanto, no es posible determinar el estado del preescalador: incluso si fuera legible, el tiempo exacto que lleva 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]), se tarda entre  $T_1 + T_2$  y  $T_1 + 2 * T_2$  antes de que la nueva frecuencia de reloj esté activa. En este intervalo, se producen dos bordes de reloj activos. Aquí,  $T_1$  es el período de reloj anterior, y  $T_2$  es el período correspondiente a la nueva configuración del preescalador. Para evitar cambios involuntarios de frecuencia de 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 a cero: CLKPR = 0x80.

2. En cuatro ciclos, escriba el valor deseado en CLKPS[3:0] mientras escribe un cero en CLKPCE: CLKPR=0x0N



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

## 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.



```

1  #include <stdint.h> // St?
2  #include <avr/io.h> // SFR
3  #include <avr/interrupt.h>
4
5  void clkPrescaleSet(uint8_t
6    cli();
7    CLKPR = (1 << CLKPCE);
8    CLKPR = divisionFactor;
9    sei();
10 }
```



Para ver esta función en uso, visite el [proyecto de ejemplo de oscilador megaAVR® \(/8avr:osc-mega-example\)](#)

## Fusible CKDIV8 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 en el arranque. 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 cualquier valor se puede escribir en los bits CLKPS independientemente de la configuración del fusible CKDIV8. El software de 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.



## Aprende más



## Proyecto

**de ejemplo de oscilador megaAVR®** Obtenga más información > (/avr:osc-mega-example)

# Proyecto de ejemplo de oscilador megaAVR®

## ⌚ Objetivo

Esta página proporciona un proyecto simple que demuestra el ajuste dinámico de la **frecuencia del reloj del sistema** a través de la modificación del registro de preescalador del reloj del sistema (CLKPR) (/avr:osc-mega-overview#system-clock-prescaler) en dispositivos megaAVR®. El ejemplo de código se ejecuta en la MCU **ATmega328PB** (<http://www.microchip.com/wwwproducts/en/ATmega328PB>).

La **fuente de reloj del sistema** se establece a través de bits de fusible de configuración para que sea el reloj externo de 16 MHz proporcionado por el chip mEDBG (consulte el diagrama de conexión a continuación).

El proyecto configura el módulo Timer/Counter1 para que funcione en modo **Clear-Timer-On-Compare** (CTC) y, en una coincidencia de período, genera una interrupción de "tick" cada 100 mS. La fuente del reloj del temporizador está configurada para ser **SYS\_CLK/64**.

El ISR timer/counter1 alterna LED0 e incrementa un contador. El bucle principal del programa supervisa el valor de Counter y actualiza el valor de CLKPR cada 10 segundos para cambiar dinámicamente la frecuencia SYS\_CLK.

CLKPR se alterna entre la configuración div/1 y div/4, cambiando así el intervalo de alternancia (interrupción) de 100 ms a 400 ms respectivamente. Esto se puede ver como la velocidad de parpadeo LED0 cambia cada 10 segundos.



Revise el archivo `main.c` del proyecto para obtener comentarios más detallados y una descripción de la operación.

## ☑ Materiales

### Herramientas de hardware

Herramienta	Acerca de
( <a href="http://www.atmel.com/tools/MEGA328PB-XMINI.aspx">http://www.atmel.com/tools/MEGA328PB-XMINI.aspx</a> )	<b>ATmega328PB Xplained Mini</b> Kit de evaluación <a href="http://www.atmel.com/tools/MEGA328PB-XMINI.aspx">(http://www.atmel.com/tools/MEGA328PB-XMINI.aspx)</a> <a href="https://www.microchipdirect.com/prc...">https://www.microchipdirect.com/prc...</a>

### Herramientas de software

Herramienta	Acerca de	Instaladores			Instrucciones de instalación
		Windows	Linux	Mac OS X	
Estudio Atmel® Entorno de desarrollo integrado	<a href="#">(/atstudio:start)</a> <a href="http://studio.download.atmel.com/7.0.1931/as-installer-7.0.1931-full.exe">(http://studio.download.atmel.com/7.0.1931/as-installer-7.0.1931-full.exe)</a>	<a href="#"></a>	<a href="#"></a>	<a href="#"></a>	<a href="#">(/install:atstudio)</a>

### Archivos de ejercicio

Archivo	Windows	Linux
<a href="https://microchipdeveloper.com/8avr:osc-mega-example">https://microchipdeveloper.com/8avr:osc-mega-example</a>		

Proyecto  
de  
ejemplo

[\(local--files/8avr:osc-mega-example/8avr-mega-oscillator-example.zip\)](#)

[\(local--files/8avr:osc-mega-example/8avr-mega-oscillator-exampl](#)

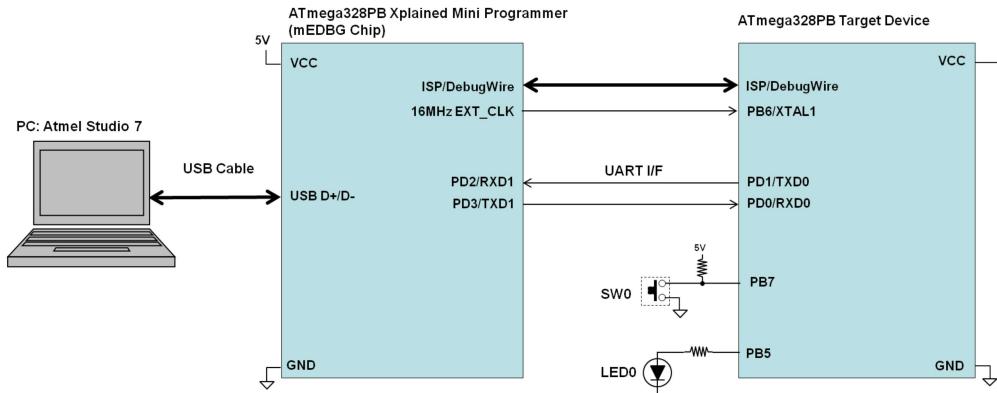


Recomendamos extraer el archivo .zip a su C:\ carpeta.

Debería ver la carpeta C:\MTT\8avr\mega\code-examples\oscillator-example\8avr-mega-oscillator-example que contiene la solución 8avr-mega-oscillator-example.atsln

## 🔧 Diagrama de conexión

The mEDBG chip controls the programming/debug interface, as well as supplying a 16 MHz clock when the Xplained board is connected via USB cable to a PC.



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

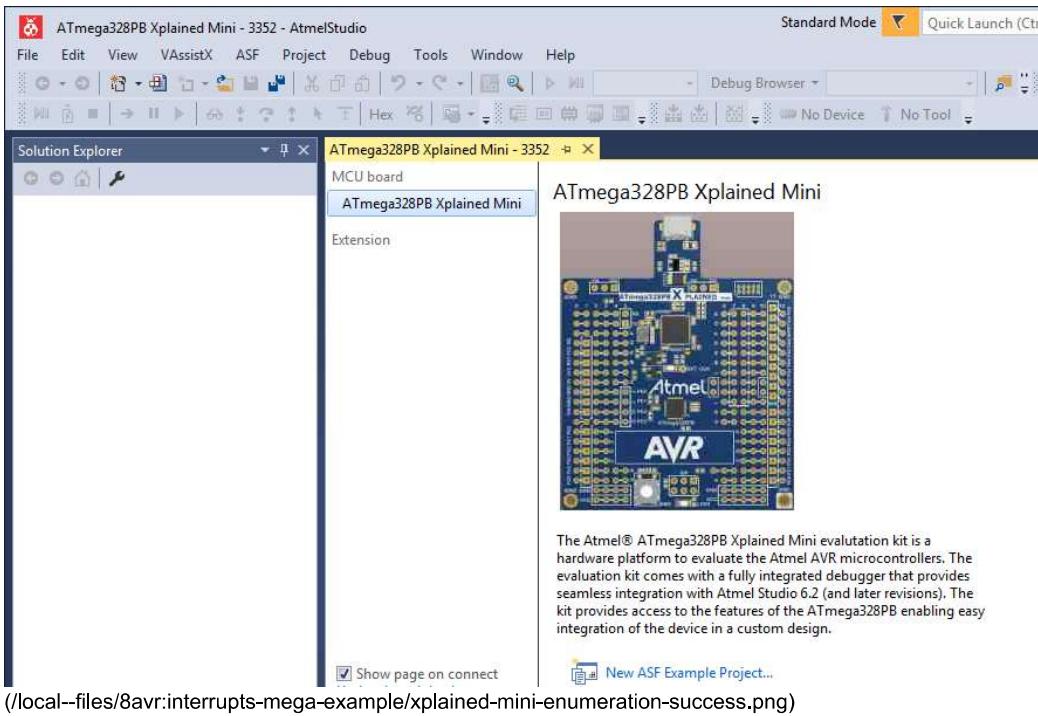


The target ATmega328PB CKSEL fuse bits are initially unchangeable on the Xplained Mini board from within Atmel Studio. "Verification" errors will be displayed if programming CLSEL bits with any other setting other than "external clock".

This may be overridden by clearing the **mEDBG fuse filter** as described in section 1.6.2 of the **ATmega328PB Xplained Mini User Guide** ([http://www.atmel.com/Images/Atmel-42469-ATmega328PB-Xplained-Mini\\_User-Guide.pdf](http://www.atmel.com/Images/Atmel-42469-ATmega328PB-Xplained-Mini_User-Guide.pdf)).

## ❗ Procedure

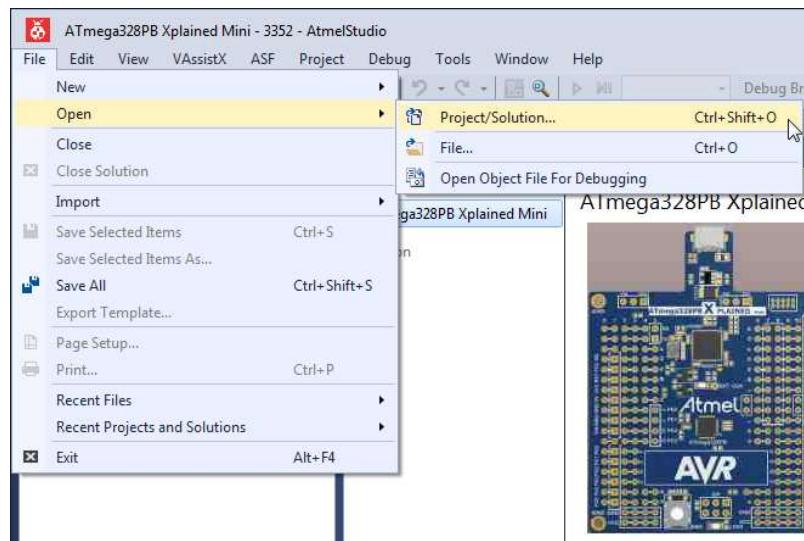
Attach the **ATmega328PB Xplained Mini board** to your computer using a USB-A-male-to-Micro-B-male cable. Start Atmel Studio 7. If the board has been successfully enumerated, you should see the board image come up in Studio as shown:



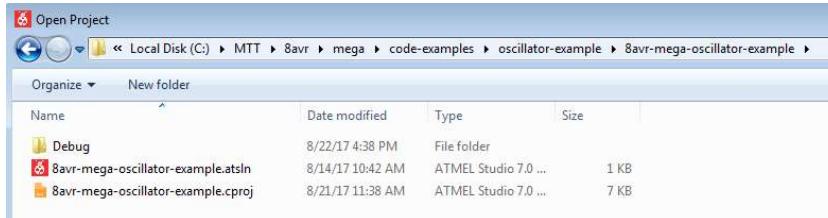
**i** The Xplained mini board is identified by the last four digits in its serial number (see sticker on bottom of board). In the above example, the last four digits are "3352"

## 1 Open the Solution

In Studio, select **File > Open > Project/Solution** and navigate to the saved location of the solution, then select the file `8avr-mega-oscillator-example.atsln`:

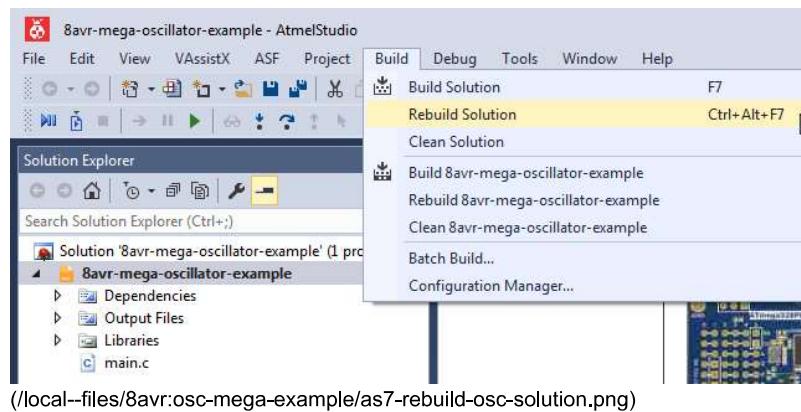


(/local--files/8avr:interrupts-mega-example/as7-open-solution.png)



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

## 2 Rebuild the Solution

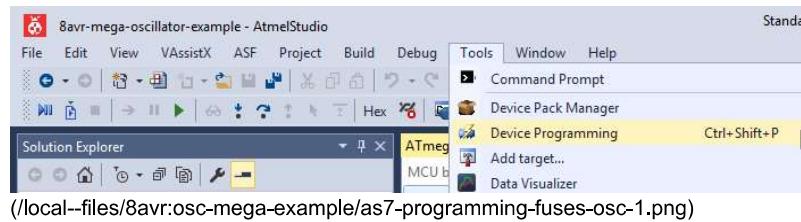


### 3 Program the Fuses

There are several key hardware configuration settings that need to be configured. The following **fuse settings** (**/8avr:avrfuses**) need to be programmed into the device:

- Ext: 0xFC
- High: 0xDF
- Low: 0xC0 (EXT CLK, Fast VDD rise, CLKDIV = 1)

Enter the Device Programming dialog as shown:



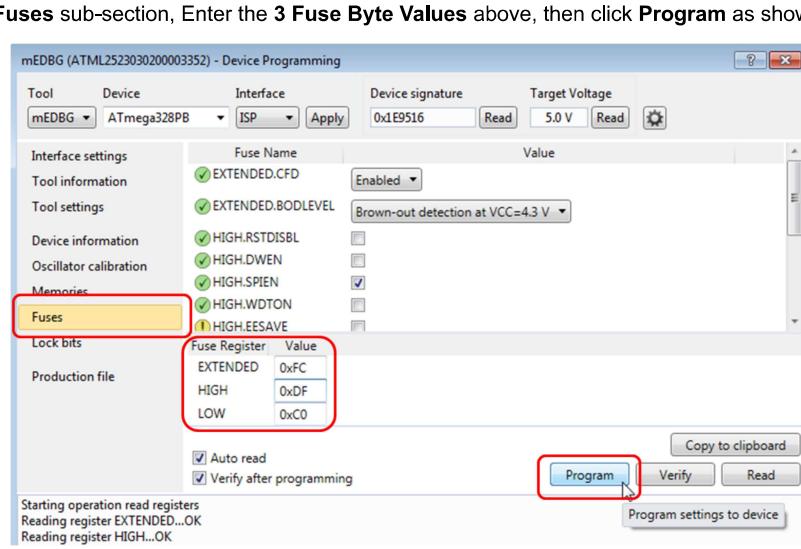
In the **Device Programming** dialog box, select **Tool**, **Device** and **Interface** as shown, then click **Apply**:



To verify a connection, select **Read** and verify that a **Device Signature** is found:

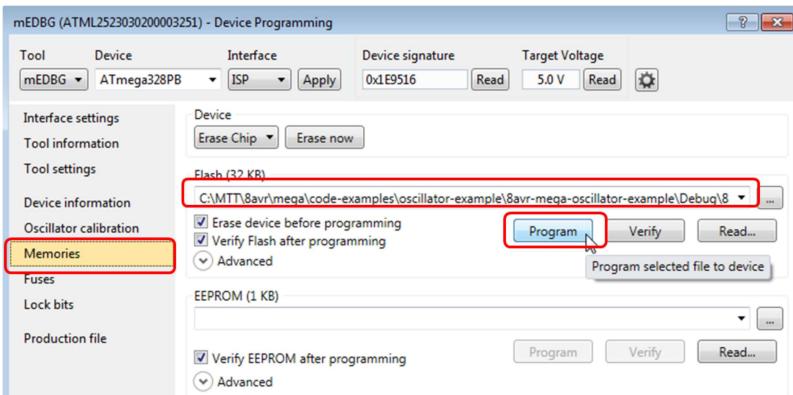


Select the **Fuses** sub-section, Enter the 3 **Fuse Byte Values** above, then click **Program** as shown:



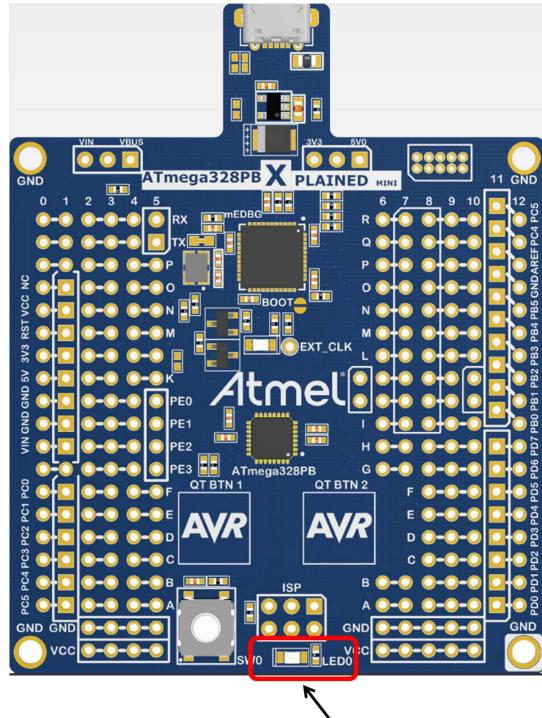
#### 4 Program the Hex File

While still in the Device Programming dialog box, select the **Memories** sub-section as shown. The path to the solution's hex file should already be listed in the dialog. Click **Program** as shown:



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

## ★ Results



**LED0 Toggle Frequency**  
**Changes Every 10 Seconds**  

- 10 Hz
- 2.5Hz

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

## 💡 Conclusions

This project has provided an example of how to dynamically adjust the system clock frequency on the megaAVR® MCU.

 **Learn More****megaAVR® Oscillator Overview**

Learn more &gt; (/avr:osc-mega-overview)

**megaAVR® Timer Overview**

Learn more &gt; (/avr:avrtimerover)

**megaAVR® Interrupts Overview**

Learn more &gt; (/avr:interrupts-mega-overview)

# Descripción general de AVR® USART

Los microcontroladores Microchip AVR® de 8 bits contienen un periférico de comunicación altamente flexible conocido como **USART** (Universal Synchronous and Asynchronous serial Receiver and Transmitter).

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 los dos modos principales: síncrono o asíncrono.

Este módulo se centra en el modo de operación **asíncrono** ([https://en.wikipedia.org/wiki/Universal\\_asynchronous\\_receiver/transmitter](https://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter)).

## Aprende más



Configuración

de megaAVR® USART Más información > (/8avr:uart-mega-configuration)



**megaAVR® USART Ejemplo (sondeado)**

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

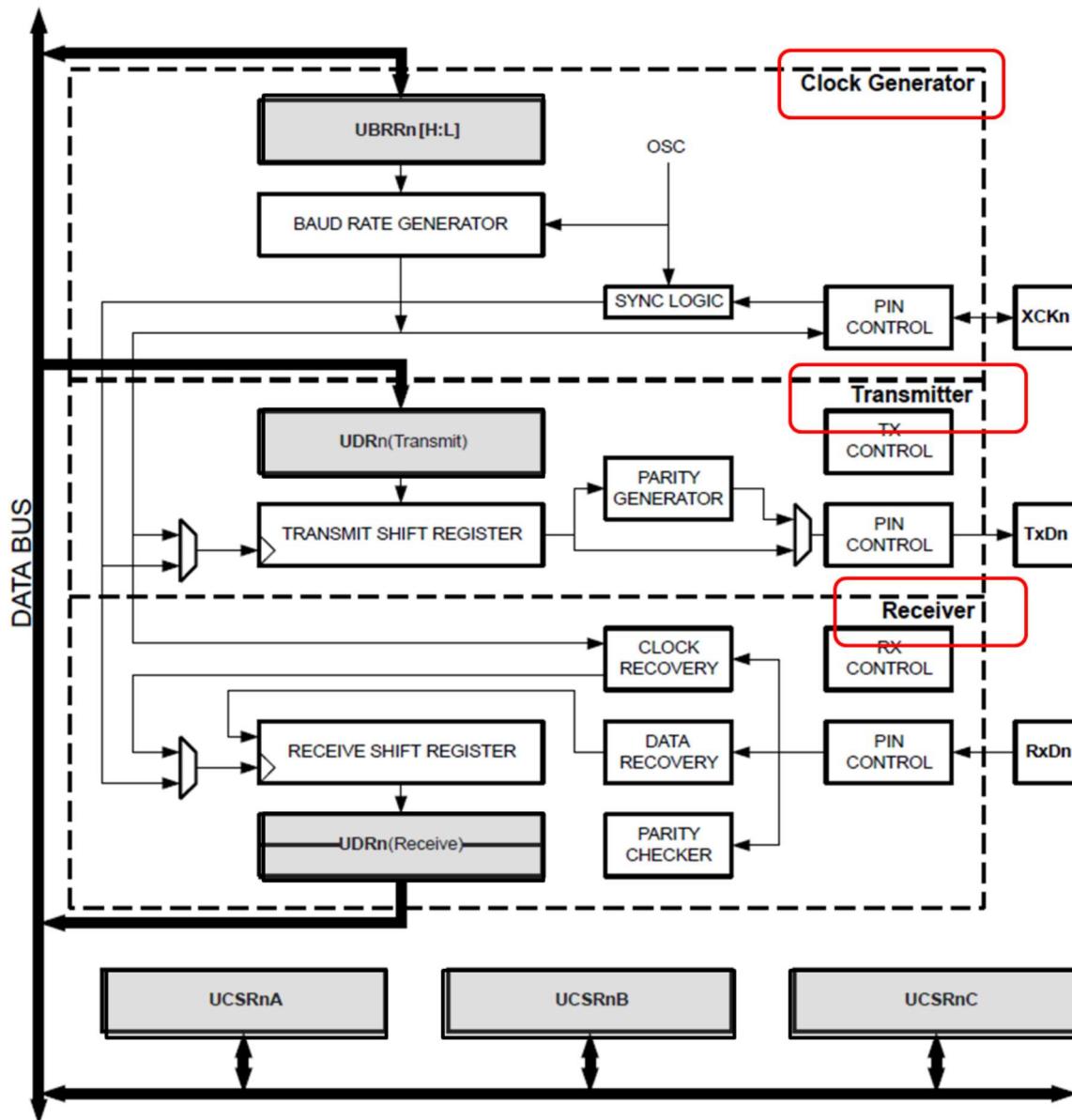
# Configuración de megaAVR® USART

---

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** (<http://www.microchip.com/wwwproducts/en/ATmega328PB>).

## Visión general

El módulo USART consta de tres secciones principales como se muestra en el siguiente diagrama: **Generador de reloj**, **Transmisor** y **Receptor**.

**Figure 24-1. USART Block Diagram**

(/local--files/8avr:uart-mega-configuration/uart-block-diagram.png)



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.
- El control de velocidad en baudios registro **UBRRn[H:L]** utilizado por el generador de reloj.



"n" en el nombre de 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**

## Uso del USART (Resumen)

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

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

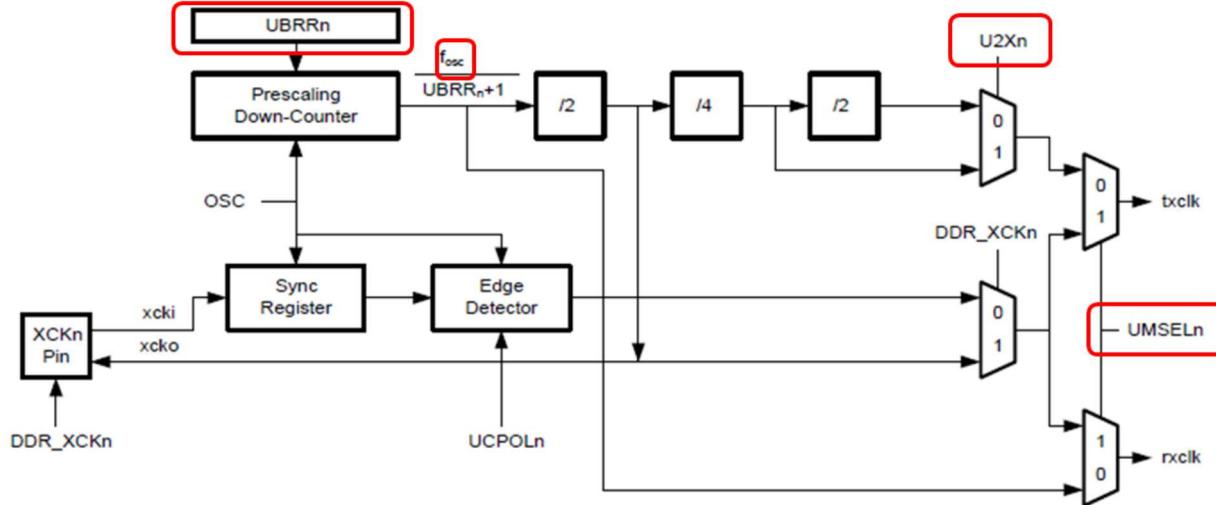
## Inicialización

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

- Ajuste de la velocidad en baudios,
- Configuración del formato de marco y
- Habilitación del Transmisor o del Receptor dependiendo del uso.

## Configuración de la velocidad en baudios

La generación de reloj interno se utiliza para el modo de operación asíncrono. La lógica de generación de reloj genera el reloj base para el transmisor y el receptor (se resaltan los registros de claves y los bits de control):

**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.
- fosc: System clock frequency.**

(/local--files/8avr:usart-mega-configuration/usart-clock-generator-diagram.png)

**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 asincrónico, los bits usart mode Select en el registro de control y estado USART C (**UCSRnC.UMSELn[1:0]**) se utilizan para seleccionar **la operación asincrónica (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

(/local--files/8avr:usart-mega-configuration/usart-mode-select-bits.png)

**Modo de doble velocidad (U2Xn)**

Para el modo asincrónico, la velocidad USART TX se puede duplicar estableciendo el bit U2Xn en el registro UCSRnA (**UCSRnA.U2Xn = 1**).



With double-speed mode set, the Receiver will only use half the number of samples (reduced from 16 to 8) for data sampling and clock recovery, and therefore a more accurate baud rate setting and system clock are required when this mode is used.

## Baud Rate Register (UBRRn)

The USART Baud Rate Register (**UBRRn**) and the down-counter connected to it function as a programmable prescaler or baud rate generator. The down-counter, running at system clock ( $f_{osc}$ ), is loaded with the UBRRn value each time the counter has counted down to zero or when the UBRRnL Register is written. A clock is generated each time the counter reaches zero. This clock is the baud rate generator clock output ( $= f_{osc}/(UBRRn+1)$ ). The Transmitter divides the baud rate generator clock output by 2, 8, or 16 depending on mode. The baud rate generator output is used directly by the Receiver's clock and data recovery units. However, the recovery units use a state machine that uses 2, 8, or 16 states depending on mode set by the state of the UMSEL, U2Xn and DDR\_XCK bits.

The table below contains equations for calculating the baud rate (in bits per second) and for calculating the UBRRn value for each mode of operation using an internally generated clock source.

**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)	$BAUD = \frac{f_{osc}}{16(UBRRn + 1)}$	$UBRRn = \frac{f_{osc}}{16BAUD} - 1$
Asynchronous Double Speed mode (U2Xn = 1)	$BAUD = \frac{f_{osc}}{8(UBRRn + 1)}$	$UBRRn = \frac{f_{osc}}{8BAUD} - 1$

(/local--files/8avr:usart-mega-configuration/usart-baud-equations.png)

- **BAUD:** Baud rate (in bits per second, bps)
- **f<sub>osc</sub>:** System oscillator clock frequency
- **UBRRn:** Contents of the UBRRnH and UBRRnL Registers, (0-4095).



The **AVR-LIBC Setbaud** ([http://www.nongnu.org/avr-libc/user-manual/group\\_\\_util\\_\\_setbaud.html](http://www.nongnu.org/avr-libc/user-manual/group__util__setbaud.html)) library contains useful macros for calculating the correct values to write to UBRRnH and UBRRnL registers. See initialization code example below.

Tables are also provided in the device data sheet containing UBRRn values for common Baud rates, given several oscillator frequencies:

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

Baud Rate [bps]	$f_{osc} = 16.0000\text{MHz}$				$f_{osc} = 18.4320\text{MHz}$				$f_{osc} = 20.0000\text{MHz}$			
	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%

(/local--files/8avr:uart-mega-configuration/usart-common-ubrrn-values.png)



For baud frequency calculations, it is generally accepted that error percentages of less than  $\pm 2\%$  are acceptable.

## Setting the Frame Format

USART Control and Status Register C (**UCSRnC**) is used to configure the UART communication frame format - parity, number of stop bits, and number of data bits. Settings for the typical “8N1” frame format are as follows:

- **UPM[1:0] = 00** for No Parity
- **USBS = 0** for 1 Stop Bit
- **UCSZ1[1:0] = 11** for 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
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

0 0 0 0 0 0 1 1 0

(/local--files/8avr:uart-mega-configuration/usart-frame-format-settings.png)

## Enabling the Transmitter

The USART Transmitter is enabled by setting the **Transmit Enable (TXEN)** bit in the **UCSRnB** Register:

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

Bit	7	6	5	4	3	2	1	0
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

(/local--files/8avr:uart-mega-configuration/usart-ucsrnb-txen.png)

When the Transmitter is enabled, the normal port operation of the TxDn pin is overridden by the USART and given the function as the Transmitter's serial output.



The baud rate, mode of operation and frame format must be set up once before doing any transmissions.

## Enabling the Receiver

The USART Receiver is enabled by writing the **Receive Enable (RXEN)** bit in the **UCSRnB** Register to '1':

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

Bit	7	6	5	4	3	2	1	0
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

(/local--files/8avr:uart-mega-configuration/usart-ucsrnb-rxen.png)

When the Receiver is enabled, the normal port operation of the RxDn pin is overridden by the USART and given the function as the Receiver's serial input.



The baud rate, mode of operation and frame format must be set up once before doing any transmissions.

## Code Example

The following USART initialization code example uses the **setbaud** utility ([http://www.nongnu.org/avr-libc/user-manual/group\\_\\_util\\_\\_setbaud.html](http://www.nongnu.org/avr-libc/user-manual/group__util__setbaud.html)) library in AVR-LIBC. This library provides macros that use the c-preprocessor to calculate appropriate values for **UBBRn**.

## Inputs

This header file requires that on entry values are already defined for **F\_CPU** and **BAUD**. In addition, the macro **BAUD\_TOL** will define the baud rate tolerance (in percent) that is acceptable during the calculations. The value of **BAUD\_TOL** will default to +/- 2%.

## Outputs

Assuming that the requested BAUD is valid for the given **F\_CPU** then the macro **UBRR\_VALUE** is set to the required prescaler value. Two additional macros are provided for the low and high bytes of the prescaler, respectively: **UBRRL\_VALUE** is set to the lower byte of the **UBRR\_VALUE** and **UBRRH\_VALUE** is set to the upper byte. An additional macro **USE\_2X** will be defined. Its value is set to 1 if the desired BAUD rate within the given tolerance could only be achieved by setting the **U2Xn** bit in the UART configuration. It will be defined to 0 if **U2Xn** is not needed.



```

1  #define F_CPU 16000000UL ? ▲
2  #define BAUD 38400UL
3  #define BAUD_TOL 2
4
5  #include <avr/io.h>
6  #include <util/setbaud.h>
7
8  void USART0_Init(void){
9
10     // Set the BAUD rate
11
12     UBRR0H = UBRRH_VALUE;
13     UBRR0L = UBRRL_VALUE;
14     #if USE_2X
15     UCSR0A |= (1 << U2X0);
16     #else
17     UCSR0A &= ~(1 << U2X0);
18     #endif
19
20     // Set the Mode & Frame
21
22     UCSR0C = 0x06;
23
24     // Enable USART0 Transmi
25
26     UCSR0B = (1 << TXEN0) |
27
28 }
```



The setbaud library generates warning messages during compilation if the input parameters generate a BAUD rate setting which will produce a baud rate outside of the desired **BAUD\_TOL**.

# Data Communications

## Transmit

A data transmission is initiated by loading the transmit buffer with the data to be transmitted. The CPU can load the transmit buffer by writing to the **UDRn** register. For polled operation, firmware should monitor the data-register-empty flag (**UCSRnA.UDREn**) before loading **UDRn**.

The buffered data in the transmit buffer will be moved to the Shift Register when the Shift Register is ready to send a new frame. The Shift Register is loaded with new data if it is in idle state (no ongoing transmission) or immediately after the last stop bit of the previous frame is transmitted. When the Shift Register is loaded with new data, it will transfer one complete frame at the rate given by the Baud Register.

**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							
Reset	0	0	0	0	0	0	0	0

(/local--files/8avr:usart-mega-configuration/usart-udrn.png)



The transmit complete interrupt flag (**USCRnA.TXCn**) is set and an optional TX interrupt may be generated (if enabled) when the entire frame in the shift register has been shifted out. The **USCRnA.TXCn** Flag bit is automatically cleared when a transmit complete interrupt is executed, or it can be cleared by writing a one to its bit location.

## Receive

The Receiver starts data reception when it detects a valid start bit. Each bit that follows the start bit will be sampled at the baud rate or XCKn clock, and shifted into the Receive Shift Register until the first stop bit of a frame is received. The receive buffer can then be read by reading the **UDRn** register. The complete reception of a byte can be checked by polling the **RXCn** bit in **UCSRnA** register.



The receive complete interrupt flag (**RXCn**) is set and an optional RX interrupt may be generated (if enabled) when the entire frame in the shift register has been copied to the **UDRn** register. This is a *persistent* interrupt - i.e. firmware must read the

received data from **UDRn** in order to clear the **RXCn** Flag

## Code Example

The following simple blocking APIs send and receive a byte of data via USART0.



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

## Learn More



### megaAVR® Interrupt Configuration

Learn more > (/avr:interrupts-mega-configuration)



### megaAVR® USART Example (Polled)

Learn more > (/avr:uart-mega-example-polled)

# MegaAVR® USART Ejemplo (Encuestado)

## ⌚ Objetivo

Esta página proporciona un proyecto simple que demuestra el funcionamiento sondeado del periférico **USART** en dispositivos megaAVR®. El ejemplo de código se ejecuta en la MCU **ATmega328PB** (<http://www.microchip.com/wwwproducts/en/ATmega328PB>).

El proyecto configura el módulo Timer/Counter1 para que funcione en modo **Clear-Timer-On-Compare** (CTC) y, en una coincidencia de período, genera una interrupción de "tick" cada 100 mS. El bucle principal monitorea estas señales de tick para implementar un reloj de hora del día (formato HH: MM: SS). LED0 también se alterna en cada evento de tick.

La pantalla del reloj se envía a USART0 TX cada segundo y se controla mediante los caracteres de "control" de entrada del usuario recibidos en USART0 RX:

- 'u' permite actualizaciones en la pantalla del reloj cada segundo
- 'f' bloquea las actualizaciones de la pantalla

La interfaz de reloj se muestra utilizando un programa de emulador de terminal ([/swtools:terminal-emulator](#)), como **Tera Term** (<https://osdn.net/projects/ttssh2/releases/>).

## ☑ Materiales

### Herramientas de hardware

Herramienta	ⓘ Acerca de
 <a href="http://www.atmel.com/tools/MEGA328PB-XMINI.aspx">(http://www.atmel.com/tools/MEGA328PB-XMINI.aspx)</a>	<a href="http://www.atmel.com/tools/MEGA328PB-XMINI.aspx">(http://www.atmel.com/tools/MEGA328PB-XMINI.aspx)</a> <a href="https://www.microchipdirect.com/prc/XMINI.aspx">(https://www.microchipdirect.com/prc/XMINI.aspx)</a>

### Herramientas de software

Herramienta	ⓘ Acerca de	Instaladores		
		Windows	Linux	Mac OSX
<b>Estudio Atmel®</b>	 Entorno de desarrollo integrado	<a href="http://atstudio:start">(/atstudio:start)</a> <a href="http://studio.download.atmel.com/7.0.1931/as-installer-7.0.1931-full.exe">Download (http://studio.download.atmel.com/7.0.1931/as-installer-7.0.1931-full.exe)</a>	<a href="#">Download</a>	<a href="#">Download</a> <a href="#">(/install)</a>
Emulador de terminal de Término Tera	<a href="#">(/swtools:terminal-emulator)</a>	<a href="https://en.osdn.jp/projects/ttssh2/releases/">Download (https://en.osdn.jp/projects/ttssh2/releases/)</a>	<a href="#">Download</a> <a href="#">Download</a>	<a href="#">Download</a> <a href="#">(/\$ir)</a>

### Archivos de ejercicio

Archivo	Windows	Linux	ⓘ Descargar

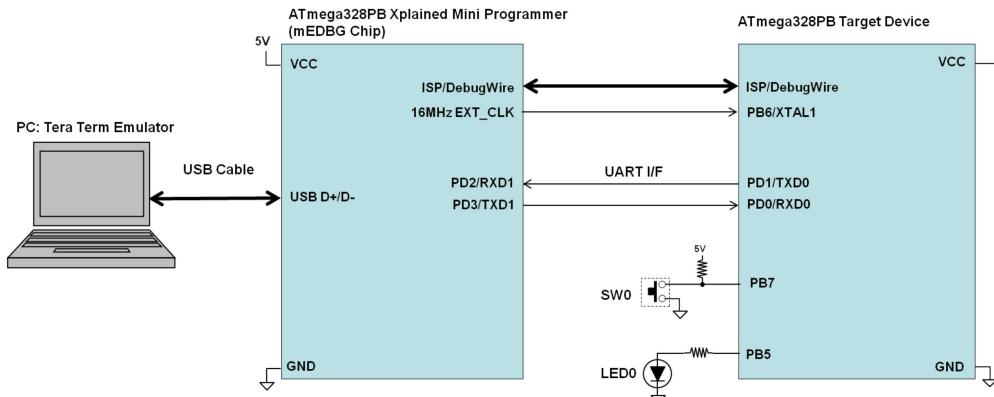
**Proyecto de ejemplo**
[\(local--files/8avr:uart-mega-example-polled/8avr-mega-uart-example-polled.zip\)](#)
[\(local--files/8avr:uart-mega-example-polled/8avr-mega-uart-example-polled.atsln\)](#)


Recomendamos extraer el archivo .zip a su C:\ carpeta.

Debería ver la carpeta C:\MTT\8avr\mega\code-examples\uart-example-polled\8avr-mega-uart-example-polled que contiene la solución 8avr-mega-uart-example-polled.atsln

## Diagrama de conexión

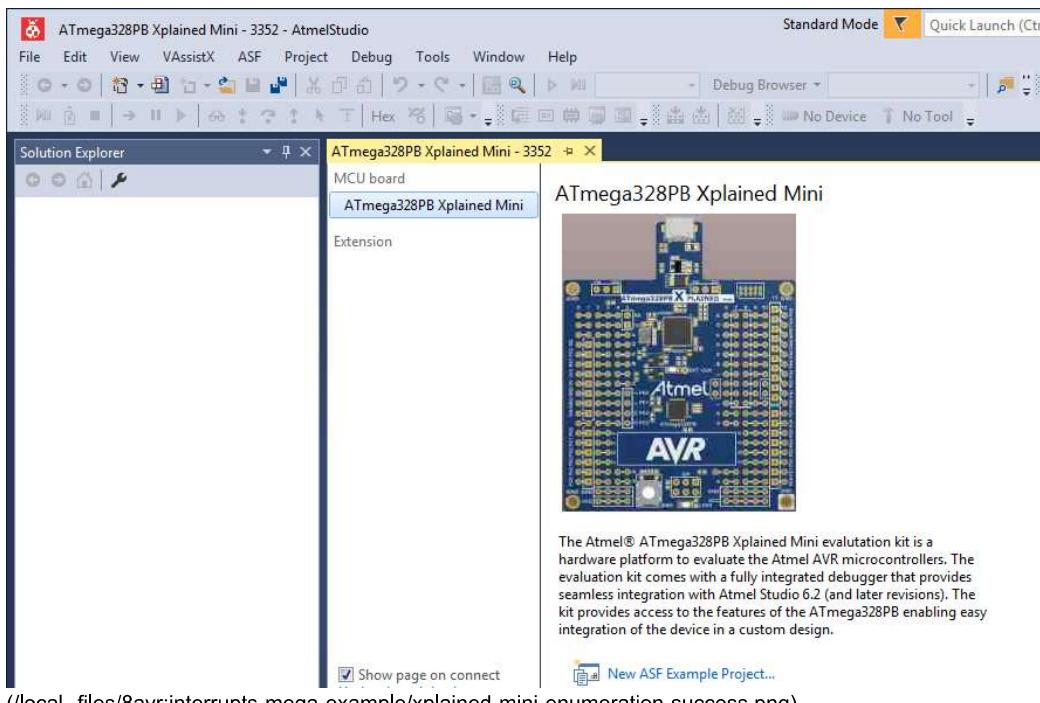
El módulo USART0 del dispositivo ATmega328PB de destino está conectado a la interfaz USART del chip mEDBG. El chip mEDBG realiza la conversión serie USB enumerando como un puerto COM virtual de clase CDC en la PC y presentando los datos USART de destino en esta interfaz. Tenga en cuenta que el mEDBG también controla la interfaz de programación / depuración, además de suministrar un reloj de 16MHz cuando la placa Xplained se conecta a través de un cable USB a una PC.



(/local--files/8avr:uart-mega-example-polled/xplained-mini-connection-diagram-teraterm.png)

## Procedure

Attach the ATmega328PB Xplained Mini board to your computer using a USB-A-male-to-Micro-B-male cable. Start Atmel Studio 7. If the board has been successfully enumerated, you should see the board image come up in Studio as shown:



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



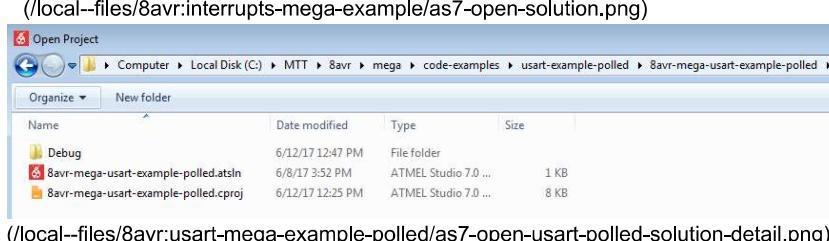
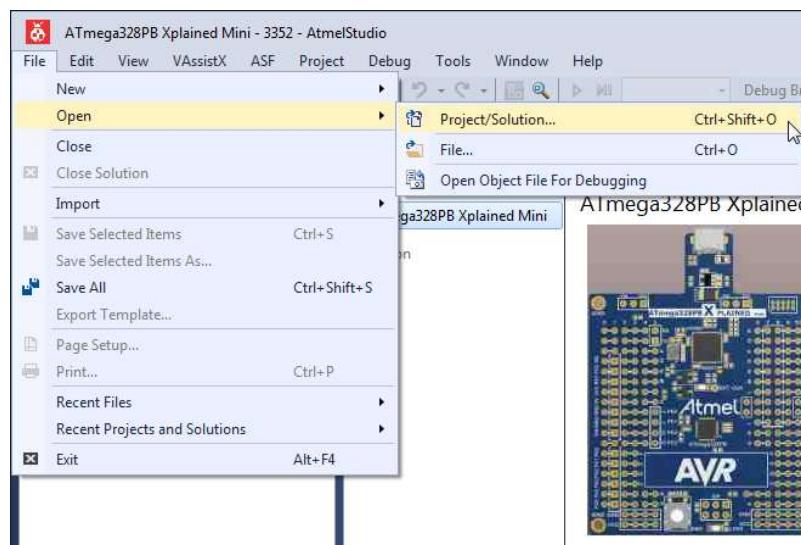
The Xplained mini board is identified by the last four digits in its serial number (see sticker on bottom of board). In the above example, the last four digits are "3352".

You should also see a **mEDBG Virtual COM Port** enumerated in your Windows Device Manager viewer. Please note the COM port number assigned to your board:



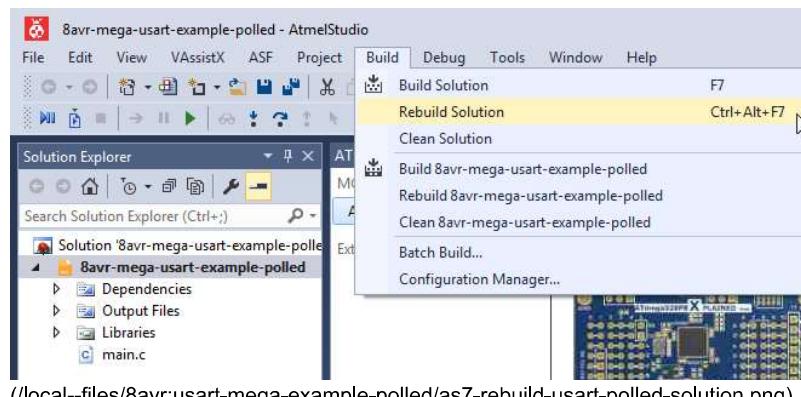
## 1 Open the Solution

In Studio, select File » Open » Project/Solution and navigate to the saved location of the solution:



To review the procedures for configuring/using the megaAVR® USART (as implemented in the project's `main.c` file), please review the **megaAVR® USART Configuration (8avr:usart-mega-configuration)** page.

## 2 Rebuild the Solution

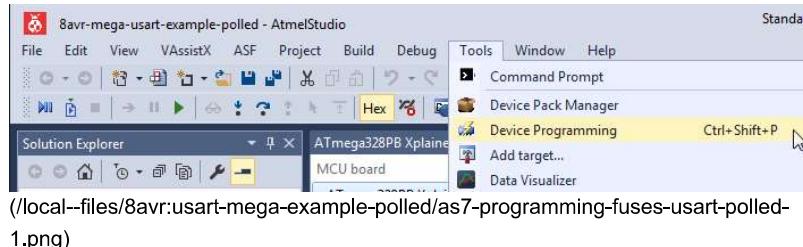


### 3 Program the Fuses

There are several key hardware configuration settings that need to be configured. The following **fuse settings** (`/avr:fuses`) need to be programmed into the device:

- HIGH: 0xDF
- LOW: 0xC0
- EXT: 0xFC

Enter the Device Programming dialog as shown:



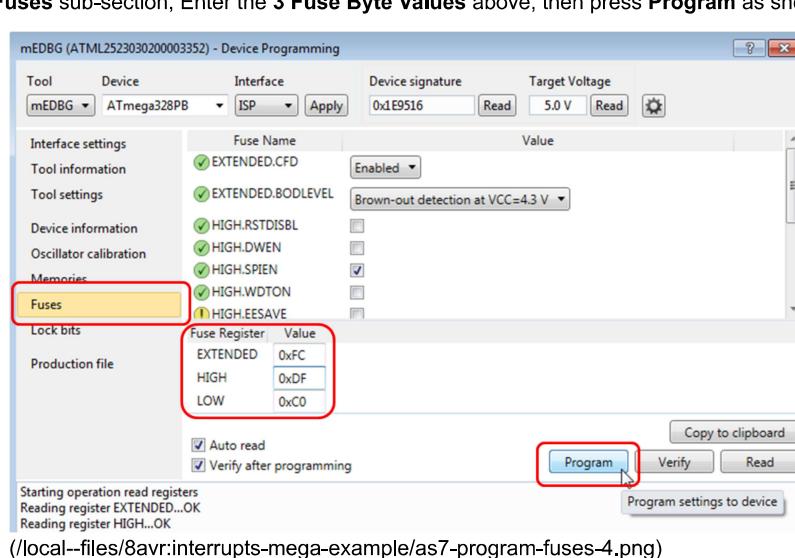
In the Device Programming dialog box, select the **Tool**, **Device** and **Interface** as shown, then press **Apply**:



To verify a connection, select **Read** and verify that a **Device Signature** is found:

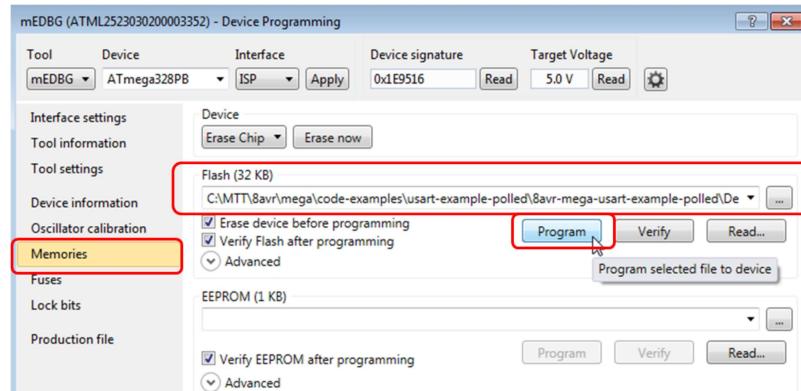


Select the **Fuses** sub-section, Enter the **3 Fuse Byte Values** above, then press **Program** as shown:



### 4 Program the Hex File

While still in the Device Programming dialog box, select the **Memories** sub-section as shown. The path to the solution's hex file should already be listed in the dialog. Press **Program** as shown:



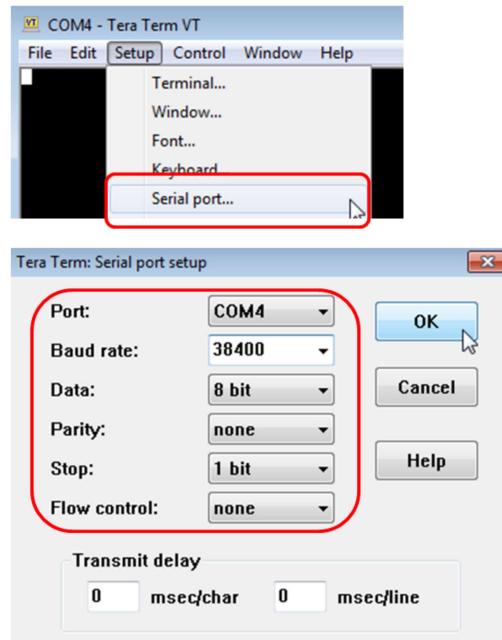
(/local--files/8avr:uart-mega-example-polled/as7-program-hex-usart-polled-1.png)

- 5 Start the Tera Term application. Select the correct mEDBG COM port in the dialog box that appears:



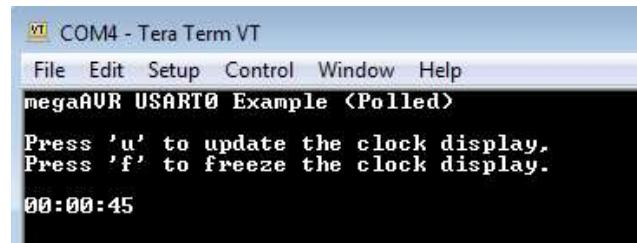
(/local--files/8avr:uart-mega-example-polled/teraterm-select-com-port.png)

Next, set the serial port parameters to match the project defaults: 38400 baud, 8-data, no parity, 1-stop, no flow-control as shown:



(/local--files/8avr:uart-mega-example-polled/teraterm-select-com-parameters.png)

## ★ Results



The screenshot shows a terminal window titled "COM4 - Tera Term VT". The menu bar includes "File", "Edit", "Setup", "Control", "Window", and "Help". The title bar says "megaAVR USART0 Example <Polled>". The main text area displays the message "Press 'u' to update the clock display." and "Press 'f' to freeze the clock display." followed by the time "00:00:45".

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



You may see gibberish on the display after resetting the communication parameters. Simply perform a board reset by shorting RST to GND, or by re-programming the hex file into the board again (see step 4 above).

## Conclusions

This project has provided an example of how to setup and use the USART module on the megaAVR® MCU.

## Learn More



### megaAVR® USART Overview

Learn more > (/8avr:usart-overview)



### megaAVR® USART Configuration

Learn more > (/8avr:usart-mega-configuration)

# 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

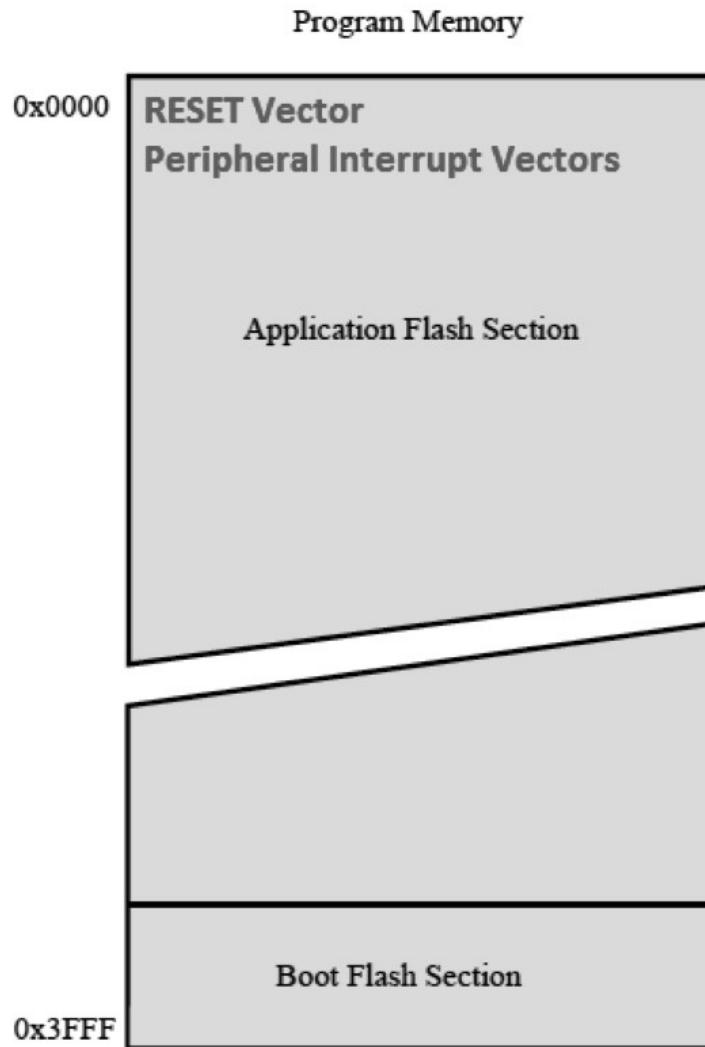
Bit	7	6	5	4	3	2	1	0
Access	R/W							
Reset	0	0	0	0	0	0	0	0

(/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:



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

## 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)<for
3 {<font></font>
4     uchar temp;<font></font>
5     /* GET MCUCR */<font></font><font>
6     temp = MCUCR;<font></font>
7     /* Enable change of Interrup
8     MCUCR = temp | (1 << IVCE);<
9     /* Move interrupts to Boot !
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 AVR-LIBC** ([http://www.nongnu.org/avr-libc/user-manual/group\\_\\_avr\\_\\_interrupts.html](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/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:

- `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 ISR(vector_id, ISR_[BLOCK|NOBLOCK]) {
2     /* Hardware auto-clears the interrupt cause */
3     /* Clear the cause of the interrupt */
4     /* ISR-specific processing */
5 }
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/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:



```
1 ISR(PCINT0_vect) ? ▲  
2 {  
3     // Code to handle the event  
4 } ISR(PCINT1_vect, ISR_ALIASOF(▼  
5 ◀ ▶
```

## 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:



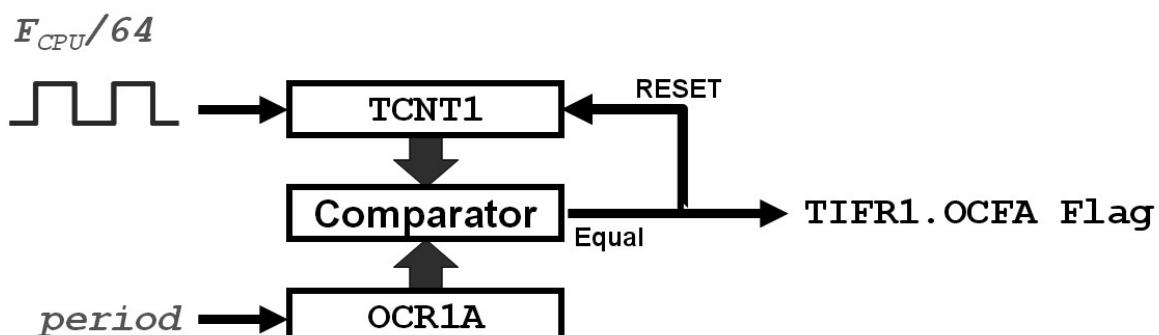
```

1 #include <avr/io.h> ▲
2 #include <avr/interruption.h>
3
4 ISR(TIMER1_COMPA_vect)
5 {
6     PORTB ^= (1 << PC0);
7 }
8
9 int main(void)
10 {
11     // Initialization
12
13     // Set LED as output
14     DDRB |= (1 << POF0);
15     PORTB &= ~(1 << POF0);
16
17     // Set up Timer/Counter1
18     TCCR1B |= (1 << WGM12) | (1 << CS10);
19     OCR1A = 25000;
20
21     TIMSK1 |= (1 << OCIE1A);
22     TCCR1B |= ((1 << COM1A1) | (1 << COM1A0));
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/interrupts.h>
3
4 ISR(TIMER1_COMPA_vect)
5 {
6     PORTB ^= (1 << PC0);
7 }
8
9 int main(void)
10 {
11     // Initialization
12
13     // Set LED as output
14     DDRB |= (1 << POF0);
15     PORTB &= ~(1 << POF0);
16
17     // Set up Timer/Counter 1
18     TCCR1B |= (1 << WGM12);
19     OCR1A = 25000;
20
21     TIMSK1 |= (1 << OCIE1A);
22     TCCR1B |= ((1 << CS10) | CS11);
23
24     // Enable all interrupts
25     sei();
26
27     while(1);
28 }
```



Este es un ejemplo de una interrupción no persistente ([/avr/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



# 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
 <a href="http://www.atmel.com/tools/MEGA328PB-XMINI.aspx">(http://www.atmel.com/tools/MEGA328PB-XMINI.aspx)</a>	Mini kit de evaluación ATmega328PB Xplained <a href="http://www.atmel.com/tools/MEGA328PB-XMINI.aspx">(http://www.atmel.com/tools/MEGA328PB-XMINI.aspx)</a> <a href="https://www.microchipdirect.co">https://www.microchipdirect.co</a>

### Herramientas de software

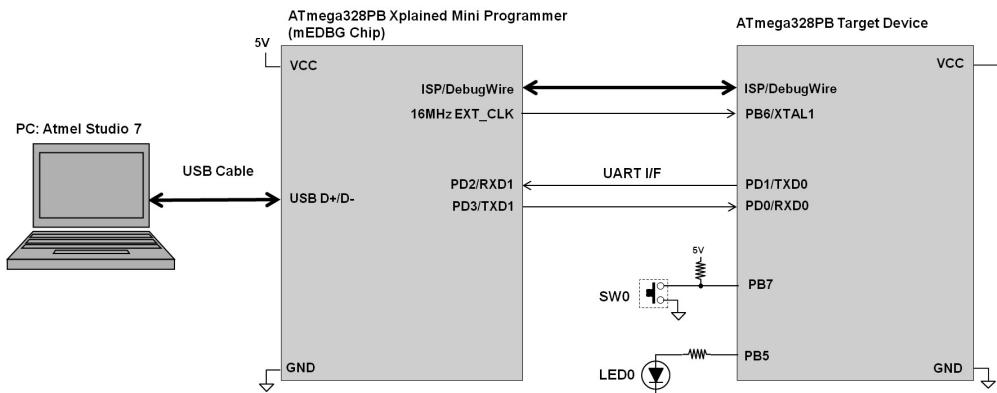
Herramienta	Sobre	Instaladores			Instrucciones de instalación
		Windows	Linux	Mac OS X	
Entorno de desarrollo integrado Atmel® Studio	<a href="#">Atmel® Studio</a>	<a href="#">(atstudio:start)</a>	<a href="#">(http://studio.download.atmel.com/7.0.1931/as-installer-7.0.1931-full.exe)</a>	<a href="#">(install:atstudi)</a>	<a href="#">(install:atstudi)</a>

### Archivos de ejercicios

	Descargar	
Expediente	Windows	Linux
Proyecto de ejemplo	<a href="#">(local--files/8avr:interrupts-mega-example/8avr-mega-int-usage.zip)</a>	<a href="#">(local--files/8avr:interrupts-mega-example/8avr-mega-int-usage)</a>

## 🔧 Diagrama de conexión

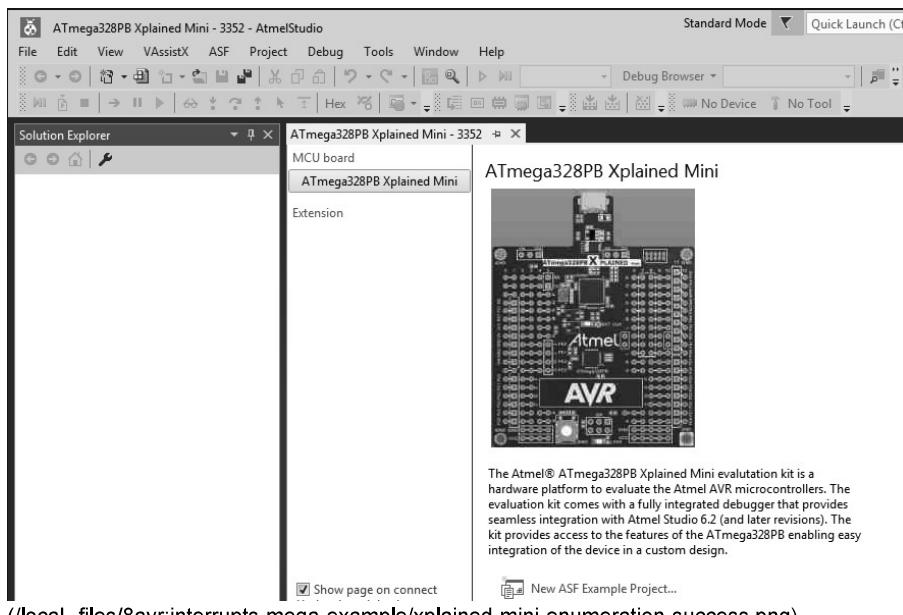
El módulo USART0 en el dispositivo ATmega328PB de destino está conectado a la interfaz USART en el chip mEDBG. El chip mEDBG realiza la conversión USB-serie enumerando como un puerto COM virtual de clase CDC en la PC y presentando los datos USART de destino en esta interfaz. El mEDBG también controla 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:

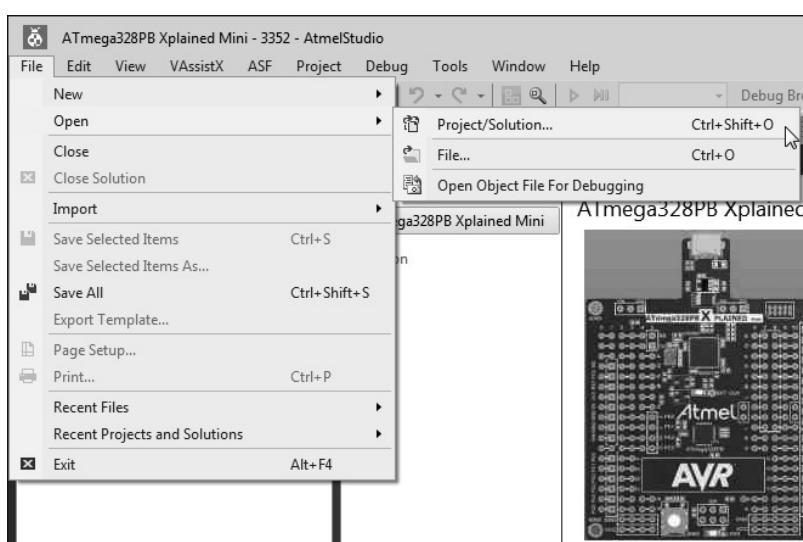


(/local--files/8avr:interrupts-mega-example/xplained-mini-enumeration-success.png)

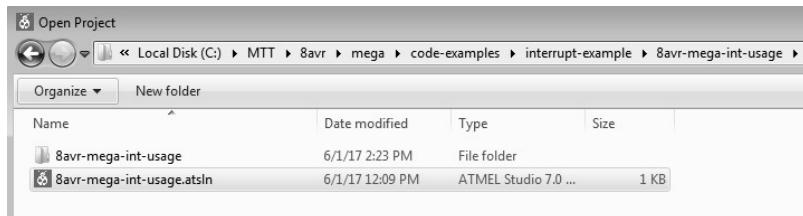


La placa se identifica por los últimos cuatro dígitos de su número de serie (consulte la etiqueta en la parte inferior de la placa). En el ejemplo anterior, los últimos cuatro dígitos son "3352"

### 1 Abra la solución



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

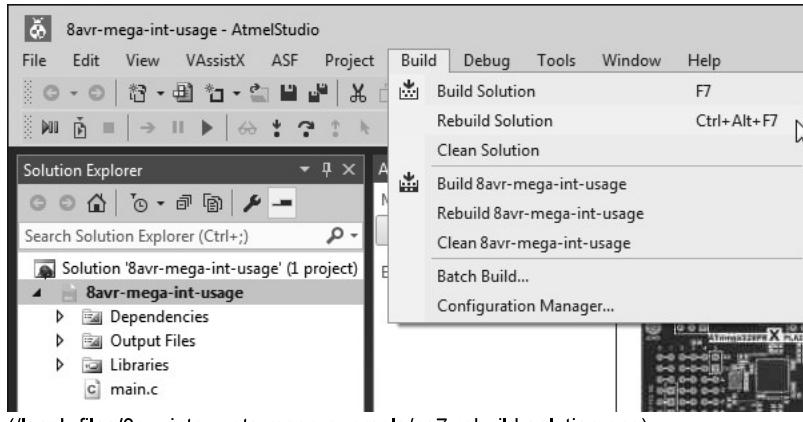


(/local--files/8avr:interrupts-mega-example/as7-open-solution-detail.png)



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



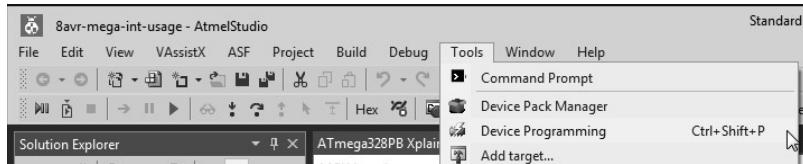
(/local--files/8avr:interrupts-mega-example/as7-rebuild-solution.png)

## 3 Programa los fusibles

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

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

Ingrese 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** :



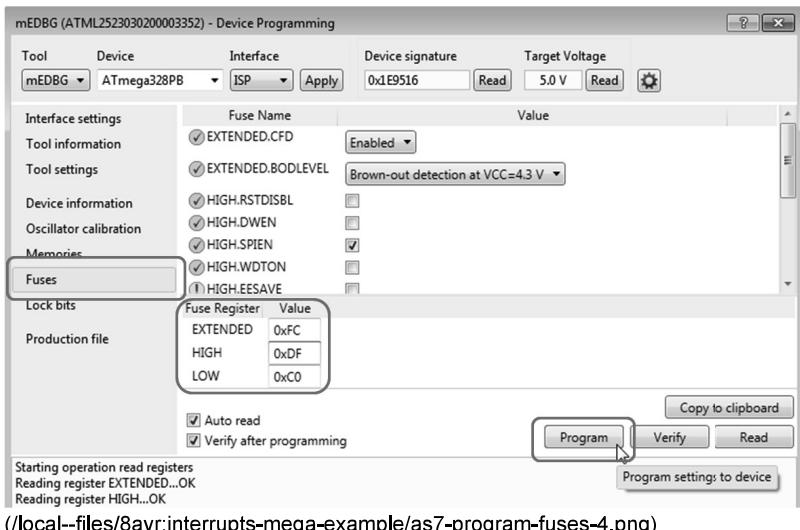
(/local--files/8avr:interrupts-mega-example/as7-program-fuses-2.png)

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)

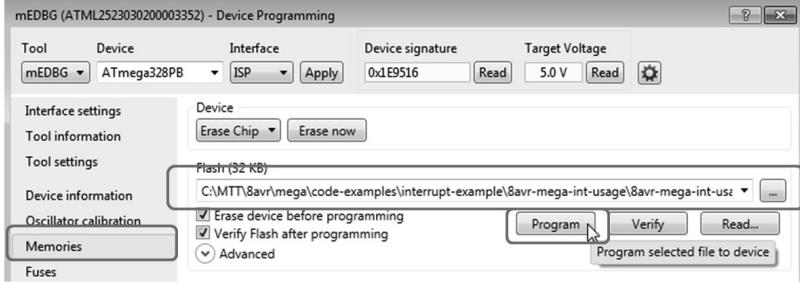
Seleccione la subsección **Fusibles** , ingrese los 3 valores de byte del fusible arriba, luego presione **Programar** como se muestra:



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

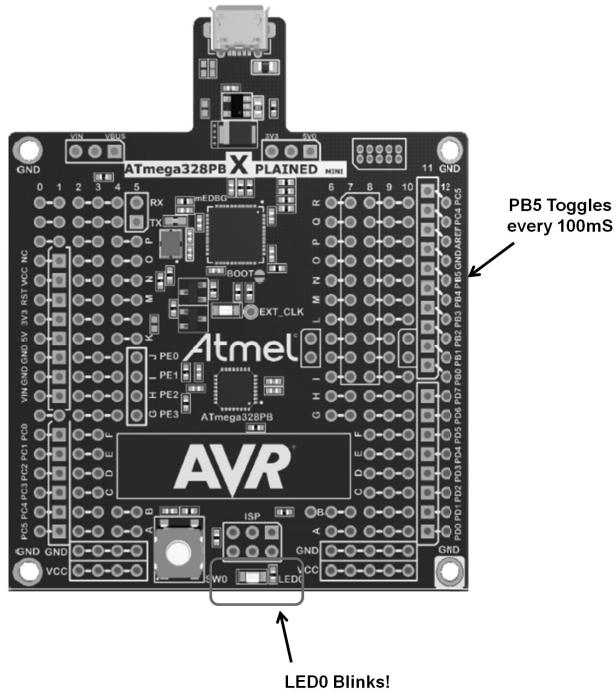
#### 4 Programa el archivo hexadecimal

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>
3 #include <avr/io.h><font>
4 #include <avr/interruptions.h>
5 <font></font>
6 volatile uint16_t ctr;
7 <font></font>
8 ISR(TIMER1_OVF_vect)<font>
9 {<font></font>
10   ctr--;<font></font>
11 }<font></font>
12 ...<font></font>
13 int<font></font>
14 main(void)<font></font><font>for(;;){<font></font>
15   ...<font></font>
16   ctr = 0x0200;<font>
17   start_timer();<font>
18   while(ctr != 0)<font>
19     // wait<font></font>
20     ;<font></font>
21     ...<font></font>
22   }<font></font>
23 }<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](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>
3 #include <avr/io.h><font>
4 #include <avr/interru&gt;
5 #include <util/atomic>
6 <font></font>
7 volatile uint16_t ctr;
8 <font></font>
9 ISR(TIMER1_OVF_vect)<font>
10 {<font></font>
11   ctr--;<font></font>
12 }<font></font>
13 ...<font></font>
14 int main(void)<font><
15 {<font></font>
diecisés      uint_16 ctr_copy;<
17   ...<font></font>
18   ctr = 0x0200;<font>
19   start_timer();<font>
20   do<font></font>
21   {<font></font>
22     ATOMIC_BLOCK(ATOMIC_WAIT,<font>
23     {<font></font>
24       ctr_copy = ctr;<font>
25     }<font></font>
26   } while(ctr != 0);<font>
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)

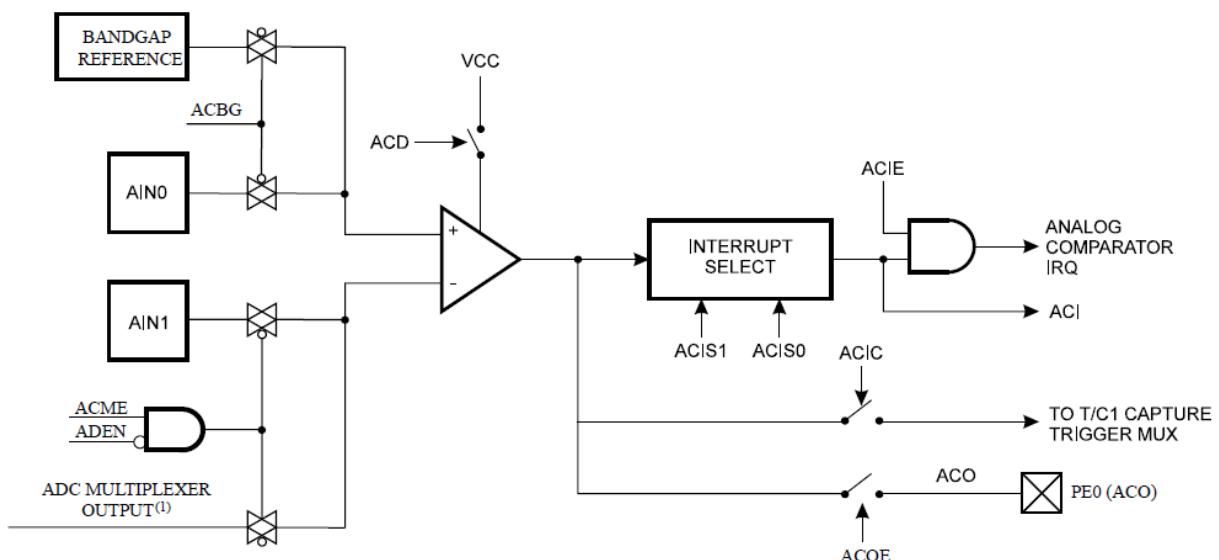


### Configuración de interrupciones megaAVR®

# Descripción general de la referencia de voltaje del comparador analógico

El comparador analógico compara los valores de entrada en el pin positivo AIN0 y el pin negativo AIN1. Cuando el voltaje en el pin positivo, AIN0, es mayor que el voltaje en el pin negativo, AIN1, se establece la salida del comparador analógico, ACO (en el puerto E[0]). La salida del comparador se puede configurar para activar la función de captura de entrada del temporizador/contador1. Además, el comparador puede disparar una interrupción separada, exclusiva del comparador analógico. El usuario puede seleccionar la activación de interrupción en el aumento, la caída o la alternancia de la salida del comparador.

Se muestra un diagrama de bloques del comparador y su lógica circundante.



(/local--files/8avr:analog-comparator-voltage-reference/comparator-logic.png)

El bit del convertidor de analógico a digital (ADC) de reducción de potencia en el registro de reducción de potencia (PRR.PRADC) debe escribirse en 0 para poder usar la entrada MUX de ADC.

## Entrada multiplexada del comparador analógico

Es posible seleccionar cualquiera de los pines ADC[7..0] para reemplazar la entrada negativa al comparador analógico. El multiplexor ADC se utiliza para seleccionar esta entrada y, en consecuencia, el ADC debe apagarse para utilizar esta función. Si el bit de habilitación del multiplexor del comparador analógico en el registro B de control y estado del ADC (ADCSRB.ACME) es uno y el ADC está apagado (ADCSRA.ADEN=0), los tres bits de selección de canal analógico menos significativos en el registro de selección del multiplexor del ADC (ADMUX.MUX[2..0]) seleccione el pin de entrada para reemplazar la entrada negativa al comparador analógico, como se muestra en la siguiente tabla. Cuando ADCSRB.ACME=0 o ADCSRA.ADEN=1, AIN1 se aplica a la entrada negativa del comparador analógico.

**Table 28-1 Analog Comparator Multiplexed Input**

ACME	ADEN	MUX[2..0]	Analog Comparator Negative Input
0	x	xxx	AIN1
1	1	xxx	AIN1
1	0	000	ADC0
1	0	001	ADC1
1	0	010	ADC2
1	0	011	ADC3
1	0	100	ADC4
1	0	101	ADC5
1	0	110	ADC6
1	0	111	ADC7

(/local--files/8avr:analog-comparator-voltage-reference/analog-comparator-mux-input.png)

## Control de comparador analógico y registro de estado B

El registro de control y estado de la memoria del programa de almacenamiento contiene los bits de control necesarios para controlar las operaciones del cargador de arranque. Al direccionar registros de E/S como espacio de datos usando instrucciones LD y ST, se debe usar el desplazamiento proporcionado. Cuando se utilizan los comandos IN y OUT específicos de E/S, el desplazamiento se reduce en 0x20, lo que da como resultado un desplazamiento de la dirección de E/S entre 0x00 y 0x3F .

Nombre: ACSRB Compensación: 0x4F Restablecimiento: 0x00

Propiedad:

Cuando se direcciona como registro de E/S: la compensación de dirección es 0x2F

Bit	7	6	5	4	3	2	1	0
Access								ACOE
Reset								0

(/local--files/8avr:analog-comparator-voltage-reference/input-register.png)

- Bit 0 – ACOE: Habilitación de salida de comparador analógico

Cuando se establece este bit, la salida del comparador analógico se conecta al pin ACO.

## Registro de control y estado del comparador analógico

Al direccionar registros de E/S como espacio de datos usando instrucciones LD y ST, se debe usar el desplazamiento proporcionado. Cuando se utilizan los comandos IN y OUT específicos de E/S, el desplazamiento se reduce en *0x20*, lo que da como resultado un desplazamiento de la dirección de E/S entre *0x00* y *0x3F*.

Nombre: ACSR Compensación: *0x50* Restablecimiento: N/A

Propiedad:

Cuando se direcciona como registro de E/S: la compensación de dirección es *0x30*

Bit	7	6	5	4	3	2	1	0
Access	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACISO
Reset	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W

(/local--files/8avr:analog-comparator-voltage-reference/status-register.png)

- Bit 7 – ACD: Deshabilitar comparador analógico

Cuando este bit se escribe como uno lógico, se desconecta la alimentación del comparador analógico. Este bit se puede configurar en cualquier momento para apagar el comparador analógico. Esto reducirá el consumo de energía en los modos Activo e Inactivo. Al cambiar el bit ACD, la interrupción del comparador analógico debe desactivarse borrando el bit ACIE en ACSR. De lo contrario, puede ocurrir una interrupción cuando se cambia el bit.

- Bit 6 – ACBG: Selección de intervalo de banda del comparador analógico

Cuando se establece este bit, un voltaje de referencia de banda fija reemplaza la entrada positiva al comparador analógico. Cuando se borra este bit, se aplica AIN0 a la entrada positiva del comparador analógico. Cuando la referencia de banda prohibida se utiliza como entrada al comparador analógico, el voltaje tardará cierto tiempo en estabilizarse. Si no se estabiliza, la primera conversión puede dar un valor incorrecto.

- Bit 5 – ACO: Salida del comparador analógico

La salida del comparador analógico se sincroniza y luego se conecta directamente a ACO. La sincronización introduce un retraso de uno a dos ciclos de reloj.

- Bit 4 – ACI: Indicador de interrupción del comparador analógico

Este bit lo establece el hardware cuando un evento de salida del comparador activa el modo de interrupción definido por ACIS1 y ACIS0. La rutina ACI se ejecuta si se establece el bit ACIE y se establece el bit I en SREG. El hardware borra ACI cuando se ejecuta el vector de manejo de interrupción correspondiente. Alternativamente, ACI se borra escribiendo un uno lógico en la bandera.

- Bit 3 – ACIE: Habilitación de interrupción del comparador analógico

Cuando el bit ACIE se escribe uno lógico y se establece el bit I en el registro de estado, se activa la interrupción del comparador analógico. Cuando se escribe cero lógico, la interrupción se deshabilita.

- Bit 2 – ACIC: Habilitación de captura de entrada de comparador analógico

Cuando se escribe como uno lógico, este bit permite que el comparador analógico active la función de captura de entrada en el temporizador/contador 1. En este caso, la salida del comparador está directamente conectada a la lógica frontal de captura de entrada, lo que hace que el comparador utilice las funciones de cancelación de ruido y selección de borde de la interrupción de captura de entrada del temporizador/contador1. Cuando se escribe cero lógico, no existe conexión entre el comparador analógico y la función de captura de entrada. Para hacer que el comparador dispare la interrupción de captura de entrada del temporizador/contador 1, se debe establecer el bit ICIE1 en el registro de máscara de interrupción del temporizador (TIMSK1).

- Bits 1:0 – ACISn: selección del modo de interrupción del comparador analógico [n = 1:0]

Estos bits determinan qué eventos del comparador activan la interrupción del comparador analógico.

<b>ACIS1</b>	<b>ACIS0</b>	<b>Interrupt Mode</b>
0	0	Comparator Interrupt on Output Toggle.
0	1	Reserved
1	0	Comparator Interrupt on Falling Output Edge.
1	1	Comparator Interrupt on Rising Output Edge.

(/local--files/8avr:analog-comparator-voltage-reference/comparator-interrupt.png)

Al cambiar los bits ACIS1/ACIS0, la interrupción del comparador analógico debe desactivarse borrando su bit de habilitación de interrupción en el registro ACSR. De lo contrario, puede ocurrir una interrupción cuando se cambian los bits.

## Registro de desactivación de entrada digital 1

Nombre: DIDR1 Compensación: 0x7F ; Restablecer: 0x00

Propiedad:

Bit	7	6	5	4	3	2	1	0
Access							AIN1D	AIN0D
Reset							R/W	R/W

(/local--files/8avr:analog-comparator-voltage-reference/digital-input-disable.png)

- Bit 1 – AIN1D: Desactivación de entrada digital AIN1
- Bit 0 – AIN0D: AIN0 Inhabilitación de entrada digital

Cuando este bit se escribe uno lógico, el búfer de entrada digital en el pin AIN1/0 está deshabilitado. El bit de registro de PIN correspondiente siempre se leerá como cero cuando se establezca este bit. Cuando se aplica una señal analógica al pin AIN1/0 y no se necesita la entrada digital de este pin, este bit debe escribirse uno lógico para reducir el consumo de energía en el búfer de entrada digital.

Especificación de voltaje de referencia ADC de 1,1 V seleccionable:

Symbol	Parameter	Condition	Min.	Typ	Max	Units
V <sub>POT</sub>	Power-on Reset Threshold Voltage (rising)		1.1	1.5	1.7	V
	Power-on Reset Threshold Voltage (falling) <sup>(2)</sup>		0.6	1.0	1.7	V
S <sub>RON</sub>	Power-on Slope Rate		0.01	-	10	V/ms
V <sub>RST</sub>	RESET Pin Threshold Voltage		0.2 V <sub>CC</sub>	-	0.9 V <sub>CC</sub>	V
t <sub>RST</sub>	Minimum pulse width on RESET Pin		-	-	2.5	μs
V <sub>HYST</sub>	Brown-out Detector Hysteresis		-	50	-	mV
t <sub>BOD</sub>	Min. Pulse Width on Brown-out Reset		-	2	-	μs
V <sub>BG</sub>	Bandgap reference voltage	V <sub>CC</sub> =2.7 T <sub>A</sub> =25°C	1.0	1.1	1.2	V
t <sub>BG</sub>	Bandgap reference start-up time	V <sub>CC</sub> =2.7 T <sub>A</sub> =25°C	-	40	70	μs
I <sub>BG</sub>	Bandgap reference current consumption	V <sub>CC</sub> =2.7 T <sub>A</sub> =25°C	-	10	-	μA

(/local--files/8avr:analog-comparator-voltage-reference/adc-reference.png)

## Proyecto de ejemplo

Un ejemplo de proyecto para el comparador analógico está disponible aquí: Proyecto de ejemplo (/8avr:analog-comparator-voltage-reference)

# Comparador analógico y ejemplo de referencia de voltaje

Este proyecto práctico demuestra un ejemplo simple de comparador analógico y voltaje de referencia: referencia de banda prohibida como entrada positiva (1,1 V) y AIN1 como entrada negativa. Al cablear AIN1 (PD7) al interruptor (SW0, PB7), que se enciende en alto (5 V) y se apaga a GND, y al cablear ACO (PE0) a PB5 (LEDO), la salida del comparador (ACO, PE0) se puede observar comprobando el estado de los LED.

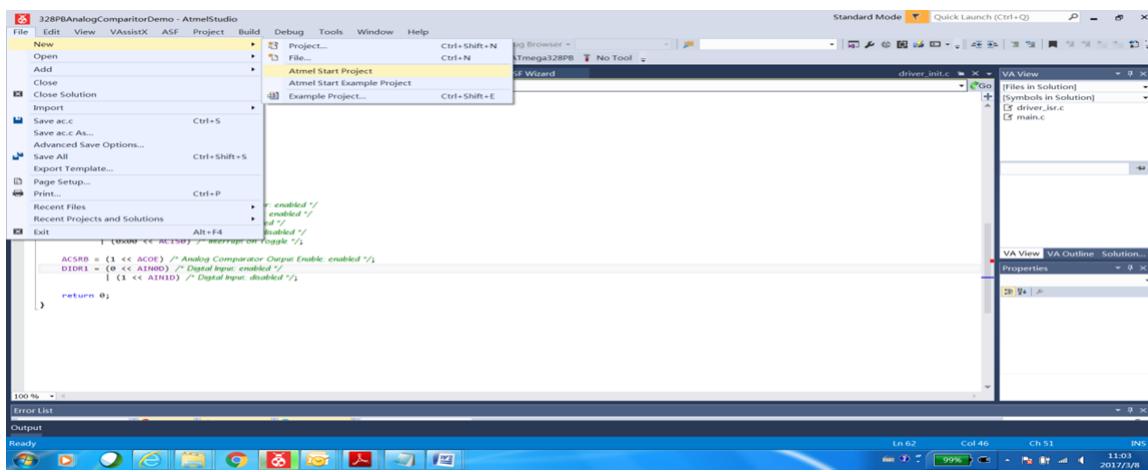
- Cableado : PD7 (AIN1) | PB7 (SW0; PE0 (ACO) | PB5 (LEDO)

Tablero explicado:



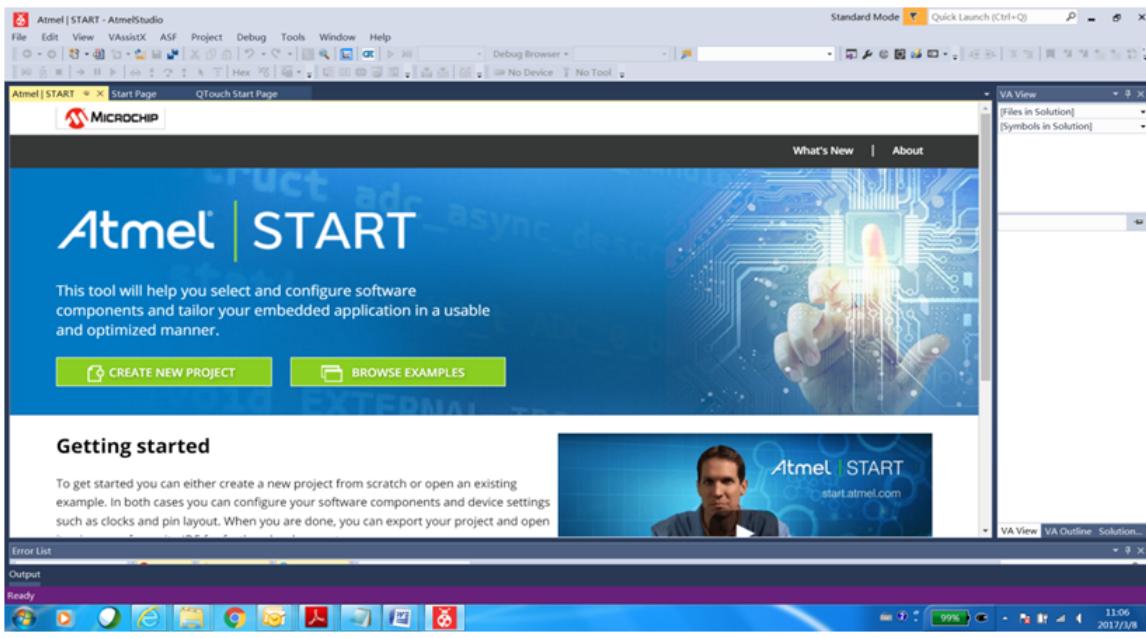
(/local--files/8avr:analog-comparator-voltage-reference/xplained-board.png)

## 1 Nuevo | Proyecto de inicio de Atmel



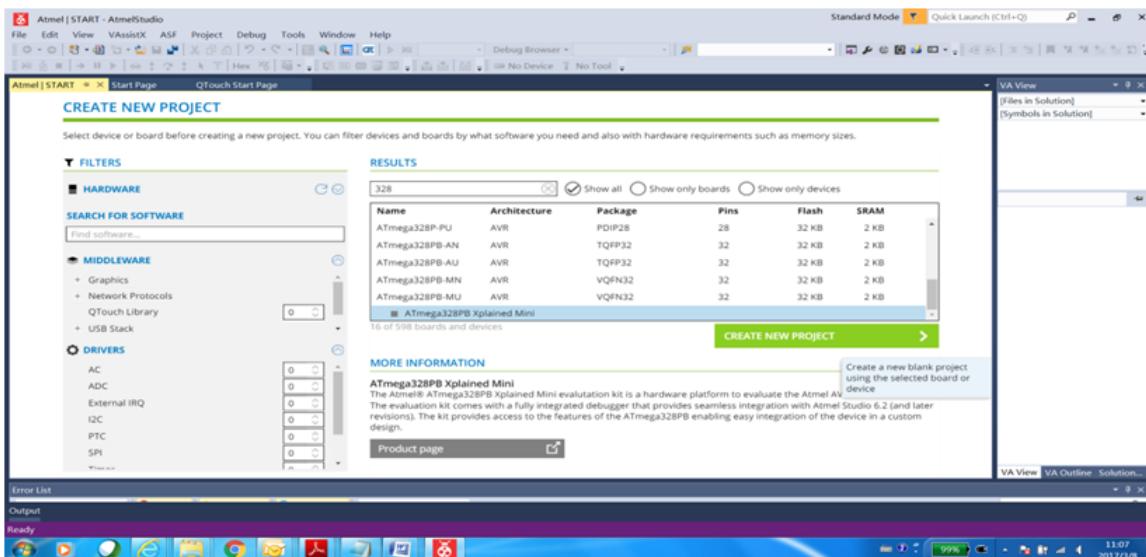
(/local--files/8avr:analog-comparator-voltage-reference/atmel-start-project.png)

## 2 Crear nuevo proyecto



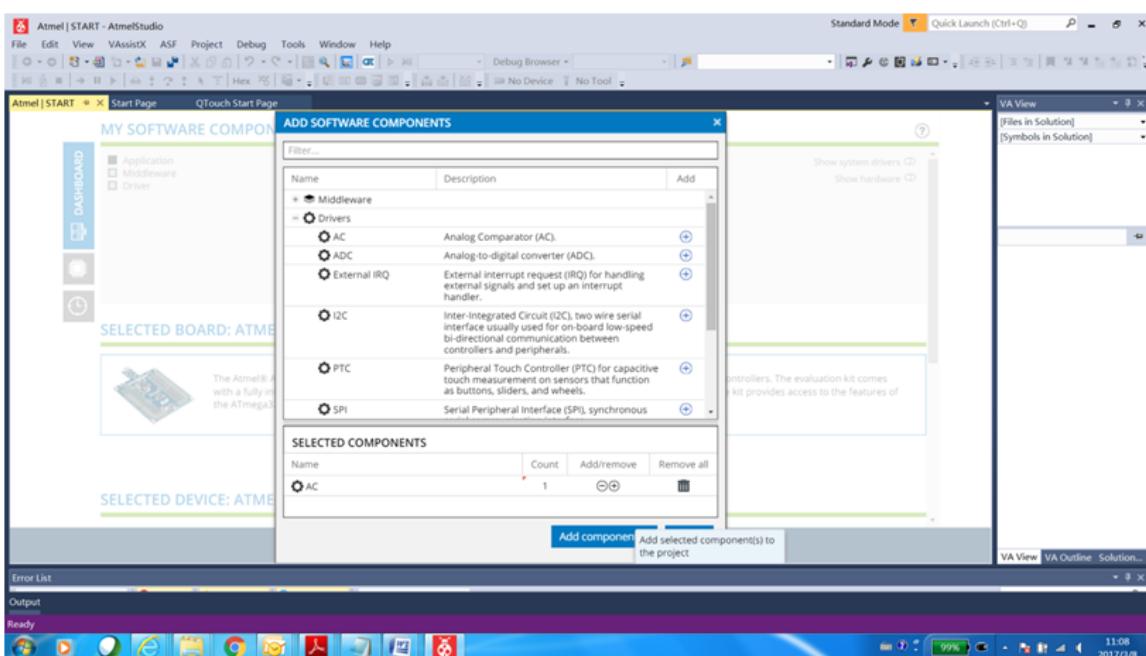
(/local--files/8avr:analog-comparator-voltage-reference/atmel-create-project.png)

### 3 Seleccione ATmega328PB Mini explicado (/boards:atavr328) :



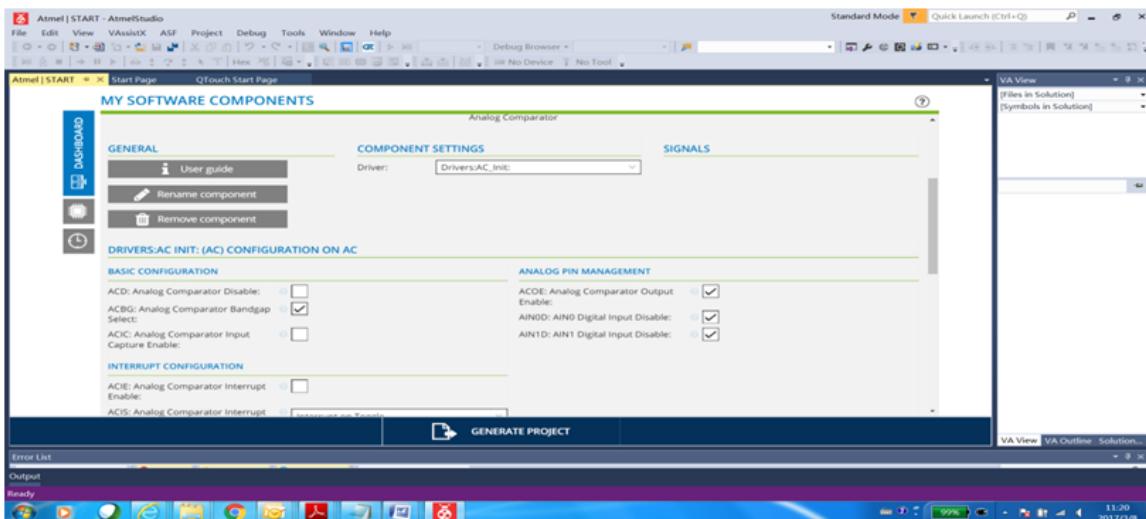
(/local--files/8avr:analog-comparator-voltage-reference/atmel-create-project-2.png)

### 4 Agregue componentes de software, seleccione AC:



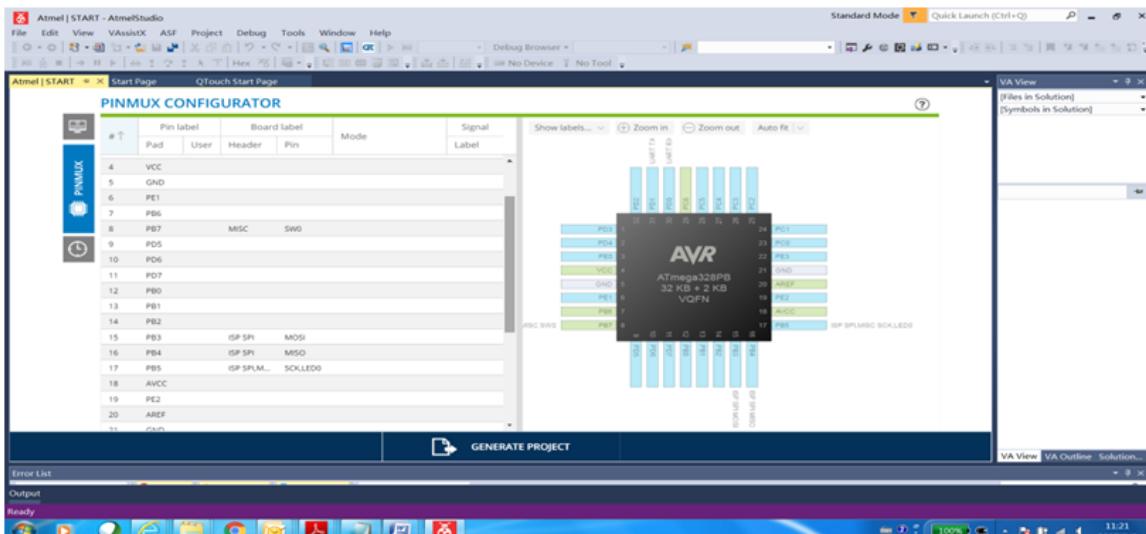
(/local--files/8avr:analog-comparator-voltage-reference/atmel-create-project-3.png)

## 5 Configuración de CA:



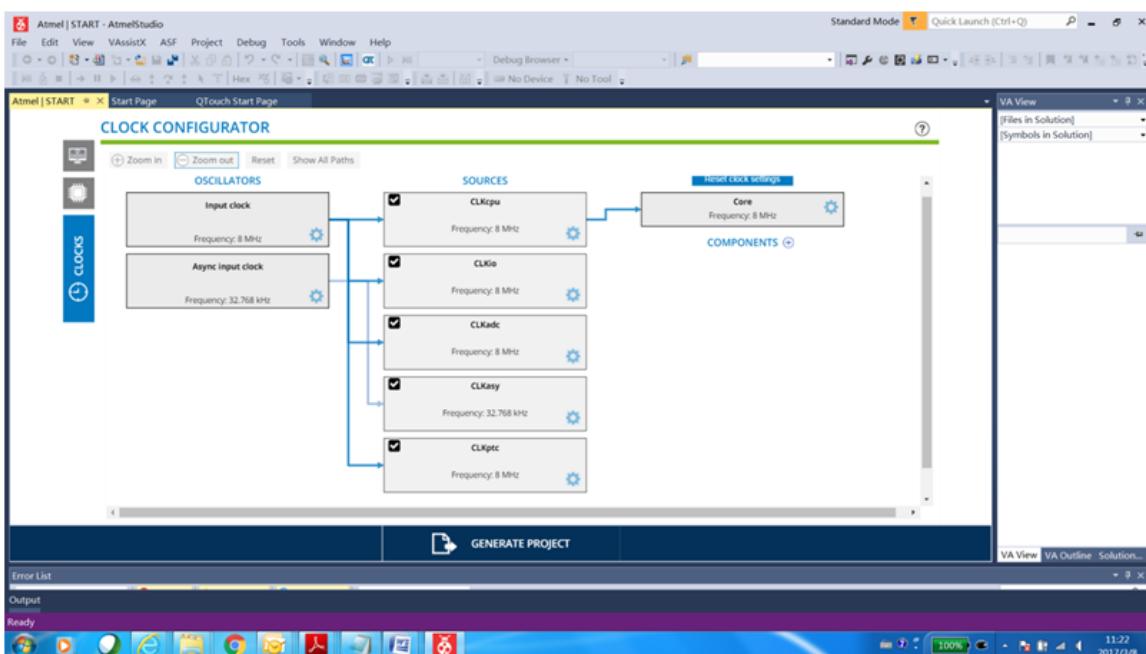
(/local--files/8avr:analog-comparator-voltage-reference/atmel-create-project-4.png)

## 6 Vista del configurador de pines:



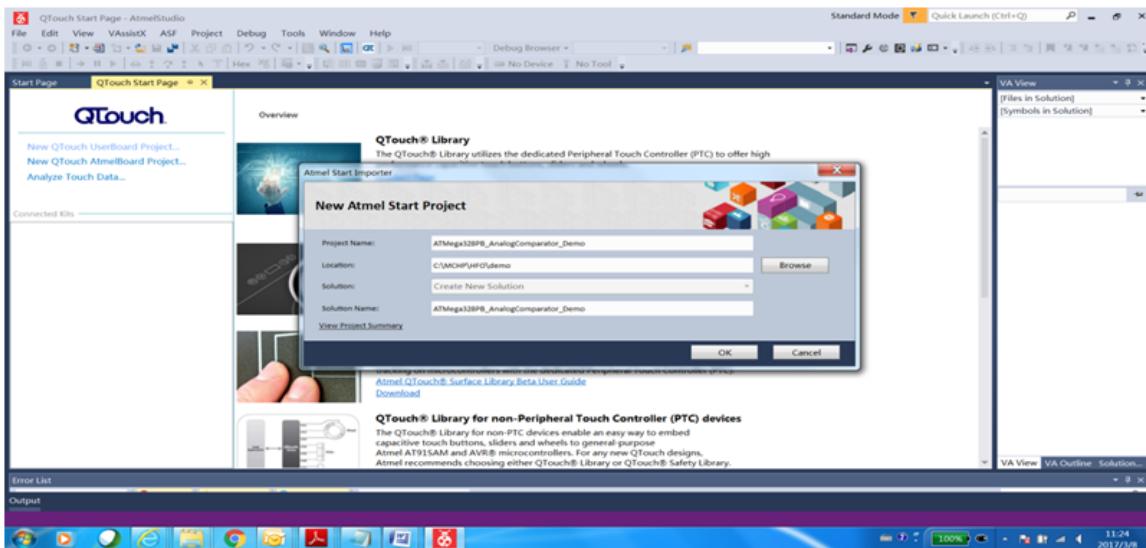
(/local--files/8avr:analog-comparator-voltage-reference/atmel-create-project-5.png)

## 7 Configuración del configurador de reloj:



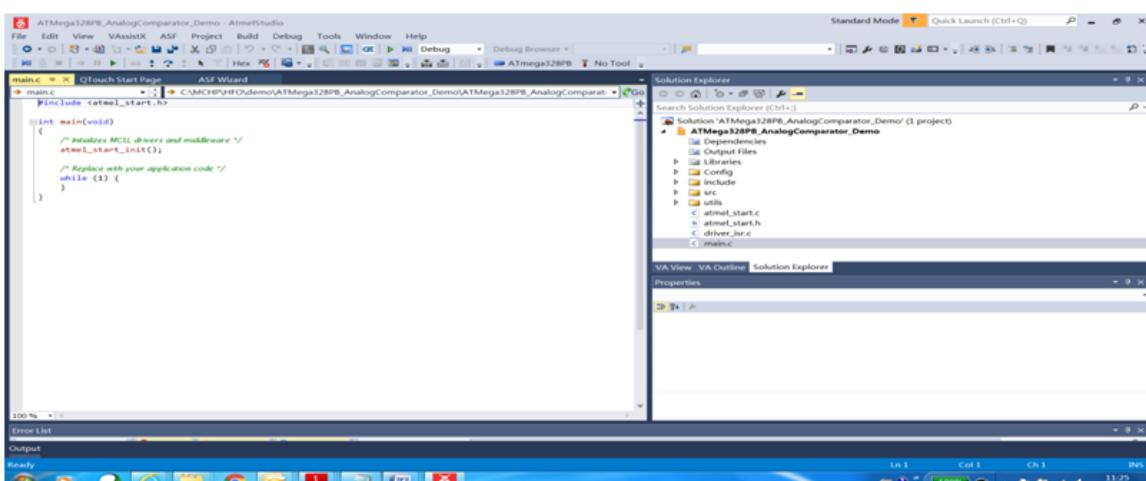
(/local--files/8avr:analog-comparator-voltage-reference/atmel-create-project-6.png)

## 8 Generar Proyecto:



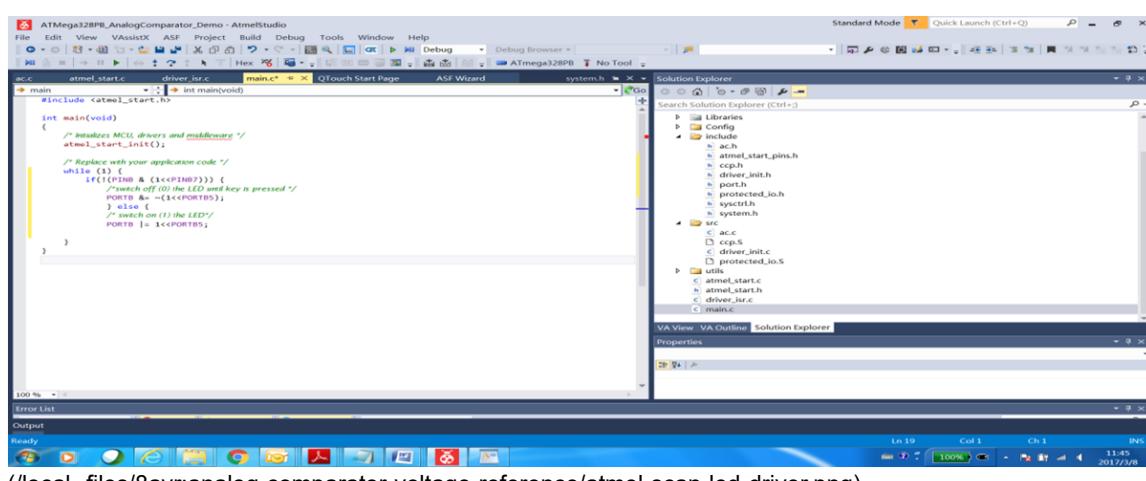
(/local--files/8avr:analog-comparator-voltage-reference/atmel-generate-project.png)

## 9 Ver Proyecto:



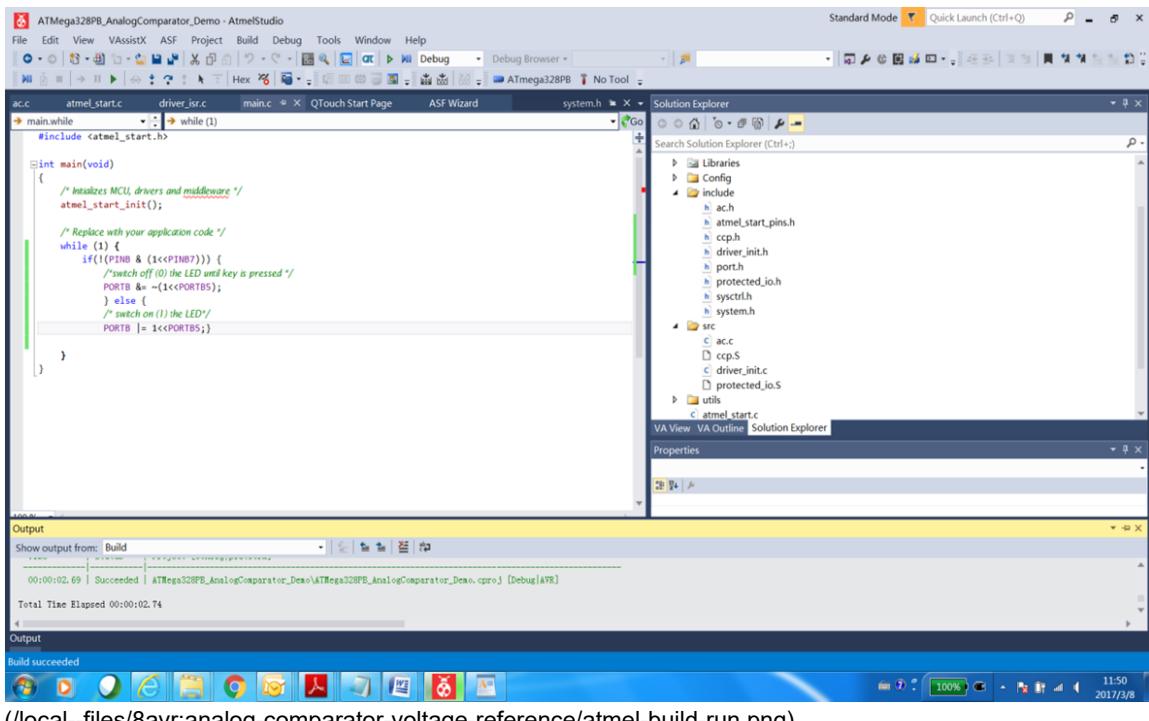
(/local--files/8avr:analog-comparator-voltage-reference/atmel-view-project.png)

## 10 En `Main.c` , agregue escaneo de teclado y controlador LED:



(/local--files/8avr:analog-comparator-voltage-reference/atmel-scan-led-driver.png)

## 11 Construir y ejecutar:



(/local--files/8avr:analog-comparator-voltage-reference/atmel-build-run.png)

Para descargar la demostración del comparador analógico ATMega328PB, visite esta [página](#) (<https://microchiptechnology.sharepoint.com/:u/s/DeveloperHelp/EUoeisQUGDNAqXHmQDJm4WgB5lendH3ses4oRBL9zPbrpw?e=AmjbWH>).

# Descripción general del temporizador AVR® MCU

Los temporizadores son una característica muy útil de un microcontrolador para contar pulsos en un pin de entrada. Cuando son impulsados por el reloj de instrucciones, pueden convertirse en una base de tiempo precisa. Los dispositivos AVR® tienen temporizadores de 8 y 16 bits de ancho y ofrecen diferentes funciones según el dispositivo. Un conjunto muy típico de temporizadores se puede encontrar en el microcontrolador AVR **ATmega328PB**. Este dispositivo tiene cinco temporizadores/contadores, como se describe aquí:

Temporizador 0	TC0	Temporizador/contador de 8 bits con modulación de ancho de pulso (PWM)
Temporizador 1	TC1	Temporizador/contador de 16 bits con PWM y funcionamiento asíncrono
Temporizador 2	TC2	Temporizador/contador de 8 bits con PWM y funcionamiento asíncrono
Temporizador 3	TC3	Temporizador/contador de 16 bits con PWM y funcionamiento asíncrono
Temporizador 4	TC4	Temporizador/contador de 16 bits con PWM y funcionamiento asíncrono

## Definiciones:

### Nomenclatura de registro

ABAJO	El contador llega al FONDO cuando llega a cero (0x00 para contadores de 8 bits y 0x0000 para contadores de 16 bits)
MÁX.	El contador alcanza su valor máximo cuando pasa a ser 0x0F (15 decimal) para contadores de 8 bits y 0x00FF (255 decimal) para contadores de 16 bits
PARTE SUPERIOR	El contador llega al TOP cuando su valor llega a ser igual al valor más alto posible. Al valor TOP se le puede asignar un valor fijo MAX o el valor almacenado en el registro OCRxA. Esta asignación depende del modo de funcionamiento

# TC0 - Temporizador/Contador de 8 bits con PWM

Timer/Counter0 (TC0) es un módulo de temporizador/contador de uso general de 8 bits, con dos unidades de comparación de salida independientes y compatibilidad con PWM.

## Registros TC0

El registro del temporizador/contador 0 (TCNT0) y los registros de comparación de salida TC0x (OCR0x) son registros de 8 bits. Todas las señales de solicitud de interrupción son visibles en el registro de bandera de interrupción del temporizador 0 (TIFR0). Todas las interrupciones se enmascaran individualmente con el registro de máscara de interrupción del temporizador 0 (TIMSK0).

**Name:** TCCR0B

**Offset:** 0x45

**Reset:** 0x00

**Property:** When addressing as I/O Register: address offset is 0x25

Bit	7	6	5	4	3	2	1	0
	FOC0A	FOC0B			WGM02	CS02	CS01	CS00
Access	R/W	R/W			R/W	R/W	R/W	R/W
Reset	0	0			0	0	0	0

(/local--files/8avr:avrtimer/TCCR0B.PNG)

## Fuentes de reloj del temporizador/contador TC0

TC0 can be clocked by an internal or an external clock source. The clock source is selected by writing to the Clock Select (CS02:0) bits in the Timer/Counter Control Register (TCCR0B).

### Bits 2:0 – CS0n: Clock Select [n = 0..2]

The three Clock Select bits select the clock source to be used by the Timer/Counter.

CS02	CS01	CS00	Description
0	0	0	No Clock source (Timer stopped)
0	0	1	clkio/1 (No prescaling)
0	1	0	clkio/8 (From prescaler)
0	1	1	clkio/64 (From prescaler)
1	0	0	clkio/256 (From prescaler)
1	0	1	clkio/1012 (From prescaler)

1	1	0	External clock source on T0 pin (clock on falling edge)
1	1	1	External clock source on T0 pin (clock on rising edge)

## TC0 Counter Unit

Depending on the mode of operation used, T0 is cleared, incremented, or decremented at each timer clock (clkT0). clkT0 can be generated from an external or internal clock source, selected by the Clock Select bits (CS0[2:0]).

The counting sequence is determined by the setting of the WGM01 and WGM00 bits located in the T0 Control Register A (TCCR0A) and the WGM02 bit located in the Timer/Counter Control Register B (TCCR0B).

### Bits 1:0 – WGM0n: Waveform Generation Mode [n = 1:0]

Combined with the WGM02 bit found in the TCCR0B Register, these bits control the counting sequence of the counter, the source for maximum (TOP) counter value and what type of waveform generation to be used. Modes of operation supported by the Timer/Counter unit are: Normal mode (counter), Clear Timer on Compare Match (CTC) Mode and two types of Pulse Width Modulation (PWM) modes.

Table 1-2 Waveform Generation Mode Bit Description

Mode	WGM2:0	Mode of Operation	TOP	OCR0x Update	TOV flag set on
0	0 0 0	Normal	0xFF	Immediate	MAX
1	0 0 1	PWM Phase Correct	0xFF	TOP	BOTTOM
2	0 1 0	CTC	OCRA	Immediate	MAX
3	0 1 1	Fast PWM	0xFF	BOTTOM	MAX
4	1 0 0	Reserved	~	~	~
5	1 0 1	PWM Phase Correct	OCRA	TOP	BOTTOM
6	1 1 0	Reserved	~	~	~
7	1 1 1	Fast PWM	OCRA	BOTTOM	MAX

Note:

1. MAX = 0xFF
2. BOTTOM = 0x00

## Modes of Operation for TC0

The mode of operation determines the behavior of TC0 and the Output Compare pins. It is defined by the combination of the Waveform Generation mode bits and Compare Output mode bits in the Timer/Counter control Registers A and B (TCCR0B.WGMn2, TCCR0A.WGM01, TCCR0A.WGM00 and TCCR0A.COM0x[1:0]).

Available modes of operation for TC0 are:

- Normal Mode
- Clear Timer on Compare Match (CTC) Mode
- Fast PWM Mode
- Phase Correct PWM Mode

### Clear Timer on Compare Match Mode

In Clear Timer on Compare or CTC mode (WGM0[2:0]=0x2), the **OCR0A Register** is used to manipulate the counter resolution: the counter is cleared to ZERO when the counter value (TCNT0) matches the OCR0A. The OCR0A defines the top value for the counter, hence also its resolution.

The counter value (TCNT0) increases until a compare match occurs between TCNT0 and OCR0A and then counter (TCNT0) is cleared. An interrupt can be generated each time the counter value reaches the TOP value by setting the OCF0A Flag. If the interrupt is enabled, the interrupt handler routine can be used for updating the TOP value.

The waveform frequency is defined by the following equation:

$$f_{OCnx} = \frac{f_{clk\_I/O}}{2 \cdot N \cdot (1 + OCRnx)}$$

(/local--files/8avr:avrtimer/freqz0.PNG)

N represents the prescaler factor (1, 8, 64, 256, or 1024).

## TC1, TC3, & TC4 - 16-bit Timer/Counters with PWM

The 16-bit Timer/Counter units allow accurate program execution timing (event management), wave generation and signal timing measurement.

### Registers (TC1, TC3, TC4)

- The Timer/Counter (TCNT<sub>n</sub>), Output Compare Registers (OCRA/B) and Input Capture Register (ICR<sub>n</sub>) are all 16-bit registers.
- The Timer/Counter Control Registers (TCCRnA/B) are 8-bit registers and have no CPU access restrictions.
- Interrupt requests (abbreviated to Int.Req. in the block diagram) signals are all visible in the Timer Interrupt Flag Register (TIFR<sub>n</sub>). All interrupts are individually masked with the Timer Interrupt Mask Register (TIMSK<sub>n</sub>).

## Timer/Counter Clock Sources (TC1, TC3, TC4)

The Timer/Counter can be clocked by an internal or an external clock source. The clock source is selected by the Clock Select logic which is controlled by the Clock Select bits in the timer/Counter control Register B (TCCRnB.CS[2:0]).

**Name:** TCCR1B, TCCR3B, TCCR4B

**Offset:** 0x81 + n\*0x10 [n=0..2]

**Reset:** 0x00

**Property:** -

Bit	7	6	5	4	3	2	1	0
Access	ICNC	ICES		WGM3	WGM2		CS[2:0]	
Reset	R/W	R/W		R/W	R/W	R/W	R/W	R/W
	0	0		0	0	0	0	0

(/local--files/8avr:avrtimer/TCCR1B.PNG)

Bits 2:0 – CS[2:0]: Clock Select [n = 0..2]

The three Clock Select bits select the clock source to be used by the Timer/Counter.

CS02	CS01	CS00	Description
0	0	0	No Clock source (Timer stopped)
0	0	1	clkio/1 (No prescaling)
0	1	0	clkio/8 (From prescaler)
0	1	1	clkio/64 (From prescaler)
1	0	0	clkio/256 (From prescaler)
1	0	1	clkio/1012 (From prescaler)
1	1	0	External clock source on T0 pin (clock on falling edge)
1	1	1	External clock source on T0 pin (clock on rising edge)

## Counter Unit (TC1, TC3, TC4)

TC1, TC3 and TC4 are programmable 16-bit bi-directional counters.

Each 16-bit counter is mapped into two 8-bit I/O memory locations: **Counter High** (TCNTnH) containing the upper eight bits of the counter and **Counter Low** (TCNTnL) containing the lower eight bits.

Depending on the selected mode of operation, the counter is cleared, incremented, or decremented at each timer clock (clkTn). The clock clkTn can be generated from an external or internal clock source, as selected by the Clock Select bits in the Timer/Counter Control Register B (TCCRnB.CS[2:0]).

The counting sequence is determined by the setting of the Waveform Generation mode bits in the Timer/Counter Control Registers A and B (TCCRnB.WGM[3:2] and TCCRnA.WGM[1:0]).

### Modes of Operation (TC1, TC3, TC4)

The mode of operation is determined by the combination of the Waveform Generation mode (WGM[3:0]) and Compare Output mode (TCCRnA.COMx[1:0]) bits.

Available modes of operation are:

- Normal Mode
- Clear Timer on Compare Match (CTC) Mode
- Fast PWM Mode
- Phase Correct PWM Mode
- Phase and Frequency Correct PWM Mode

### Fast PWM Mode

The Fast Pulse Width Modulation or Fast PWM modes (modes 5, 6, 7, 14, and 15, WGM[3:0]= 0x5, 0x6, 0x7, 0xE, 0xF) provide a high frequency PWM waveform generation option. The Fast PWM differs from the other PWM options by its single-slope operation. The counter counts from BOTTOM to TOP then restarts from BOTTOM.

In Fast PWM mode the counter is incremented until the counter value matches either one of the fixed values 0x00FF, 0x01FF, or 0x03FF (WGM[3:0] = 0x5, 0x6, or 0x7), the value in ICRn (WGM[3:0]=0xE), or the value in OCRnA (WGM[3:0]=0xF). The counter is then cleared at the following timer clock cycle.

The PWM frequency for the output can be calculated by the following equation:

$$f_{OCnxPWM} = \frac{f_{clk\_I/O}}{N \cdot (1 + TOP)}$$

(/local--files/8avr:avrtimer/freqz1.PNG)

**Note:**

- The “n” in the register and bit names indicates the device number ( $n = 0$  for Timer/Counter 0) and the “x” indicates Output Compare unit (A/B).
- $N$  represents the prescale divider (1, 8, 64, 256, or 1024).

## TC2 - 8-bit Timer/Counter2 with PWM and Asynchronous Operation

Timer/Counter2 (TC2) is a general purpose, dual channel, 8-bit Timer/Counter module.

### TC2 Registers

The Timer/Counter (TCNT2) and Output Compare Register (OCR2A and OCR2B) are 8-bit registers. Interrupt request signals are all visible in the Timer Interrupt Flag Register (TIFR2). All interrupts are individually masked with the Timer Interrupt Mask Register (TIMSK2).

**Name:** TCCR2B  
**Offset:** 0xB1  
**Reset:** 0x00  
**Property:** -

Bit	7	6	5	4	3	2	1	0
Access	FOC2A	FOC2B			WGM22	CS22	CS21	CS20
Reset	R/W	R/W			R/W	R/W	R/W	R/W

(/local--files/8avr:avrtimer/TCCR2B.PNG)

### TC2 Clock Sources

TC2 can be clocked by an internal synchronous or an external asynchronous clock source: The three Clock Select bits (CS2:CS0) select the clock source to be used by the Timer/Counter.

CS02	CS01	CS00	Description
0	0	0	No Clock source (Timer stopped)
0	0	1	clkio/1 (No prescaling)
0	1	0	clkio/8 (From prescaler)
0	1	1	clkio/64 (From prescaler)
1	0	0	clkio/256 (From prescaler)

1	0	1	clkio/1012 (From prescaler)
1	1	0	External clock source on T0 pin (clock on falling edge)
1	1	1	External clock source on T0 pin (clock on rising edge)

## TC2 Counter Unit

Depending on the mode of operation used, the counter is cleared, incremented, or decremented at each timer clock (clkT2). clkT2 can be generated from an external or internal clock source, selected by the Clock Select bits (CS2[2:0]).

The counting sequence is determined by the setting of the WGM21 and WGM20 bits located in the Timer/Counter Control Register (TCCR2A) and the WGM22 bit located in the Timer/Counter Control Register B (TCCR2B).

## TC2 Modes of Operation

The mode of operation, i.e., the behavior of the Timer/Counter and the Output Compare pins, is defined by the combination of the Waveform Generation mode (WGM2[2:0]) and Compare Output mode (COM2x[1:0]) bits.

Available modes of operation are:

- Normal Mode
- Clear Timer on Compare Match (CTC) Mode
- Fast PWM Mode
- Phase Correct PWM Mode

### Normal Mode

In the Normal mode (WGM22:0 = 0) the counting direction is always up (incrementing) without having the counter cleared. The counter will roll over to 0c00 when it passes its maximum 8-bit value (TOP = 0xFF).

In normal operation, the Timer/Counter Overflow Flag (TOV2) will be set in the same timer clock cycle as the TCNT2 becomes zero. The TOV2 Flag, in this case, behaves like a ninth bit, except that it is only set, not cleared. However, combined with the timer overflow interrupt that automatically clears the TOV2 flag, the timer resolution can be increased by software. There are no special cases to consider in the Normal mode, a new counter value can be written at any time.



Regardless of which mode is used the programmer needs to remember two things:

1. The timer has to be started by selecting the clock source.
2. If interrupts are used, they must be enabled.



# Ejemplo de temporizador MCU AVR® de 8 bits



Esta página ilustra varios métodos para configurar los temporizadores en un MCU AVR® de 8 bits . Se proporciona una descripción general de los temporizadores en un AVR antes de presentar el ejemplo paso a paso.

## Requisitos para ejecutar los ejemplos prácticos

### Requisitos de hardware

- ATmega328PB Mini tablero explicado
- Cable micro USB

#### Herramienta



(<http://www.atmel.com/tools/MEGA328PB-XMINI.aspx>)

Mini kit de evaluación  
ATmega328PB  
Xplained

#### Sobre

[\(http://www.atmel.com/tools/MEGA328PB-XMINI.aspx\)](http://www.atmel.com/tools/MEGA328PB-XMINI.aspx)

[\(https://www.microchipdirect.com/prc\)](https://www.microchipdirect.com/prc)

El kit de evaluación ATmega328PB Xplained Mini es una plataforma de hardware para evaluar el microcontrolador Atmel ATmega328PB. NO se necesita un kit de depurador externo para ejecutar estos laboratorios. El ATmega328PB tiene un depurador incorporado completamente integrado.

Este ejercicio práctico demuestra cómo configurar los diferentes temporizadores.

### Herramientas de software

#### Instaladores



#### Instrucciones de instalación

#### Herramienta

#### Sobre

Atmel® Studio  
Integrated Development Environment

[\(/atstudio:start\)](http://atstudio:start)

<http://studio.download.atmel.com/7.0.1931/as-installer-7.0.1931-full.exe>

[Download](#)

[Download](#)

[\(/install:atstudio\)]((/install:atstudio)

### Additional Files

#### Files

[Xplained Board User Guide](http://www.atmel.com/Images/Atmel-42287-ATmega328P-Xplained-Mini-User-Guide_UserGuide.pdf) ([http://www.atmel.com/Images/Atmel-42287-ATmega328P-Xplained-Mini-User-Guide\\_UserGuide.pdf](http://www.atmel.com/Images/Atmel-42287-ATmega328P-Xplained-Mini-User-Guide_UserGuide.pdf))

## Overview of Timers on 8-bit AVR MCUs

**ATmega328PB** microcontroller has five Timer/Counters:

Timer 0	TC0	8-bit Timer/counter with PWM
Timer 1	TC1	16-bit Timer/counter with PWM and Asynchronous Operation.
Timer 2	TC2	8-bit Timer/counter with PWM and Asynchronous Operation.
Timer 3	TC3	16-bit Timer/counter with PWM and Asynchronous Operation.
Timer 4	TC4	16-bit Timer/counter with PWM and Asynchronous Operation.

**Definitions:****Register Nomenclature**

**BOTTOM** The counter reaches the BOTTOM when it becomes zero (0x00 for 8-bit counters and 0x0000 for 16bit counters)

**MAX** The counter reaches its maximum value when it becomes 0x0F (15 decimal) for 8-bit counters and 0x00FF (255 decimal) for 16-bit counters

**TOP** The counter reaches the TOP when its value becomes equal to the highest value possible. The TOP value can be assigned fixed value MAX or the value stored in the OCRxA register. This assignment is dependent upon the mode of operation

## Timer 0 - 8-bit Timer/Counter with PWM

Timer/Counter0 (TC0) is a general purpose 8-bit Timer/Counter module, with two independent Output Compare Units, and PWM support.

### TC0 Registers

The Timer/Counter 0 register (TCNT0) and Output Compare TC0x registers (OCR0x) are 8-bit registers. Interrupt request signals are all visible in the Timer Interrupt Flag Register 0 (TIFR0). All interrupts are individually masked with the Timer Interrupt Mask Register 0 (TIMSK0).

**Name:** TCCR0B  
**Offset:** 0x45  
**Reset:** 0x00

**Property:** When addressing as I/O Register: address offset is 0x25

Bit	7	6	5	4	3	2	1	0
Access	FOC0A	FOC0B			WGM02	CS02	CS01	CS00
R/W	R/W	R/W			R/W	R/W	R/W	R/W
Reset	0	0			0	0	0	0

(/local--files/avr8:avrtimer/TCCR0B.PNG)

### TC0 Timer/Counter Clock Sources

TC0 can be clocked by an internal or an external clock source. The clock source is selected by writing to the Clock Select (CS02:0) bits in the Timer/Counter Control Register (TCCR0B).

#### Bits 2:0 – CS0n: Clock Select [n = 0..2]

The three Clock Select bits select the clock source to be used by the Timer/Counter.

CS02	CS01	CS00	Description
0	0	0	No Clock source (Timer stopped)
0	0	1	clkio/1 (No prescaling)
0	1	0	clkio/8 (From prescaler)
0	1	1	clkio/64 (From prescaler)
1	0	0	clkio/256 (From prescaler)
1	0	1	clkio/1012 (From prescaler)
1	1	0	External clock source on T0 pin (clock on falling edge)
1	1	1	External clock source on T0 pin (clock on rising edge)

### TC0 Counter Unit

Depending on the mode of operation used, T0 is cleared, incremented, or decremented at each timer clock (clkT0). clkT0 can be generated from an external or internal clock source, selected by the Clock Select bits (CS0[2:0]).

The counting sequence is determined by the setting of the WGM01 and WGM00 bits located in the T0 Control Register A (TCCR0A) and the WGM02 bit located in the Timer/Counter Control Register B (TCCR0B).

### Bits 1:0 – WGM0n: Waveform Generation Mode [n = 1:0]

Combined with the WGM02 bit found in the TCCR0B Register, these bits control the counting sequence of the counter, the source for maximum (TOP) counter value, and what type of waveform generation to be used. Modes of operation supported by the Timer/Counter unit are: Normal mode (counter), Clear Timer on Compare Match (CTC) Mode, and two types of Pulse Width Modulation (PWM) modes.

Table 1-2 Waveform Generation Mode Bit Description

Mode	WGM2:0	Mode of Operation	TOP	OCR0x Update	TOV flag set on
0	0 0 0	Normal	0xFF	Immediate	MAX
1	0 0 1	PWM Phase Correct	0xFF	TOP	BOTTOM
2	0 1 0	CTC	OCRA	Immediate	MAX
3	0 1 1	Fast PWM	0xFF	BOTTOM	MAX
4	1 0 0	Reserved	~	~	~
5	1 0 1	PWM Phase Correct	OCRA	TOP	BOTTOM
6	1 1 0	Reserved	~	~	~
7	1 1 1	Fast PWM	OCRA	BOTTOM	MAX

**Note:**

1. MAX = 0xFF
2. BOTTOM = 0x00

### Modes of Operation for TC0

The mode of operation determines the behavior of TC0 and the Output Compare pins. It is defined by the combination of the Waveform Generation mode bits and Compare Output mode bits in the Timer/Counter control Registers A and B (TCCR0B.WGMn2, TCCR0A.WGM01, TCCR0A.WGM00, and TCCR0A.COM0x[1:0]).

Available modes of operation for TC0 are:

- Normal Mode
- Clear Timer on Compare Match (CTC) Mode
- Fast PWM Mode
- Phase Correct PWM Mode

### Clear Timer on Compare Match Mode

In Clear Timer on Compare mode (WGM0[2:0]=0x2), the OCR0A Register is used to manipulate the counter resolution: the counter is cleared to ZERO when the counter value (TCNT0) matches the OCR0A. The OCR0A defines the top value for the counter, hence also its resolution.

The counter value (TCNT0) increases until a compare match occurs between TCNT0 and OCR0A, and then counter (TCNT0) is cleared. An interrupt can be generated each time the counter value reaches the TOP value by setting the OCF0A Flag. If the interrupt is enabled, the interrupt handler routine can be used for updating the TOP value.

The waveform frequency is defined by the following equation:

$$f_{\text{OCnx}} = \frac{f_{\text{clk\_I/O}}}{2 \cdot N \cdot (1 + \text{OCRnx})}$$

(/local--files/avr8avr:avrtimer/freqz0.PNG)

N represents the prescaler factor (1, 8, 64, 256, or 1024).

## TC1, TC3, & TC4 - 16-bit Timer/Counters with PWM

The 16-bit Timer/Counter units allow accurate program execution timing (event management), wave generation, and signal timing measurement.

### Registers (TC1, TC3, TC4)

- The Timer/Counter (TCNTn), Output Compare Registers (OCRA/B), and Input Capture Register (ICRn) are all 16-bit registers.
- The Timer/Counter Control Registers (TCCRnA/B) are 8-bit registers and have no CPU access restrictions.
- Interrupt requests (abbreviated to Int.Req. in the block diagram) signals are all visible in the Timer Interrupt Flag Register (TIFRn). All interrupts are individually masked with the Timer Interrupt Mask Register (TIMSKn).

### Timer/Counter Clock Sources (TC1, TC3, TC4)

The Timer/Counter can be clocked by an internal or an external clock source. The clock source is selected by the Clock Select logic which is controlled by the Clock Select bits in the timer/Counter control Register B (TCCRnB.CS[2:0]).

Name: TCCR1B, TCCR3B, TCCR4B  
 Offset: 0x81 + n\*0x10 [n=0..2]  
 Reset: 0x00  
 Property: -

Bit	7	6	5	4	3	2	1	0
	ICNC	ICES		WGM3	WGM2		CS[2:0]	
Access	R/W	R/W		R/W	R/W	R/W	R/W	R/W
Reset	0	0		0	0	0	0	0

(/local--files/8avr:avrtimer/TCCR1B.PNG)

Bits 2:0 – CS[2:0]: Clock Select [n = 0..2]

The three Clock Select bits select the clock source to be used by the Timer/Counter.

CS02	CS01	CS00	Description
0	0	0	No Clock source ( Timer stopped)
0	0	1	clkio/1 (No prescaling)
0	1	0	clkio/8 (From prescaler)
0	1	1	clkio/64 (From prescaler)
1	0	0	clkio/256 (From prescaler)
1	0	1	clkio/1012 (From prescaler)
1	1	0	External clock source on T0 pin (clock on falling edge)
1	1	1	External clock source on T0 pin (clock on rising edge)

### Counter Unit (TC1, TC3, TC4)

TC1, TC3, and TC4 are programmable 16-bit bi-directional counters.

Each 16-bit counter is mapped into two 8-bit I/O memory locations: Counter High (TCNTnH) containing the upper eight bits of the counter, and Counter Low (TCNTnL) containing the lower eight bits.

Depending on the selected mode of operation, the counter is cleared, incremented, or decremented at each timer clock (clkTn). The clock clkTn can be generated from an external or internal clock source, as selected by the Clock Select bits in the Timer/Counter Control Register B (TCCRnB.CS[2:0]).

The counting sequence is determined by the setting of the Waveform Generation mode bits in the Timer/Counter Control Registers A and B (TCCRnB.WGM[3:2] and TCCRnA.WGM[1:0]).

### Modes of Operation (TC1, TC3, TC4)

The mode of operation is determined by the combination of the Waveform Generation mode (WGM[3:0]) and Compare Output mode (TCCRnA.COMx[1:0]) bits.

Available modes of operation are:

- Normal Mode
- Clear Timer on Compare Match (CTC) Mode
- Fast PWM Mode
- Phase Correct PWM Mode
- Phase and Frequency Correct PWM Mode

### Fast PWM Mode

The Fast Pulse Width Modulation or Fast PWM modes (modes 5, 6, 7, 14, and 15, WGM[3:0]=0x5, 0x6, 0x7, 0xE, 0xF) provide a high frequency PWM waveform generation option. The Fast PWM differs from the other PWM options by its single-slope operation. The counter counts from BOTTOM to TOP then restarts from BOTTOM.

In Fast PWM mode the counter is incremented until the counter value matches either one of the fixed values 0x00FF, 0x01FF, or 0x03FF (WGM[3:0] = 0x5, 0x6, or 0x7), the value in ICRn (WGM[3:0]=0xE), or the value in OCRnA (WGM[3:0]=0xF). The counter is then cleared at the following timer clock cycle.

The PWM frequency for the output can be calculated by the following equation:

$$f_{OCnxPWM} = \frac{f_{clk\_I/O}}{N \cdot (1 + TOP)}$$

(/local--files/8avr:avrtimer/freqz1.PNG)

#### Note:

- The "n" in the register and bit names indicates the device number (n = 0 for Timer/Counter 0), and the "x" indicates Output Compare unit (A/B).
- N represents the prescale divider (1, 8, 64, 256, or 1024).

## TC2 - 8-bit Timer/Counter2 with PWM and Asynchronous Operation

Timer/Counter2 (TC2) is a general purpose, dual channel, 8-bit Timer/Counter module.

### TC2 Registers

The Timer/Counter (TCNT2) and Output Compare Register (OCR2A and OCR2B) are 8-bit registers. Interrupt request signals are all visible in the Timer Interrupt Flag Register (TIFR2). All interrupts are individually masked with the Timer Interrupt Mask Register (TIMSK2).

Name: TCCR2B  
Offset: 0xB1  
Reset: 0x00  
Property: -

Bit	7	6	5	4	3	2	1	0
	FOC2A	FOC2B			WGM22	CS22	CS21	CS20
Access	R/W	R/W			R/W	R/W	R/W	R/W
Reset	0	0			0	0	0	0

(/local--files/8avr:avrtimer/TCCR2B.PNG)

### TC2 Clock Sources

TC2 can be clocked by an internal synchronous or an external asynchronous clock source:

The three Clock Select bits (CS2:CS0) select the clock source to be used by the Timer/Counter.

CS02	CS01	CS00	Description
0	0	0	No Clock source (Timer stopped)
0	0	1	clkio/1 (No prescaling)
0	1	0	clkio/8 (From prescaler)
0	1	1	clkio/64 (From prescaler)
1	0	0	clkio/256 (From prescaler)
1	0	1	clkio/1012 (From prescaler)
1	1	0	External clock source on T0 pin (clock on falling edge)
1	1	1	External clock source on T0 pin (clock on rising edge)

### TC2 Counter Unit

Depending on the mode of operation used, the counter is cleared, incremented, or decremented at each timer clock (clkT2). clkT2 can be generated from an external or internal clock source, selected by the Clock Select bits (CS2[2:0]).

The counting sequence is determined by the setting of the WGM21 and WGM20 bits located in the Timer/Counter Control Register (TCCR2A) and the WGM22 bit located in the Timer/Counter Control Register B (TCCR2B).

## TC2 Modes of Operation

The mode of operation, i.e., the behavior of the Timer/Counter and the Output Compare pins, is defined by the combination of the Waveform Generation mode (WGM2[2:0]) and Compare Output mode (COM2x[1:0]) bits.

Available modes of operation are:

- Normal Mode
- Clear Timer on Compare Match (CTC) Mode
- Fast PWM Mode
- Phase Correct PWM Mode

### Normal Mode

In the Normal mode (WGM22:0 = 0) the counting direction is always up (incrementing), without having the counter cleared. The counter will roll over to 0c00 when it passes its maximum 8-bit value (TOP = 0xFF).

In normal operation, the Timer/Counter Overflow Flag (TOV2) will be set in the same timer clock cycle as the TCNT2 becomes zero. The TOV2 Flag, in this case, behaves like a ninth bit, except that it is only set, not cleared. However, combined with the timer overflow interrupt that automatically clears the TOV2 flag, the timer resolution can be increased by software. There are no special cases to consider in the Normal mode, a new counter value can be written any time.



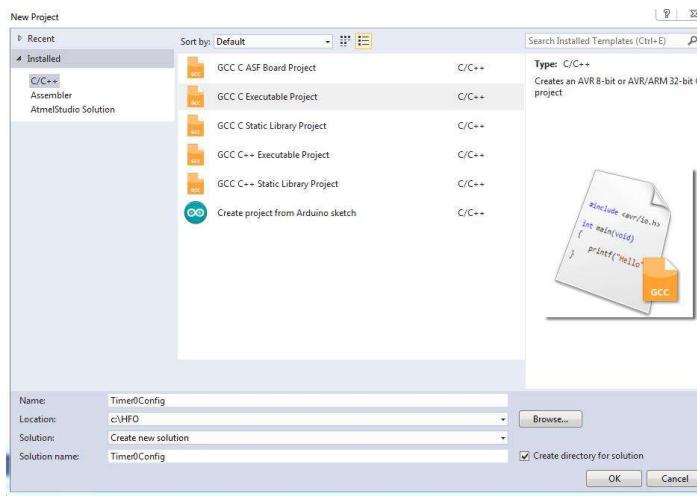
Regardless of which mode is used the programmer needs to remember two things:

1. The timer has to be started by selecting the clock source.
2. If interrupts are used they must be enabled.

## Step-by-Step

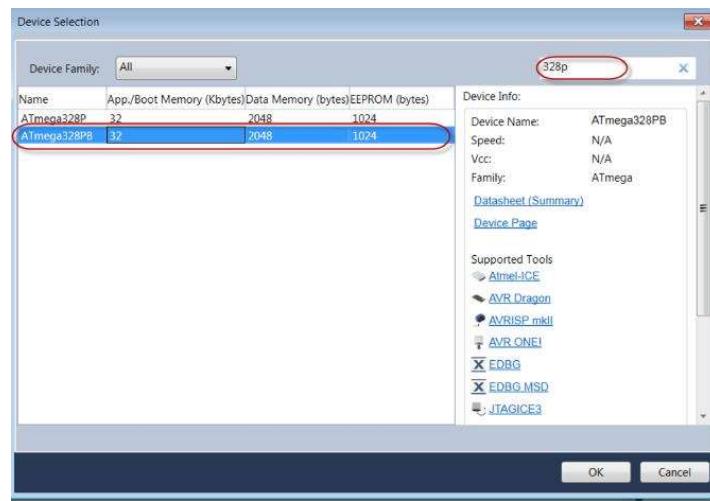
### Project Creation

- 1 Open Atmel Studio 7
- 2 Select **File > New > Project**
- 3 In the **New Project** window select **GCC C executable Project** and give the project name to "Timer0Config". Set the location of the project files. (This example uses "C:\HFO\TimersConfigurations"). Click **OK**.



(/local--files/8avr:avrtimer/new-project.jpg)

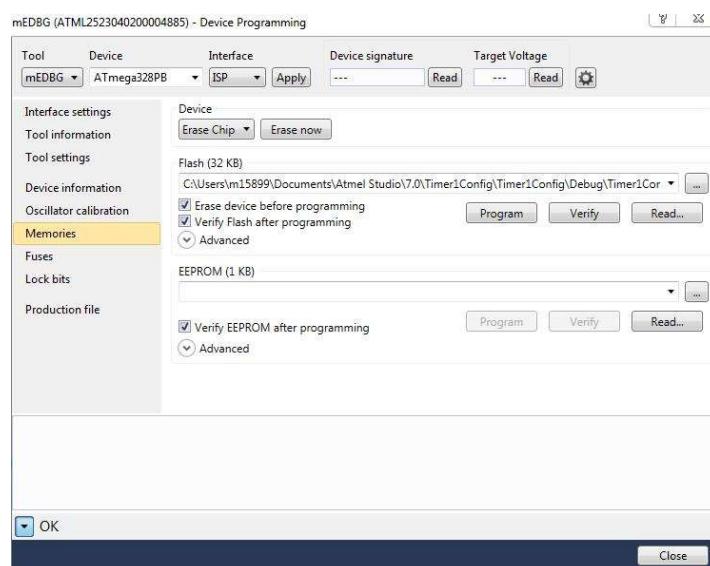
- 4 In the search bar of the **Device Selection** window type "328p" then select the device **Atmega328PB**, click **OK**.



(/local--files/8avr:avrtimer/select-device.jpg)

## Programming the board

- 5 Select **Build > Build Solution** from the top menu to compile the code. You will see a "BUILD SUCCEEDED" message in the output window of Studio 7.
- 6 Select **Tools > Device Programming**, make sure the settings are as in the picture below and click **Apply**.



(/local--files/8avr:avrtimer/program.jpg)

- 7 Select **Memories > Program** Wait until the "Verifying Flash...OK" message to appear.

## Project Modifications



`avr/io.h` and `avr/interrupt.h` must be "#included" in `main.c` for this project to compile and run.

- `io.h` provides for the definition of all of the AVR peripherals.
- `interrupt.h` is required for all applications using interrupts.

## 8 Modification 1 - Blinking a LED using TC0

In this example Timer 0 blinks the LED connected to PB5 every 32 ms.

- a Modify the main function by adding the following code:

```

1 <font></font> ? 
2 int main(void)<font></font>
3 <font></font>
4 {<font></font>
5   //set RB5 as output<font>
6   DDRB |= 1 << DDBR5; <font>
7 <font></font>
8   //call TMR0 initializati<font>
9   init_TC0(); <font></font>
10 <font></font>
11   //enable interrupt<font>
12   sei(); <font></font>
13 <font></font>
14   while (1)<font></font>
15   {<font></font>
16     //main loop<font></font>
17   }<font></font>
18 }<font></font>
19 <font></font>

```

- b** Create the function `init_TC0()` in `main.c`. Add the following code:

```

1 <font></font> ? 
2 void init_TC0(void)<font></font>
3 {<font></font>
4 <font></font>
5   // Set the Timer Mode to CTC<font>
6   TCCR0A |= (1 << WGM01); <font>
7 <font></font>
8   // Set the value that you want<font>
9   OCR0A = 0xF9; //249<font>*<font>
10 <font></font>
11   //Set the ISR COMPA vect<font>
12   TIMSK0 |= (1 << OCIE0A); <font>
13 <font></font>
14   // set prescaler to 1024 <font>
15   TCCR0B |= (1 << CS00) | (0 << CS01); <font>
16 }<font></font>
17 <font></font>

```

The preceding code does the following:

- Configures Timer 0 in CTC mode by setting the corresponding WGM bit in TCCR0A register;
- Sets the TOP value in the OCR0A register calculated using the equation 1 to blink the LED at 30 kHz;
- Enables the Timer Interrupt;
- Sets the pre-scaler to 1024 by configuring the CS00 and CS02 bits in TCCR0B register;

- c** Add the interrupt service routine to `main.c`:

```

1 <font></font> ? 
2 ISR (TIMER0_COMPA_vect) <font></font>
3 {<font></font>
4   //event to be executed every 30ms<font>
5   counter++; <font></font>
6   if (counter == MyTimerConstant)
7     counter = 0; <font></font>
8   PORTB ^= 1 << PORTB5; <font>
9 }<font></font>
10 }<font></font>
11 <font></font>

```

**⚠** The use of the constant `MyTimerConstant` and the variable `counter` is optional. If used, changing the value of MyTimer Constant will alter the time it takes the LED to blink.

## 9 Modification 2 - Using TC1 in PWM mode to dim the LED

This example uses TC1 to generate a PWM signal which is fed through an I/O pin to drive an LED. Changing the PWM duty-cycle will result in a change in the brightness of the LED.

- a** Modify `main()` by adding the following code.



```

1 <font></font>?
2 int main(void)<font></font>
3 <font></font>
4 {<font></font>
5   //set direction of pin F
6   DDRB |= 1 << DDB1;<font>
7 <font></font>
8   init_TC1_pwm();<font></f
9 <font></font>
10  //enable global interrupt
11  sei(); <font></font>
12 <font></font>
13  while (1)<font></font>
14  {<font></font>
15    //main loop<font></f
16  }<font></font>
17 }<font></font>
18 <font></font>

```

For configuring the direction bit of pin PB1, write the DDB1 bit to logic 1 in the DDRB register. On the ATmega328PB Xplained Mini Board, LED is connected to PB5 and PWM is generated on PB1. To view the LED dimming, connect a wire from PB5 to PB1 on the ATmega328PB Xplained Mini Board, as shown below.

- b** Create the init\_TC1\_pwm () function before the main loop with the following code:



```

1 <font></font>?
2 void init_TC1_pwm(void)<font>
3 {<font></font>
4   //clear OCnA on compare m:
5   TCCR1A = (1 << COM1A1);<fo
6 <font></font>
7   //the counting sequence is:
8   TCCR1A |= (1 << WGM10);<fo
9   TCCR1B |= (1 << WGM12);<fo
10 <font></font>
11   //256 prescaler clock selec
12   TCCR1B |= (0 << CS10) | (1
13 <font></font>
14   //enable interrupts<font><
15   TIMSK0 |= (1 << OCIE1A);<fo
16 }<font></font>
17 <font></font>

```

init\_TC1\_PWM() performs the following"

- Sets the bits COMA1=1 and COMA0=0 (non-inverting mode from datasheet table Compare Output Mode, Fast PWM in the TCCR1A register;
- Configures Bits 1:0 – WGM[1:0]:Waveform Generation Mode accordingly for Fast PWM, 8-bit mode;
- Verifies the configuration for the clock source and prescaler to be used by the Timer/Counter, which decides the frequency of the PWM. Internal clock source divided by 256 is configured by writing to the CS10 and CS12 bits in TCCR1B register;
- Enables the Timer Interrupt;

- c** Add the ISR function after the main loop to execute dimming LED event:



```

1 <font></font>?
2 ISR (TIMER1_COMPA_vect) //?
3 {<font></font>
4   duty_cycle--;<font></for
5   if (duty_cycle == 0) {<fo
6     duty_cycle = 0xFF;<fo
7   }<font></font>
8   OCR1A = duty_cycle;<font>
9 <font></font>
10 } <font></font>
11 <font></font>

```

The register OC1RA decides the duty cycle of PWM. To dim the LED gradually decrease the duty cycle.

## 10 Blinking an LED using TC2

For this example, Timer 2 (TC2) is configured to blink the LED connected to PB5 at a period of 3 seconds.

**a** Modify the main function by adding the following code:



```

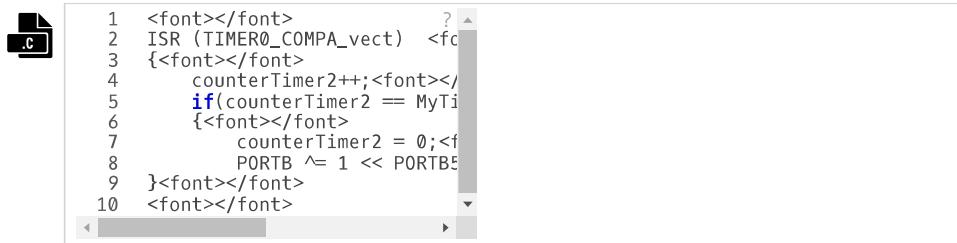
1 <font></font>
2 int main(void)<font></font>
3 {<font></font>
4     //set direction of PB5 a
5     DDRB |= 1 << DDB5;<font>
6     /* Timer clock = I/O clk
7     TCCR2B = (1 << CS22) | (
8     <font></font>
9     /* Clear overflow flag */
10    TIFR2 = 1 << TOV2;<font>
11    <font></font>
12    /* Enable Overflow Interrupt */
13    TIMSK2 = 1 << TOIE2;<font>
14    <font></font>
15    // enable global interrupt
16    sei();<font></font>
17    <font></font>
18    while (1)<font></font>
19    {<font></font>
20        // Main loop<font></font>
21        }<font></font>
22    }<font></font>
23 <font></font>
24 <font></font>

```

- Configures the direction bit of pin PB5, write the DDB5 bit to logic 1 in the DDRB register;
- Sets the prescaler to 1024 by configuring the CS20, CS21 and CS22 bits in TCCR2B register;
- Clears Overflow flag - TOV2 is cleared by writing a logic one to the flag.
- Enables global interrupts by calling `sei();`

**b** Enable the Overflow Timer Interrupt;

Add the ISR function after the main loop to execute blinking LED event:



```

1 <font></font>
2 ISR (TIMER0_COMPA_vect) <font></font>
3 {<font></font>
4     counterTimer2++;<font></font>
5     if(counterTimer2 == MyTi
6     {<font></font>
7         counterTimer2 = 0;<f
8         PORTB ^= 1 << PORTB5
9     }<font></font>
10 <font></font>

```

Set the PORTB register to blink the LED

**!** MyTimer2Constant is optional. This constant can be changed to alter the time it takes the LED to blink.

# Sensor de temperatura interna AVR

Algunos dispositivos AVR tienen un sensor de temperatura interno. Se puede utilizar para medir la temperatura central del dispositivo (no la temperatura ambiente alrededor del dispositivo). El voltaje medido tiene una relación lineal con la temperatura. La sensibilidad de voltaje es de aproximadamente 1 mV/°C, la precisión de la medición de temperatura es de ±10°C.

La medición de temperatura se basa en el sensor de temperatura en el chip que está acoplado a un canal ADC de un solo extremo. Seleccionar el canal ADC 8 escribiendo '1000' en ADMUX.MUX[3:0] habilita el sensor de temperatura. La referencia de voltaje interno de 1,1 V también debe seleccionarse para la fuente de referencia de voltaje ADC. Cuando el sensor de temperatura está habilitado, el convertidor ADC se puede usar en modo de conversión simple para medir el voltaje sobre el sensor de temperatura.

## Muestra de datos de medición

Temperature	-45°C	+25°C	+85°C
Voltage	242mV	314mV	380mV

(/local--files/8avr:avrtemp/temp1.png)

## Calibración

The results from temperature measurements have offset and gain errors. The internal temperature reference can be corrected for these errors by making calibration measurements at one or two known temperatures and adjusting the output values. This can result in very precise temperature measurements, sometimes as accurate as ±2°C.

More detail can be found in this application note. ([http://www.atmel.com/Images/Atmel-8108-Calibration-of-the-AVR's-Internal-Temperature-Reference\\_ApplicationNote\\_AVR122.pdf](http://www.atmel.com/Images/Atmel-8108-Calibration-of-the-AVR's-Internal-Temperature-Reference_ApplicationNote_AVR122.pdf))

# Configuring the ADC

The internal 1.1 V voltage reference must be selected for the ADC voltage reference source when using the internal temperature sensor. Writing “11” to the **REFS1** and **REFS0** bits of the **ADMUX register** selects the internal 1.1 V Voltage Reference.

The ADC has multiple input channels and modes of operation. The **Single Conversion** mode can be used to convert the temperature sensor signal connected to channel 8. To select channel 8, writing “1000” to the MUX3 thru MUX0 bits selects channel 8 or the temperature sensor.

Once the conversion is complete, the result is stored in two 8-bit ADC data registers ADCH (higher 8-bits) and ADCL (lower 8-bits). The 10-bit result can be either left justified or right justified. If ADLAR bit is set to a “1”, then the result is left adjusted to the upper 10-bits of the two registers. If set to “0” then the result occupies the lower 10 bits of the two registers. By default, each bit is cleared and the word is right justified.

This code statement will set the bits as described.

```
ADMUX = (1«REFS1) | (1«REFS0) | (0«ADLAR) | (1«MUX3) | (0«MUX2) | (0«MUX1) |  
(0«MUX0);
```

**Name:** ADMUX

**Offset:** 0x7C

**Reset:** 0x00

**Property:** -

Bit	7	6	5	4	3	2	1	0
	REFS1	REFS0	ADLAR		MUX3	MUX2	MUX1	MUX0
Access	R/W	R/W	R/W		R/W	R/W	R/W	R/W

#### Bits 7:6 – REFSn: Reference Selection [n = 1:0]

These bits select the voltage reference for the ADC. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in ADCSRA is set). The internal voltage reference options may not be used if an external reference voltage is being applied to the AREF pin.

Table 29-3 ADC Voltage Reference Selection

REFS[1:0]	Voltage Reference Selection
00	AREF, Internal $V_{ref}$ turned off
01	$AV_{CC}$ with external capacitor at AREF pin
10	Reserved
11	Internal 1.1V Voltage Reference with external capacitor at AREF pin

(/local--files/8avr:avrtemp/temp2.png)

**Bits 3:0 – MUXn: Analog Channel Selection [n = 3:0]**

The value of these bits selects which analog inputs are connected to the ADC. If these bits are changed during a conversion, the change will not go in effect until this conversion is complete (ADIF in [ADCSRA](#) on page 323 is set).

**Table 29-4 Input Channel Selection**

MUX[3:0]	Single Ended Input
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5

(/local--files/8avr:avrtemp/temp3.png)

MUX[3:0]	Single Ended Input
0110	ADC6
0111	ADC7
1000	Temperature sensor
1001	Reserved
1010	Reserved
1011	Reserved
1100	Reserved
1101	Reserved
1110	1.1V ( $V_{BG}$ )
1111	0V (GND)

(/local--files/8avr:avrtemp/temp4.png)

## Configuring the ADC Clock and Conversion Timing

The ADC can prescale the system clock to provide an ADC clock that is between 50 kHz and 200 kHz to get maximum resolution. If an ADC resolution of less than 10-bits is required, then the ADC clock frequency can be higher than 200 kHz. At 1 MHz it is possible to achieve up to eight bits of resolution.

The prescaler value is selected with **ADPS bits** in **ADCSRA Register**. For example; writing “110” to the **ADCSRA register** selects the divide by 64 pre-scaler resulting in a 125 KHz ADC clock when an 8 MHz oscillator clock is used.

### ADC Control and Status Register A

**Name:** ADCSRA**Offset:** 0x7A**Reset:** 0x00**Property:** -

Bit	7	6	5	4	3	2	1	0
Access	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
Reset	0	0	0	0	0	0	0	0

(/local--files/8avr:avrtemp/temp5.png)

**Bit 7 – ADEN: ADC Enable**

Writing this bit to one enables the ADC. By writing it to zero, the ADC is turned off. Turning the ADC off while a conversion is in progress, will terminate this conversion.

**Bit 6 – ADSC: ADC Start Conversion**

In Single Conversion mode, write this bit to one to start each conversion. In Free Running mode, write this bit to one to start the first conversion. The first conversion after ADSC has been written after the ADC has been enabled, or if ADSC is written at the same time as the ADC is enabled, will take 25 ADC clock cycles instead of the normal 13. This first conversion performs initialization of the ADC.

ADSC will read as one as long as a conversion is in progress. When the conversion is complete, it returns to zero. Writing zero to this bit has no effect.

**Bit 5 – ADATE: ADC Auto Trigger Enable**

When this bit is written to one, Auto Triggering of the ADC is enabled. The ADC will start a conversion on a positive edge of the selected trigger signal. The trigger source is selected by setting the ADC Trigger Select bits, ADTS in ADCSRB.

**Bit 4 – ADIF: ADC Interrupt Flag**

This bit is set when an ADC conversion completes and the Data Registers are updated. The ADC Conversion Complete Interrupt is executed if the ADIE bit and the I-bit in SREG are set. ADIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, ADIF is cleared by writing a logical one to the flag. Beware that if doing a Read-Modify-Write on ADCSRA, a pending interrupt can be disabled. This also applies if the SBI and CBI instructions are used.

**Bit 3 – ADIE: ADC Interrupt Enable**

When this bit is written to one and the I-bit in SREG is set, the ADC Conversion Complete Interrupt is activated.

**Bits 2:0 – ADPSn: ADC Prescaler Select [n = 2:0]**

These bits determine the division factor between the system clock frequency and the input clock to the ADC.

(/local--files/8avr:avrtemp/temp6.png)

**Table 29-5 Input Channel Selection**

ADPS[2:0]	Division Factor
000	2
001	2

(/local--files/8avr:avrtemp/temp7.png)

ADPS[2:0]	Division Factor
010	4
011	8
100	16
101	32
110	64
111	128

(/local--files/8avr:avrtemp/temp8.png)

## Starting a Conversion

In single conversion mode the **ADSC bit** in the **ADCSRA register** must be set to a logical one state to start the ADC conversion. This bit remains at logic high while the conversion is in progress and is cleared by the hardware, once the conversion is complete.

The first conversion after the ADC is switched on takes 25 ADC clock cycles in order to initialize the analog circuitry. Then, for further conversions, it takes 13 ADC clock cycles (13.5 for Auto-triggered conversions).

## Sample Project

A sample project for using the Temperature Sensor is available here. (/8avr:avradc)

# Ejemplo de sensor de temperatura interna del convertidor analógico a digital (ADC) AVR de 8 bits

## ◎ Objetivo

Este proyecto práctico muestra un ejemplo simple de lectura del sensor de temperatura en el chip. Leer el sensor de temperatura puede ser un proyecto gratificante en sí mismo. Confirma que completó la compilación del software y la configuración del hardware del ADC, y pudo programar el microcontrolador con éxito. Finalmente, el depurador se usa para ver la temperatura usando la ventana de salida de Studio 7.

El sensor de temperatura en el chip está acoplado a un canal ADC8 de un solo extremo. Seleccionar el canal ADC8 escribiendo `ADMUX.MUX[3:0]` a '1000' habilita el sensor de temperatura. La referencia de voltaje interna de 1,1 V también debe seleccionarse para la fuente de referencia de voltaje ADC en la medición del sensor de temperatura. Cuando el sensor de temperatura está habilitado, el convertidor ADC se puede usar en modo de conversión simple para medir el voltaje sobre el sensor de temperatura.

La sensibilidad de voltaje es de aproximadamente 1 mV/°C, la precisión de la medición de temperatura es de ±10°C.

## ☒ Materiales

### Herramientas de hardware (*opcional*)

Herramienta	Sobre
	Mini kit de evaluación ( <a href="http://www.atmel.com/tools/MEGA328PB-XMINI.aspx">http://www.atmel.com/tools/MEGA328PB-XMINI.aspx</a> ) <a href="http://www.atmel.com/tools/MEGA328PB-XMINI.aspx">ATmega328PB-Xplained</a> <a href="http://www.atmel.com/tools/MEGA328PB-XMINI.aspx">(http://www.atmel.com/tools/MEGA328PB-XMINI.aspx)</a> <a href="https://www.microchipdirect.com/prc...">T (https://www.microchipdirect.com/prc...</a>

### Herramientas de software

Herramienta	Sobre	Instaladores			Instrucciones de instalación
		Windows	Linux	Mac OSX	
	<a href="http://studio.download.atmel.com/7.0.1931/as-installer-7.0.1931-full.exe">□ (atstudio:start)</a> <a href="http://studio.download.atmel.com/7.0.1931/as-installer-7.0.1931-full.exe">Download</a> <a href="http://studio.download.atmel.com/7.0.1931/as-installer-7.0.1931-full.exe">http://studio.download.atmel.com/7.0.1931/as-installer-7.0.1931-full.exe</a> <a href="#">Windows</a> <a href="#">Linux</a> <a href="#">Mac OSX</a> <a href="http://studio.download.atmel.com/7.0.1931/as-installer-7.0.1931-full.exe">(/install:atstudio)</a>				

### Exercise Files

File	Windows
	<a href="https://microchiptechnology.sharepoint.com/:f/s/DeveloperHelp/EgvbV6l04yBAvXfl6BemPdoBdqWAfKoruX9pkt21RgFG-w?e=pCb004">Download</a> <a href="https://microchiptechnology.sharepoint.com/:f/s/DeveloperHelp/EgvbV6l04yBAvXfl6BemPdoBdqWAfKoruX9pkt21RgFG-w?e=pCb004">https://microchiptechnology.sharepoint.com/:f/s/DeveloperHelp/EgvbV6l04yBAvXfl6BemPdoBdqWAfKoruX9pkt21RgFG-w?e=pCb004</a>

### Additional Files

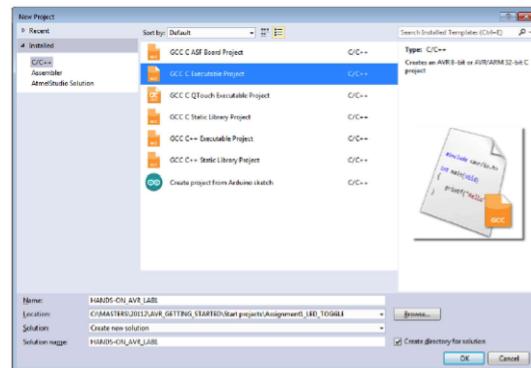
**Files**

 [Xplained Board User Guide](http://www.atmel.com/Images/Atmel-42287-ATmega328P-Xplained-Mini-User-Guide_UserGuide.pdf) ([http://www.atmel.com/Images/Atmel-42287-ATmega328P-Xplained-Mini-User-Guide\\_UserGuide.pdf](http://www.atmel.com/Images/Atmel-42287-ATmega328P-Xplained-Mini-User-Guide_UserGuide.pdf))

## Procedure

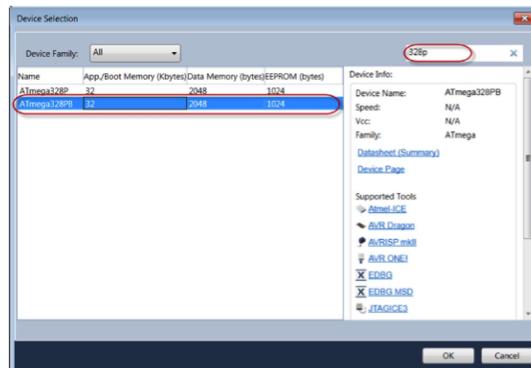
### 1 Task 1 - Project Creation

- Open Atmel Studio 7
- Select **File > New > Project**
- Select **GCC C Executable Project** and give it the name **Project1**
- Choose a location to save the project on your computer



(/local--files/8avr:avradc/step1.png)

- The Device Selection window will appear. In the search bar enter **328P**, then select the device **Atmega328PB** and click OK.



(/local--files/8avr:avradc/step2.png)

### 2 Task 2 - Main.c

This project reads the internal temperature sensor, converts the result to degrees centigrade, then stores the result in **ADC Temperature Result**.

- 1) The `main.c` file is where the application code is added. The project has a `main.c` file already created but it only contains a `while(1)` statement. Modify `main.c` by entering the lines in the gray code block below.



#include <avr/io.h> is automatically added to the `main.c` file when it is generated. This should always be placed before the `main(void)` loop. The header file `io.h` calls the `iom328pb.h` file that defines the ADC register definitions.

```

unsigned int Ctemp;
unsigned int Ftemp;

int main(void)
{

    /* Setup ADC to use int 1.1V reference
    and select temp sensor channel */
    ADMUX = (1<<REFS1) | (1<<REFS0) | (0<<ADLAR) | (1<<MUX3) | (0<<MUX2) | (0<<MUX1) | (0<
    <MUX0);

    /* Set conversion time to
    112usec = [(1/(8Mhz / 64)) * (14 ADC clocks per conversion)]
    and enable the ADC*/
    ADCSRA = (1<<ADPS2) | (1<<ADPS1) | (1<<ADEN);

    /* Perform Dummy Conversion to complete ADC init */
    ADCSRA |= (1<<ADSC);

    /* wait for conversion to complete */
    while ((ADCSRA & (1<<ADSC)) != 0);

    /* Scan for changes on A/D input pin in an infinite loop */
    while(1)
    {
        /* start a new conversion on channel 8 */
        ADCSRA |= (1<<ADSC);

        /* wait for conversion to complete */
        while ((ADCSRA & (1<<ADSC)) != 0)
        ;

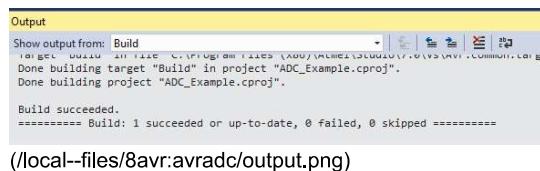
        /* Calculate the temperature in C */
        Ctemp = (ADC - 247)/1.22;
        Ftemp = (Ctemp * 1.8) + 32;
    }

    return -1;
}

```

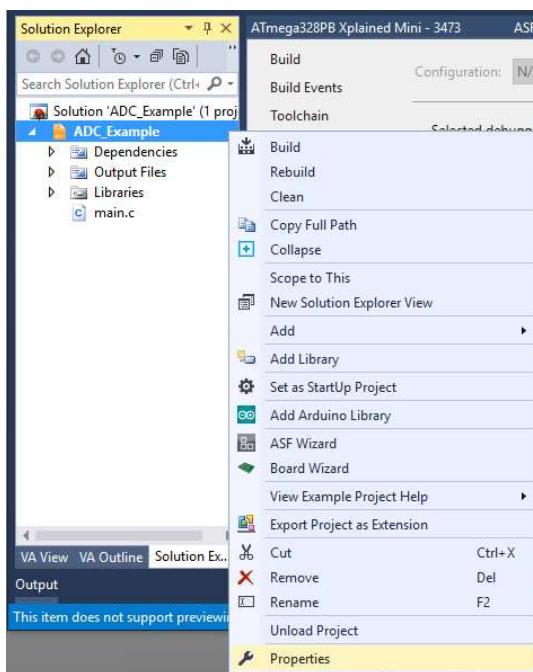
### 3 Task 3 - Build Project

- Select **Build > Build Solution** from the Studio 7 menu to compile the code. You will see a **Build Succeeded** message in the output window. If there are any errors, check `main.c` for any mistakes in entering the program code.



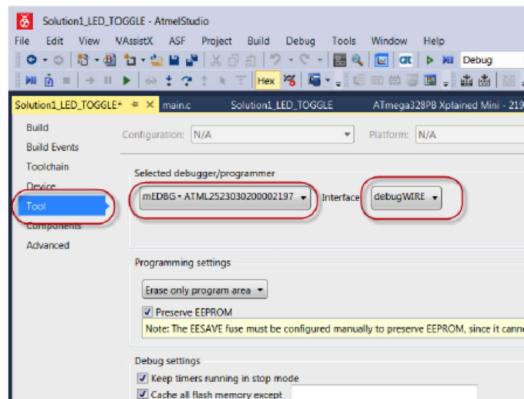
### 4 Task 4 - Programming the Xplained Board

- Connect the Xplained board to the USB port of the computer using the included cable.
- In the **Solution Explorer** area, right-click on the project name and select **Properties**.



(/local--files/8avr:avradc/solution.png)

- Under the **Tool** menu selection, choose the **mEDBG** and **debugWire** as the interface.



(/local--files/8avr:avradc/step42.png)

- Select **Debug > Start Without Debugging** from the Studio 7 menu. The project will build and then program the xplained board with the project code along with debug control.



(/local--files/8avr:avradc/step52.png)

- Studio 7 will show a **Ready** message when the programming is complete.

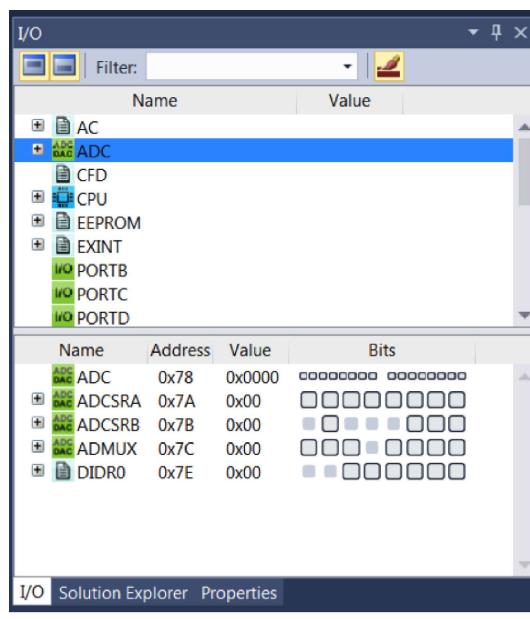


If the Xplained board will not connect, there may be a fuse setting causing this to occur (/boards:debugbrick).

## 5 Task 5 - Debugging

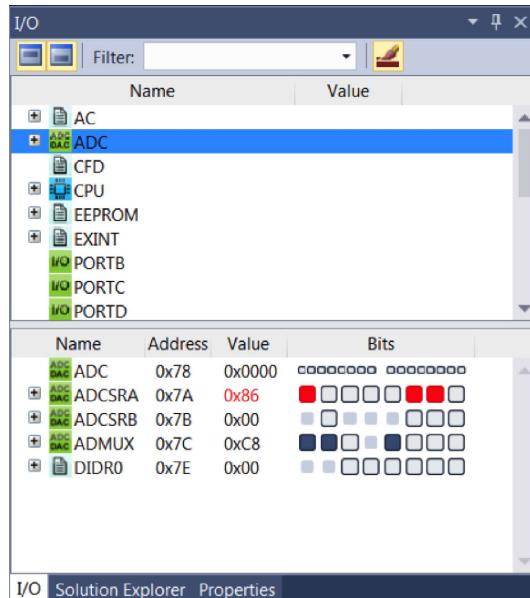
Debugging a device is essential for determining how a program may be running.

- Select **Debug > Start Debugging and Break**. The project will build and the program will be loaded into the Xplained board
- The I/O View window will open up showing the various peripherals
- Click on the **ADC** selection to open the I/O view for the ADC



(/local--files/avr8:avradc/adcddebug.png)

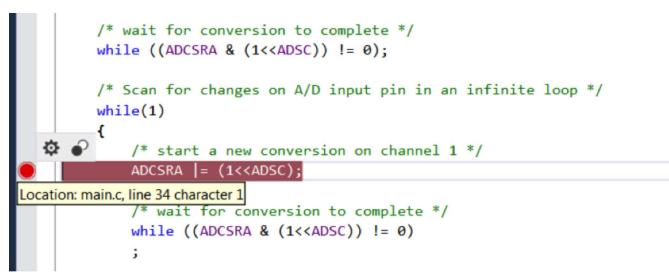
- Select **Debug > Step Over (F10)** to single-step through the program on the Xplained board. Monitor the ADC registers in the I/O View while single-stepping



(/local--files/avr8:avradc/adcddebug2.png)

## 6 Task 6 - Breakpoint and Output

- Click on the **Debug > Break All** from the top menu of Studio 7
- Click on the margin to enable a breakpoint on the command line at the ADCSRA statement within the While loop



(/local--files/avr8:avradc/breakpoint.png)

- Move the mouse over the breakpoint red circle, to be able to see the setting pop-up option (gear symbol) and click on it



(/local--files/8avr:avradc/breakpoint2.png)

- Inside the **Breakpoint Setting Window**, check the **Actions** box. Inside the "Log a message to Output Window" insert the following: **Temperature in C = {Ctemp}** and check the **Continue execution box**. Click on **Close**
- Enter Debug mode by clicking on **Start Debugging and Break** in the top **Debug** menu
- Open the **Output Window** by selecting **Debug > Windows > Output** to see the value of the temperature variable
- Click on **Debug > Continue** and view the temperature in the Output window

## ✿ Results

---

The temperature of the chip is displayed in the output window. Pressing on it with a finger will heat up the reading by a couple of degrees.



(/local--files/8avr:avradc/breakpoint3.png)

### 7 Task 7 - Disable debugWIRE and Close

The **debugWire fuse** needs to be reset in order to program the Xplained board in the future. While still running in debug mode select **Debug > Disable debugWire and Close**. This will release the debugWire fuse.

## Q Analysis

---

Using the ADC is quite easy and the debug feature makes it easy to monitor the results.

## 💡 Conclusions

---

This project can become the basis for future ADC related projects.

# Descripción general de la operación de bajo consumo de AVR® MCU

## Descripción general de bajo consumo

El microcontrolador AVR® de 8 bits proporciona varios modos de reposo y sincronización de reloj controlada por software para adaptar el consumo de energía a los requisitos de la aplicación. Los modos de suspensión permiten que el microcontrolador apague los módulos no utilizados para ahorrar energía. Cuando el dispositivo entra en modo de suspensión, la ejecución del programa se detiene y se utilizan interrupciones o reinicio para reactivar el dispositivo nuevamente. El reloj individual de los periféricos no utilizados se puede detener durante el funcionamiento normal o en suspensión, lo que permite una gestión de energía mucho más precisa que los modos de suspensión por sí solos.

Para llegar a las cifras de potencia más bajas posibles, hay un par de puntos a los que prestar atención. No es solo el modo de suspensión lo que define el consumo de energía, sino también el estado de los pines de E/S, la cantidad de módulos periféricos habilitados, etc.

El consumo de energía es proporcional al voltaje de funcionamiento y, para conservar energía, debe considerar usar el voltaje del sistema más bajo posible. Además, el consumo también es directamente proporcional a la frecuencia del reloj y, si no se utilizan los modos de suspensión, el dispositivo debe funcionar con la frecuencia más baja posible.

### Consejos y trucos para reducir la potencia de un AVR®

- Use el **registro de reducción de energía (PRR0)** para detener el reloj en los periféricos individuales no utilizados, lo que reduce el consumo de energía.

**Name:** PRR0**Offset:** 0x64**Reset:** 0x00**Property:** -

Bit	7	6	5	4	3	2	1	0
Access	PRTWI0	PRTIM2	PRTIM0	PRUSART1	PRTIM1	PRSPI0	PRUSART0	PRADC
Reset	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

(./local--files/8avr:low-power-overview/prp-picture.png)

**Bit 7 – PRTWI0: Power Reduction TWI0**

Writing a logic one to this bit shuts down the TWI 0 by stopping the clock to the module. When waking up the TWI again, the TWI should be re initialized to ensure proper operation.

**Bit 6 – PRTIM2: Power Reduction Timer/Counter2**

Writing a logic one to this bit shuts down the Timer/Counter2 module in synchronous mode (AS2 is 0). When the Timer/Counter2 is enabled, operation will continue like before the shutdown.

**Bit 5 – PRTIM0: Power Reduction Timer/Counter0**

Writing a logic one to this bit shuts down the Timer/Counter0 module. When the Timer/Counter0 is enabled, operation will continue like before the shutdown.

**Bit 4 – PRUSART1: Power Reduction USART1**

Writing a logic one to this bit shuts down the USART by stopping the clock to the module. When waking up the USART again, the USART should be re initialized to ensure proper operation.

**Bit 3 – PRTIM1: Power Reduction Timer/Counter1**

Writing a logic one to this bit shuts down the Timer/Counter1 module. When the Timer/Counter1 is enabled, operation will continue like before the shutdown.

**Bit 2 – PRSPI0: Power Reduction Serial Peripheral Interface 0**

If using debugWIRE On-chip Debug System, this bit should not be written to one. Writing a logic one to this bit shuts down the Serial Peripheral Interface by stopping the clock to the module. When waking up the SPI again, the SPI should be re initialized to ensure proper operation.

**Bit 1 – PRUSART0: Power Reduction USART0**

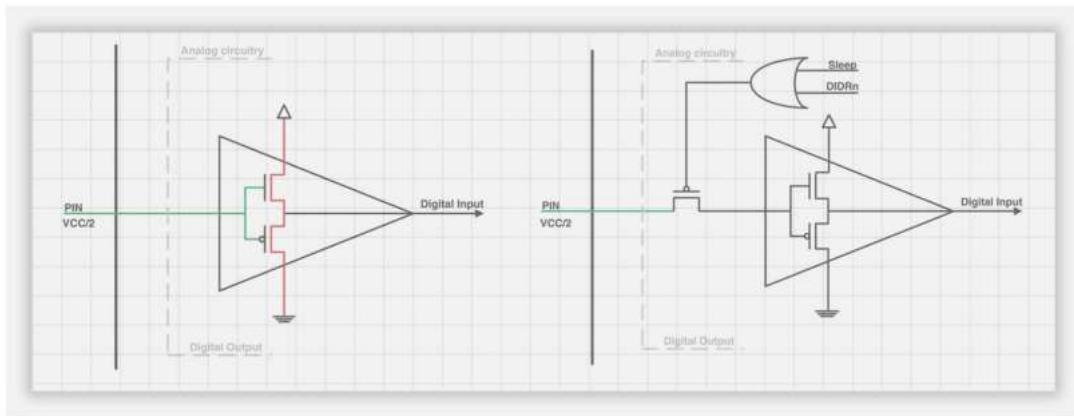
Writing a logic one to this bit shuts down the USART by stopping the clock to the module. When waking up the USART again, the USART should be re initialized to ensure proper operation.

**Bit 0 – PRADC: Power Reduction ADC**

Writing a logic one to this bit shuts down the ADC. The ADC must be disabled before shut down. The analog comparator cannot use the ADC input MUX when the ADC is shut down.

(./local--files/8avr:low-power-overview/prt-description.png)

- Utilice el registro de desactivación de entrada digital (DIDR) para apagar los búferes de entrada digital no utilizados y detener la corriente de fuga.



(/local--files/8avr:low-power-overview/didr-circuit.png)

**Name:** DIDR0  
**Offset:** 0x7E  
**Reset:** 0x00  
**Property:** -

Bit	7	6	5	4	3	2	1	0
Access	ADC7D	ADC6D	ADC5D	ADC4D	ADC3D	ADC2D	ADC1D	ADC0D
Reset	0	0	0	0	0	0	0	0

(/local--files/8avr:low-power-overview/didr0-picture.png)

**Name:** DIDR1  
**Offset:** 0x7F  
**Reset:** 0x00  
**Property:** -

Bit	7	6	5	4	3	2	1	0
Access							AIN1D	AIN0D
Reset							R/W	R/W

**Bit 1 – AIN1D: AIN1 Digital Input Disable**

**Bit 0 – AIN0D: AIN0 Digital Input Disable**

When this bit is written logic one, the digital input buffer on the AIN1/0 pin is disabled. The corresponding PIN Register bit will always read as zero when this bit is set. When an analog signal is applied to the AIN1/0 pin and the digital input from this pin is not needed, this bit should be written logic one to reduce power consumption in the digital input buffer.

(/local--files/8avr:low-power-overview/didr1.png)



Visite la página ([/8avr:low-power-example](#)) de ejemplo de bajo consumo para ver un código de ejemplo sobre cómo reducir el consumo de energía del AVR.

# Proyecto de ejemplo de AVR de baja potencia

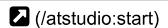
## ⌚ Objetivo

Esta página ilustra varios métodos para configurar un MCU AVR® de 8 bits para que funcione con bajo consumo de energía. Con este ejercicio lograrás:

- Cree un proyecto y agregue un código simple para hacer parpadear un LED
- Ejecutar el proyecto y medir el consumo de energía
- Hacer modificaciones al código para reducir la potencia
- Ejecute el proyecto modificado y observe un consumo de energía reducido

## ☑ Materiales

### Requisitos de Software

Herramienta	Sobre	Instaladores			Instrucciones de instalación
		Windows	Linux	Mac OS X	
Entorno de desarrollo integrado Atmel® Studio	<a href="#">(atstudio:start)</a> 	 <a href="http://studio.download.atmel.com/7.0.1931/as-installer-7.0.1931-full.exe">(http://studio.download.atmel.com/7.0.1931/as-installer-7.0.1931-full.exe)</a>			 <a href="#">(/install:atstudio)</a>

### Requisitos de hardware

- ATmega328PB Mini tablero explicado
- Kit depurador de energía
- Dos cables micro USB
- Tres cables hembra a macho y un cable macho a macho

Herramienta	Sobre
 <a href="http://www.atmel.com/tools/MEGA328PB-XMINI.aspx">(http://www.atmel.com/tools/MEGA328PB-XMINI.aspx)</a>	 Mini kit de evaluación ATmega328PB-Xplained <a href="http://www.atmel.com/tools/MEGA328PB-XMINI.aspx">(http://www.atmel.com/tools/MEGA328PB-XMINI.aspx)</a>

### ATmega328PB Tarjeta explicada

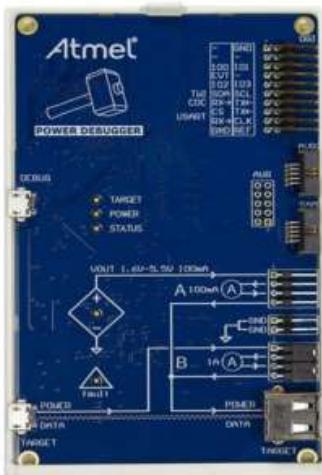
El **mini kit de evaluación ATmega328PB Xplained** es una plataforma de hardware para evaluar el microcontrolador Atmel ATmega328PB. NO se necesita un depurador externo para ejecutar estos ejercicios. El ATmega328PB tiene un depurador incorporado completamente integrado.



(/local--files/8avr:low-power-example/xplained-board.png)

### Kit depurador de energía

Power Debugger es un depurador compatible con CMSIS-DAP que funciona con Atmel Studio v7.0 o posterior. El depurador de energía envía datos de depuración de aplicaciones y medidas de energía en tiempo de ejecución al visualizador de datos.



(/local--files/8avr:low-power-example/power-debug.png)

El Power Debugger tiene dos canales de detección de corriente independientes para medir y optimizar el consumo de energía de un diseño.

El canal 'A' es el canal recomendado para medir con precisión corrientes bajas.

El canal 'B' es el canal recomendado para medir corrientes más altas con una resolución más baja.

## configuración de hardware

### Conexión del depurador de energía a la placa Xplained ATmega328PB.

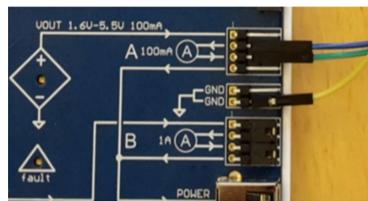
Los puertos de medición de corriente de los canales 'A' y 'B' en la placa Power Debugger se representan con símbolos de amperímetro en la serigrafía. El suministro de voltaje está conectado a la entrada del amperímetro y la carga (objetivo) está conectada a la salida. Con los siguientes pasos, el depurador de energía mide el consumo en el núcleo AVR.

Tabla 1-1 Power Debugger y **ATmega328PB** Explicación de la conexión Mini.

Power Debugger	ATmega328PB Xplained Mini
Port A Input : Blue wire	5V
Port A output: Green wire	VCC
GND : Yellow Wire	GND

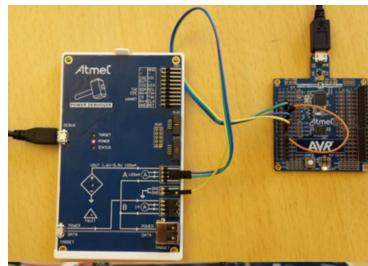
(/local--files/8avr:low-power-example/table.png)

Figura 1-1 Conexión de Power Debugger y **ATmega328PB Xplained Mini**.



(/local–files/8avr:low-power-example/connection.png)

## Configuración de hardware



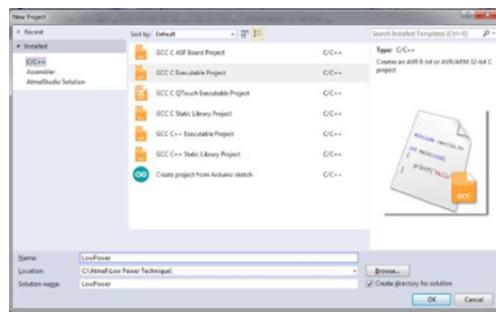
(/local–files/8avr:low-power-example/hw.png)

## Medición de la corriente en modo activo

### Procedimiento

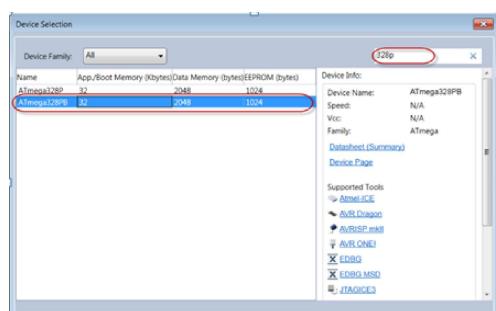
#### A Creación de proyectos

- Abrir Atmel Studio 7
- Seleccione **Archivo > Nuevo > Proyecto**
- Seleccione **GCC C Executable Project** y asignele el nombre **LowPower**.
- Elija una ubicación para guardar el proyecto en su computadora.



(/local–files/8avr:low-power-example/location.png)

- Cuando aparezca la ventana de selección de dispositivo, ingrese 328P en la ventana de búsqueda y luego seleccione Atmega328PB. Haga clic en Aceptar.



(/local–files/8avr:low-power-example/device-

(selection.png)

- Se creará un proyecto que contiene un `ciclo while(1)` vacío en `main()`.



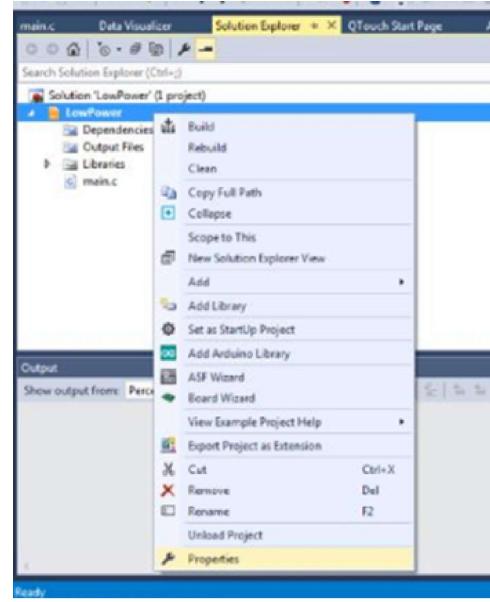
```

1 <font></font>
2 #include <avr/io.h><font></f
3 <font></font>
4 int main(void)<font></font>
5 {<font></font>
6     /* Replace with your app
7     while (1) <font></font>
8     {<font></font>
9         }<font></font>
10    } <font></font>
11 <font></font>

```

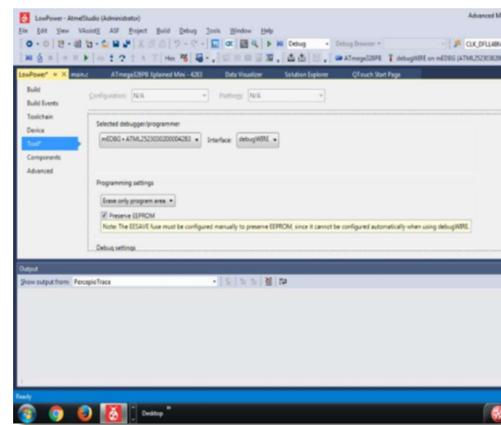
Seleccione **Ver > Explorador** de soluciones en la barra de menú.

En el Explorador de soluciones, haga clic con el botón derecho en el proyecto y seleccione **Propiedades**.



(local--files/8avr:low-power-example/se.png)

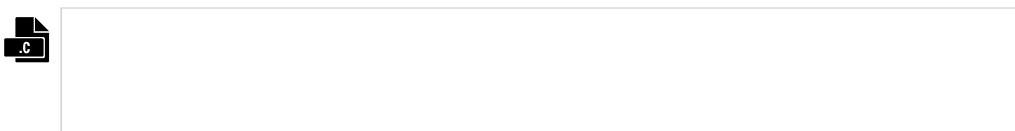
En **Herramienta**, seleccione **mEDBG** y **debugWIRE**



(local--files/8avr:low-power-example/debugwire.png)

## B Agregar código al proyecto

- Agregue las siguientes dos declaraciones en `main()` para llamar a `TMR0_init` y establecer un pin de E/S como salida:



```
<font></font>
TMR0_init();<font></font>
<font></font>
DDRB |= 1<<DDRB5; // Direction o
<font></font>
```

Agregue la rutina de inicialización del temporizador 0

```
<font></font>
***** TM *****
void TMR0_init( void ){<font></font>
<font></font>
// enable timer overflow inter
TIMSK0=(1<<TOIE0) <font></font>
// set timer0 counter initial
TCNT0=0x00;<font></font>
// start timer0 with /1024 pre
TCCR0B = (1<<CS02) | (1<<CS00)
<font></font>
// enable interrupts<font></font>
sei(); <font></font>
}<font></font>
<font></font>
```

- Proporcione la rutina de servicio de interrupción TMR0 al proyecto y haga que el LED parpadee a aproximadamente 1 Hz.

```
<font></font>
uint8_t count; <font></font>
ISR(TIMER0_OVF_vect){<font></font>
<font></font>
count++;<font></font>
if(count==20){<font></font>
count=0; <font></font>
// Toggle LED<font></font>
PORTB=PORTB ^ 0x20;<font></font>
}<font></font>
}<font></font>
```

El programa completo es el siguiente,



```

<font></font>
#include <avr/io.h><font></font>
#include "avr/interrupt.h"<font></font>
<font></font>
  uint8_t   count;<font></font>
/* Timer 0 Interrupt */<font></font><font>
ISR(TIMER0_OVF_vect){<font></font>
<font></font>
  count++;<font></font>
  if(count==20) {<font></font>
    count=0;<font></font>
    // Toggle LED<font></font>
    PORTB=PORTB ^ 0x20;<font></font>
  }<font></font>
}<font></font>
int main(void)<font></font>
{<font></font>
  TMR0_init();<font></font>
<font></font>
  DDRB |= 1<<DDRB5; // Direct
  /* Replace with your applicati
  while (1) <font></font>
  {<font></font>
  }<font></font>
}<font></font>
<font></font>
*****TM*****
*****TM*****
void TMR0_init( void ){<font></font>
<font></font>
  // enable timer overflow inter
  TIMSK0=(1 <<TOIE0) ;<font></fo
<font></font>
  // set timer0 counter initial
  TCNT0=0x00;<font></font>
<font></font>
  // start timer0 with /1024 pre
  TCCR0B = (1 <<CS02) | (1<<CS00
<font></font>
  // enable interrupts<font></fo
  sei(); <font></font>
}<font></font>
}<font></font>
<font></font>

```

## C Verificar las ejecuciones del proyecto

- Compile el proyecto y programe el dispositivo seleccionando **Depurar > Continuar**.



El LED parpadeará si el proyecto funciona correctamente

- Asegúrese de que la depuración del proyecto finalice seleccionando **Depurar > Deshabilitar debugWire** y luego **Cerrar**.

## D Medir el consumo de energía en modo activo

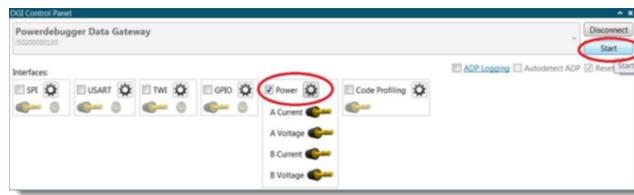
- En Atmel Studio 7, abra el menú **Herramientas > Visualizador de datos**

Power Debugger Data Gateway debe seleccionarse de forma predeterminada en el Panel de control de DGI; selecciónelo si no lo está. Haga clic en **Conectar**.

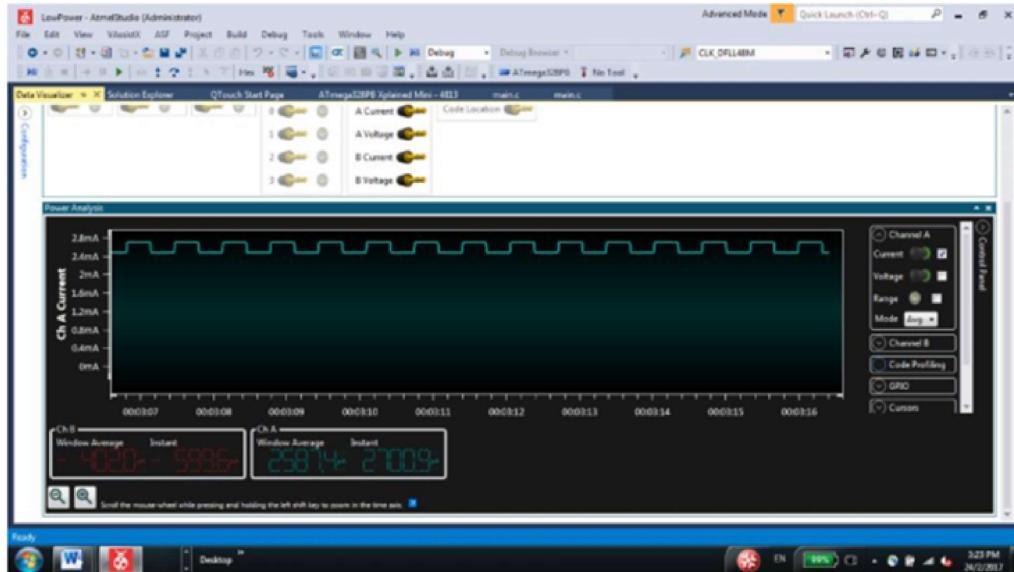


(/local--files/8avr:low-power-example/connect.png)

- Seleccione **Encendido y arranque**



(/local--files/8avr:low-power-example/power-and-start.png)



(/local--files/8avr:low-power-example/power-monitor.png)

Apague la fuente de alimentación de destino a la placa expandida y habilite la medición de energía



(/local--  
files/8avr:low-  
power-  
example/enable-  
power.png)

- Cierre la programación del dispositivo
- Verifique que el consumo medio de corriente sea de unos 2,6 mA

## Reducir el consumo de energía

Existen varios métodos que puede utilizar para reducir el consumo de energía de un microcontrolador AVR®. Este ejemplo reduce el poder por:

- apagar los periféricos no utilizados
- Detener la corriente de fuga en los pines de E/S digital
- Uso del modo de suspensión. Las técnicas de optimización del consumo de energía se implementan en la función `Optimize_power_consumption()` que se llama desde `main()`.

### Deshabilitar los búferes de entrada digital y el comparador analógico

Para los pines de entrada analógica, el búfer de entrada digital debe estar deshabilitado.

El búfer de entrada digital mide el voltaje en el pin y representa el valor como uno o cero lógico. Un nivel de señal analógica cercano a VCC/2 en un pin de entrada puede causar un consumo de corriente adicional significativo. Si el pin se usa como pin analógico, no hay necesidad de saber si el valor digital de la señal analógica sería uno o cero; estos pines digitales deben estar deshabilitados.

## Apague los periféricos no utilizados

Deshabilite los periféricos no utilizados en la aplicación. El **registro de reducción de energía ( PRR0 ) y ( PRR1 )** pueden detener el reloj de periféricos individuales para reducir el consumo de energía. Los recursos utilizados por el periférico permanecen ocupados cuando se detiene el reloj. En la mayoría de los casos, los periféricos deben desactivarse antes de que se detenga el reloj.

1. Incluya el archivo <power.h> en el programa principal **#include <avr/power.h>**
2. Agregue el código a continuación para **optimizar\_el\_consumo\_de\_energía()**.



```
<font></font>
/* Power shutdown to unused periph
PPR0 = 0xDF; <font></font>
PPR1 = 0x3F; <font></font>
<font></font>
```

3. Agregue e **#include** para <wdt.h> a main.c. **#include <avr/wdt.h>**
4. Agregue el siguiente código en **optimize\_power\_consumption()** para desactivar el temporizador de vigilancia.



```
<font></font>
/* Disable interrupts*/ <font></font>
Cli(); <font></font>
Wdt_reset(); <font></font>
MCUSR&= ~(1<<WDRE); <font></font>
<font></font>
```

## Aplicar resistencias pull-up

Los pines no utilizados y desconectados consumen energía. La carga de energía innecesaria de los pines flotantes se puede evitar usando las resistencias pull-up internas del AVR en los pines no utilizados. Los pines del puerto se pueden configurar para ingresar pull-ups configurando el bit **DDxn** en 0 y el bit **PORTxn** en 1. (donde x es PORT B, C, D, E y n es 0 a 7)



```
1 <font></font> ??
2 /* Unused pins set as input
3    DDRB &= 0xE0; <font></font>,
4    DDRC &= 0xC9; <font></font>,
5    DDRD &= 0x03; <font></font>,
6    DDRE &= 0xF3; <font></font>,
7 <font></font>
8    PORTB |= ~0xE0; <font></font>
9    PORTC |= ~0xC9; <font></font>
10   PORTD |= ~0x03; <font></font>
11   PORTE |= ~0xF3; <font></font>
12 <font></font>
```

## Usar la función de suspensión

El modo de suspensión permite que la aplicación ahorre energía al apagar los módulos no utilizados. El AVR proporciona varios modos de suspensión que permiten al usuario adaptar el consumo de energía a los requisitos de la aplicación.

1. Incluya el archivo de encabezado **<avr/sleep.h>** de la biblioteca AVR en la parte superior del archivo.
2. Establezca el modo de suspensión deseado en la función **Optimize\_Power\_consumption()** configurando el microcontrolador para usar el modo de suspensión **SLEEP\_MODE\_PWR\_DOWN** para un consumo mínimo de energía.

```

<font></font>
    /* Set sleep mode */
    set_sleep_mode(SLEEP_MODE_PWR_DOWN);
<font></font>

```

3. Llame a la función **sleep\_mode()** desde **main()** para ingresar al modo de suspensión.

```

1 <font></font>           ? 
2 while(1) {             ? 
3     sleep_mode();       ? 
4 }                       ? 
5 <font></font>           ? 
6 <font></font>           ? 

```

## Uso de la interrupción de cambio de pin

Para despertar el microcontrolador de **SLEEP\_MODE\_PWR\_DOWN**, se utiliza la **interrupción de cambio de pin**. El interruptor de la **mini placa ATmega328PB Xplained** (SW0) está conectado a **PB7**. El pin **PB7** es la fuente de la interrupción por cambio de pin. Haz las siguientes adiciones a **main()**:

```
#include avr/interrupt.h
```

Configuración del bit PCINT7 y registro PCICR:

```

<font></font>
    PCMSK0 |= (1<<PCINT7); //Enable PCINT7
    PCICR |= (1<<PCIE0); //Enable PCIE0
    sei();<font></font>
<font></font>

```

## Agregue la rutina de servicio de interrupción:

Para habilitar las rutinas ISR, el nombre del vector para PCINT7 es ISR (PCINT0\_vect), definido en el archivo de encabezado del dispositivo.

```

<font></font>
ISR (PCINT0_vect) {
<font></font>
    if (!(PINB & (1<<PINB7))) // 
        PORTB |= (1 <<PORTB5); // 
    else
        PORTB &= ~(1<<PORTB5); // 
}
<font></font>

```

## Código completado

Después de realizar los cambios anteriores, el código completo debería ser como el siguiente:

```

<font></font>
/* lowpower2.c<font></font>
 */
<font></font>
//#include <avr/io.h><font></font>
<font></font>
#include <avr/io.h><font></font>
#include "avr/interrupt.h"<font></font>
#include "avr/wdt.h"<font></font>
#include "avr/sleep.h"<font></font>
<font></font>

```

```

/*
 * This example shows how to implement low power consumption
 * on an AVR microcontroller. It demonstrates how to disable
 * digital inputs, analog comparators, unused pins, and
 * watchdog timer. It also shows how to enable sleep mode
 * and how to use a timer overflow interrupt to toggle an LED.
 */

#include <avr/power.h><font></font>
#include <avr/interrupt.h><font></font>
#ifndefinclude <util/delay.h><font></font>
<font></font>
void optimize_power_consumption()
{
    /* Disable digital input buffer
     * DIDR0 = (1 << ADC5D) | (1 << A
     * DIDR0 |= 0xC0 ; /*ADC7D and
    <font></font>
    /* Disable digital input buffer
     * DIDR1 |= (1 << AIN1D) | (1 <<
     * /* Disable Analog Comparator *
     * ACSR |= (1 << ACD);<font></fon
    <font></font>
    /*Power shutdown to unused per
     * PRR0 = 0xDF;<font></font>
     * PRR1 = 0x3F;<font></font>
     /*Unused pins set as input pul
     * DDRB &= 0xE0;<font></font>
     * DDRC &= 0xC9;<font></font>
     * DDRD &= 0x03;<font></font>
     * DDRE &= 0xF3;<font></font>
    <font></font>
    PORTB |=~(0xE0);<font></font>
    PORTC |=~(0xC9);<font></font>
    PORTD |=~(0x03);<font></font>
    PORTE |=~(0xf3);<font></font>
    <font></font>
    /*Watchdog Timer OFF*/<font></font>
    <font></font>
    /* Disable interrupts */<font>
    cli();<font></font>
    /* Reset watchdog timer */<font>
    wdt_reset();<font></font>
    /* Clear WDRF in MCUSR */<font>
    MCUSR &= ~(1<<WDRF);<font></fo
    /* Turn off WDT */<font></font>
    WDTCSR = 0x00;<font></font>
    <font></font>
    /* Set sleep mode */<font></font>
    set_sleep_mode(SLEEP_MODE_PWR_
    <font></font>
    }<font></font>
    <font></font>
    ****
    TM
    ****
void TMR0_init( void )
{
    // enable timer overflow inter
    TIMSK0=(1<<TOIE0) ;<font></fon
    <font></font>
    // set timer0 counter initial
    TCNT0=0x00;<font></font>
    <font></font>
    // start timer0 with /1024 pre
    TCCR0B = (1<<CS02) | (1<<CS00)
    <font></font>
    // enable interrupts<font></fo
    sei(); <font></font>
    <font></font>
} <font></font>
    uint8_t count;<font></font>
/* Timer 0 Interrupt */<font></fon
ISR(TIMER0_OVF_vect){<font></font>
    <font></font>
    count++;<font></font>
    if(count==20){<font></font>
        count=0;<font></font>
        // Toggle LED<font></font>
        PORTB=PORTB ^ 0x20;<font><
    }<font></font>
}<font></font>
<font></font>
ISR (PCINT0_vect)<font></font>
{<font></font>
    if (! (PINB & (1<<PINB7))) //
    {<font></font>
        PORTB |= (1<<PORTB5); // T
    }<font></font>
    else<font></font>
    {<font></font>
        PORTB &= ~(1<<PORTB5); // T
    }<font></font>
}<font></font>
<font></font>
int main(void)
{
    TMR0_init();<font></font>
    <font></font>
    DDRB |= 1<<DDRB5; // Directi
    DDRB &= ~(1<<DDRB7); //Set PORT
    <font></font>
    optimize_power_consumption();<
}

```

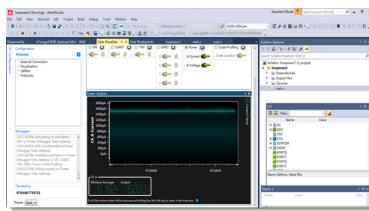
```

<font></font>
    PCMSK0 |= (1<<PCINT7); //Enable
    PCICR |= (1<<PCIE0); //Enable
    sei();<font></font>
<font></font>
    //set_sleep_mode(SLEEP_MODE_PWR
    //sleep_enable();<font></font>
    //sleep_cpu();<font></font>
<font></font>
    /* Replace with your application
    while (1) <font></font>
    {<font></font>
        sleep_mode();<font></font>
    }<font></font>
<font></font>

```

## Programe la aplicación y mida el consumo de energía.

1. Programe el código seleccionando **Depurar > Continuar**.
2. Espere hasta que la aplicación esté programada y el mensaje en la parte inferior de la ventana aparezca como **En ejecución**.
3. Salga del depurador seleccionando **Depurar > Deshabilitar debugWire** y luego **Cerrar**.
4. Abra la **ventana** del Visualizador de datos y verifique el consumo de energía como se muestra a continuación.



(/local–files/8avr:low-power-example/one-five.png)

El consumo de corriente observado debería ser de alrededor de 1,52 mA.

- Establezca el interruptor de alimentación objetivo apagado: **Herramientas > Programación de dispositivos > Configuración de herramientas**



(/local–files/8avr:low-power-example/threema.png)

El consumo de corriente debería haberse reducido a aproximadamente 20,7  $\mu$ A.



Tenga en cuenta que el temporizador no activa el dispositivo desde el modo SLEEP. La aplicación está configurada para salir del modo de suspensión cuando se produce la **interrupción de cambio de pin**.

## 💡 Conclusiones

Este ejemplo demostró varias técnicas diferentes para reducir el consumo de energía de una aplicación AVR. El consumo de energía se puede reducir significativamente mediante:

- Diseño inteligente
- Uso de los modos de suspensión
- Desactivación de periféricos no utilizados

Este ejemplo usó el **visualizador de datos Atmel Studio 7** y la placa de **depuración de energía** para medir la energía.

# Capacitación sobre ADC y optimización de energía para microcontroladores tinyAVR® y megaAVR®

## Introducción:

Este tutorial contiene cinco aplicaciones prácticas para la conversión de datos ADC, con el consumo de corriente medido para cada aplicación. El tutorial comienza con una sencilla aplicación de conversión de ADC. En las siguientes aplicaciones, se introducen diferentes técnicas para demostrar cómo se puede reducir el consumo de corriente en los microcontroladores de las series **tinyAVR® 0 y 1** y **megaAVR® 0**.

Este tutorial también demuestra cómo usar Atmel START para comenzar con el desarrollo de aplicaciones ADC de dispositivos AVR®. Las aplicaciones ADC se han desarrollado paso a paso en Atmel Studio. Este tutorial se ha desarrollado en el kit de evaluación ATtiny817 Xplained Pro, pero debería ser aplicable para todos los dispositivos tinyAVR 0- y 1-series, y megaAVR 0-series.

Los proyectos de solución para cada una de las asignaciones están disponibles en el navegador de ejemplo Atmel START (<http://start.atmel.com/#examples>).

En la categoría 'Primeros pasos', busque ADC y solución de optimización de energía (1-5).

Los enlaces directos a los proyectos de ejemplo relevantes se proporcionan en las descripciones de tareas a continuación.

- ADC y optimización de energía: manual tutorial práctico  
(<http://ww1.microchip.com/downloads/en/DeviceDoc/40002008A.pdf>)

## Requisitos previos de hardware

- Kit de evaluación ATtiny817 Xplained Pro
- Cable micro-USB (Tipo-A/Micro-B)
- un potenciómetro
- Tres cables macho a hembra
- conexión a Internet

## Requisitos previos del software

- Estudio Atmel 7.0
- Navegador web. La lista de navegadores compatibles se puede encontrar aquí: START navegadores compatibles (<http://start.atmel.com/static/help/index.html?GUID-51435BA6-0D59-4458-A413-08A066F6F7CA>)

**Tiempo estimado de finalización: 120 minutos**

## Tarea 1: Conversión de ADC con la aplicación de impresión USART (<http://alexandria.atmel.com/keyword/AVR.TRAINING.ADCPOWER.ASSIGN1/redirect>)

En esta tarea, se utiliza Atmel Studio para desarrollar una aplicación utilizando controladores ADC y USART de Atmel START. El ADC está configurado para funcionar en modo de conversión única y se conecta un potenciómetro al pin de entrada del ADC para estudiar la funcionalidad del ADC. Los datos del ADC se envían a través de USART al terminal integrado en el visualizador de datos de Atmel Studio. En Data Visualizer, el consumo actual de la aplicación se analiza utilizando el Power Analyzer integrado.



Solución  
([http://start.atmel.com/#example/Atmel%3AADC\\_and\\_Power\\_Optimization%3A1.0.0%3A%3AAApplication%3AADC\\_and\\_Power\\_Optimization\\_Solutio](http://start.atmel.com/#example/Atmel%3AADC_and_Power_Optimization%3A1.0.0%3A%3AAApplication%3AADC_and_Power_Optimization_Solutio))

de

la

tarea

## Tarea 2: RTC interrumpe los disparadores ADC y USART Imprimir (<http://alexandria.atmel.com/keyword/AVR.TRAINING.ADCPOWER.ASSIGN2/redirect>)

En esta asignación, se utiliza el módulo Contador en tiempo real (RTC). La interrupción de desbordamiento de RTC se utiliza para activar una conversión de ADC cada medio segundo. La interrupción ADC Result Ready (RESRDY) activa una impresión del resultado ADC en el terminal USART. Cuando la interrupción de desbordamiento de RTC no se activa, el dispositivo se mantiene en modo de suspensión en espera para reducir el consumo de energía. Atmel START se utiliza para agregar el módulo RTC y configurar los controladores RTC, ADC, CPUINIT y SLEEPCTRL. Posteriormente, se regenera un proyecto de Atmel Studio.



Solución de la tarea  
([http://start.atmel.com/#example/Atmel%3AADC\\_and\\_Power\\_Optimization%3A1.0.0%3AAApplication%3AADC\\_and\\_Power\\_Optimization\\_Solutio](http://start.atmel.com/#example/Atmel%3AADC_and_Power_Optimization%3A1.0.0%3AAApplication%3AADC_and_Power_Optimization_Solutio)

## Tarea 3: Optimización de energía en pines de E/S (<http://alexandria.atmel.com/keyword/AVR.TRAINING.ADCPOWER.ASSIGN3/redirect>)

En esta asignación, el búfer de entrada digital en los pines de E/S está deshabilitado para reducir el consumo de corriente. El consumo de corriente se reduce aún más cuando el pin USART TX se configura como un pin de alta impedancia durante el período sin transmisión de datos. Aquí se utilizan los mismos controladores y configuraciones de la asignación anterior. Atmel Studio se utiliza para desarrollar aún más el código.



Solución de la tarea  
([http://start.atmel.com/#example/Atmel%3AADC\\_and\\_Power\\_Optimization%3A1.0.0%3AAApplication%3AADC\\_and\\_Power\\_Optimization\\_Solutio](http://start.atmel.com/#example/Atmel%3AADC_and_Power_Optimization%3A1.0.0%3AAApplication%3AADC_and_Power_Optimization_Solutio)

## Tarea 4: Conversión de ADC usando el modo de comparación de ventana (<http://alexandria.atmel.com/keyword/AVR.TRAINING.ADCPOWER.ASSIGN4/redirect>)

En esta asignación, la interrupción de resultado listo de ADC se reemplaza por la interrupción de WCMP de ADC para activar una transmisión USART. En este caso, el resultado de ADC, que está por debajo del valor umbral de la ventana de ADC, activa la transmisión USART. Los resultados de ADC, que están por encima del valor de umbral de la ventana, se ignoran y no activan ninguna transmisión USART. Atmel START se utiliza para reconfigurar el módulo ADC y el proyecto Atmel Studio se actualiza con la nueva configuración.



Solución de la tarea  
([http://start.atmel.com/#example/Atmel%3AADC\\_and\\_Power\\_Optimization%3A1.0.0%3AAApplication%3AADC\\_and\\_Power\\_Optimization\\_Solutio](http://start.atmel.com/#example/Atmel%3AADC_and_Power_Optimization%3A1.0.0%3AAApplication%3AADC_and_Power_Optimization_Solutio)

## Tarea 5: Sistema de eventos (EVSYS) utilizado para reemplazar el controlador de interrupciones RTC (<http://alexandria.atmel.com/keyword/AVR.TRAINING.ADCPOWER.ASSIGN5/redirect>)

En esta asignación, el sistema de eventos con la señal de evento de desbordamiento de RTC, en lugar de la interrupción de desbordamiento de RTC, se utiliza para activar una conversión ADC. El sistema de eventos permite la señalización directa de periférico a periférico. Permite que un cambio en un periférico (el generador de eventos) active acciones en otros periféricos (los usuarios de eventos) a través de canales de eventos sin usar la CPU. Una ruta de canal puede ser asíncrona o síncrona con el reloj principal.



Solución de la tarea  
([http://start.atmel.com/#example/Atmel%3AADC\\_and\\_Power\\_Optimization%3A1.0.0%3AAApplication%3AADC\\_and\\_Power\\_Optimization\\_Solutio](http://start.atmel.com/#example/Atmel%3AADC_and_Power_Optimization%3A1.0.0%3AAApplication%3AADC_and_Power_Optimization_Solutio)

## Resumen

Este tutorial contiene cinco aplicaciones prácticas que realizan la conversión de datos ADC, con el consumo de corriente medido para cada aplicación. Comienza con una sencilla aplicación de conversión ADC y se introducen diferentes técnicas para demostrar cómo se puede reducir el consumo de corriente. Esta es una base útil para

desarrollar futuras aplicaciones ADC con requisitos específicos de consumo de corriente.

# Proyecto de ejemplo de fuentes de restablecimiento de AVR®

## Objetivo

Este proyecto pasa por varias condiciones de reinicio diferentes: reinicio de encendido (POR), reinicio de apagado (BOR) y tiempo de espera del temporizador de vigilancia (WDT), y muestra cómo funciona cada uno en una placa Xplained de 328 PB . Se muestra que algunos circuitos externos producen una entrada de voltaje variable, pero también funcionará una fuente de alimentación ajustable.

Para obtener más detalles sobre las **fuentes de restablecimiento de AVR®**, visite [Descripción general de las fuentes de restablecimiento de AVR.](#)

## Materiales

### Herramientas de hardware (opcional )

Herramienta	Sobre	Compra
 Mini kit de evaluación ATmega328PB Xplained		

### Herramientas de software

Herramienta	Instaladores			Instrucciones de instalación
	 ventanas	 linux	 Mac OS X	
 Entorno de desarrollo integrado Atmel® Studio				

### Archivos de ejercicios

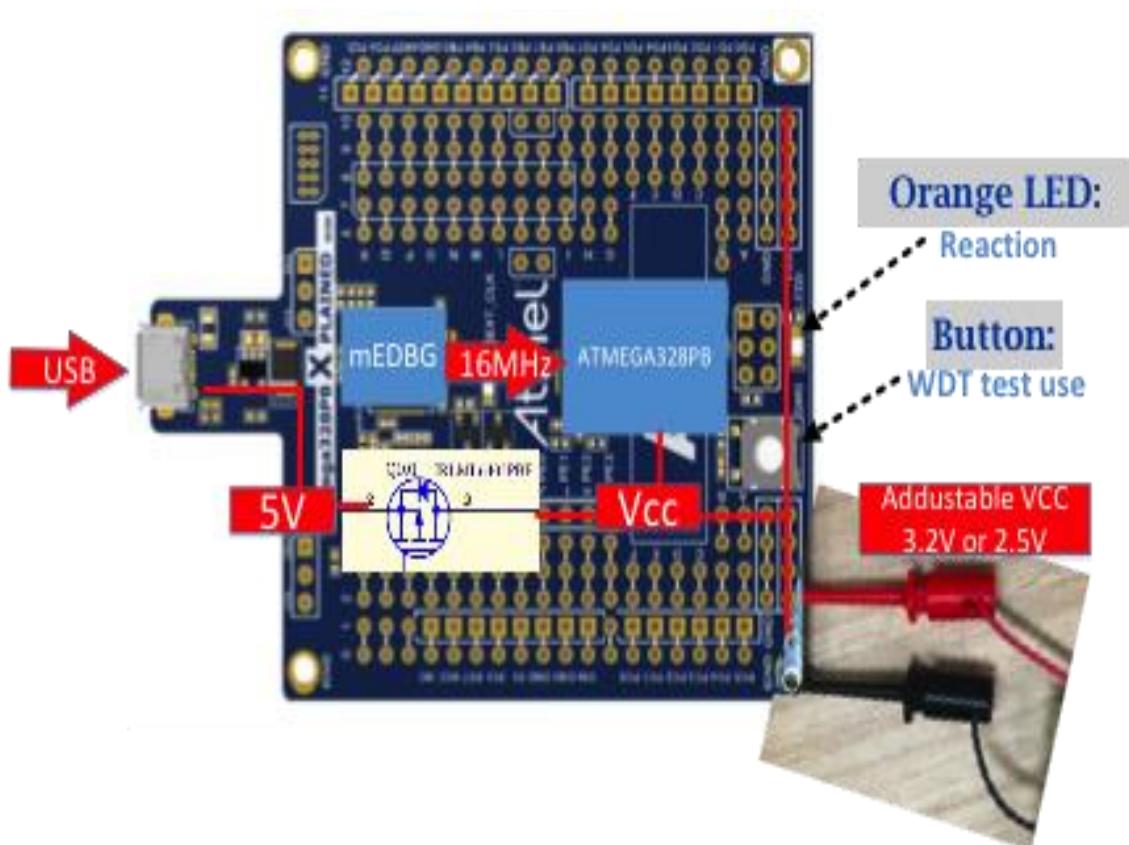
Expediente	Descargar			Instrucciones de instalación
	 ventanas	 linux	 Mac OS X	
 Archivos de proyecto y fuente				

## Archivos adicionales

archivos

 [Guía del usuario de la miniplaca Xplained 328PB](#)

## Diagrama de conexión

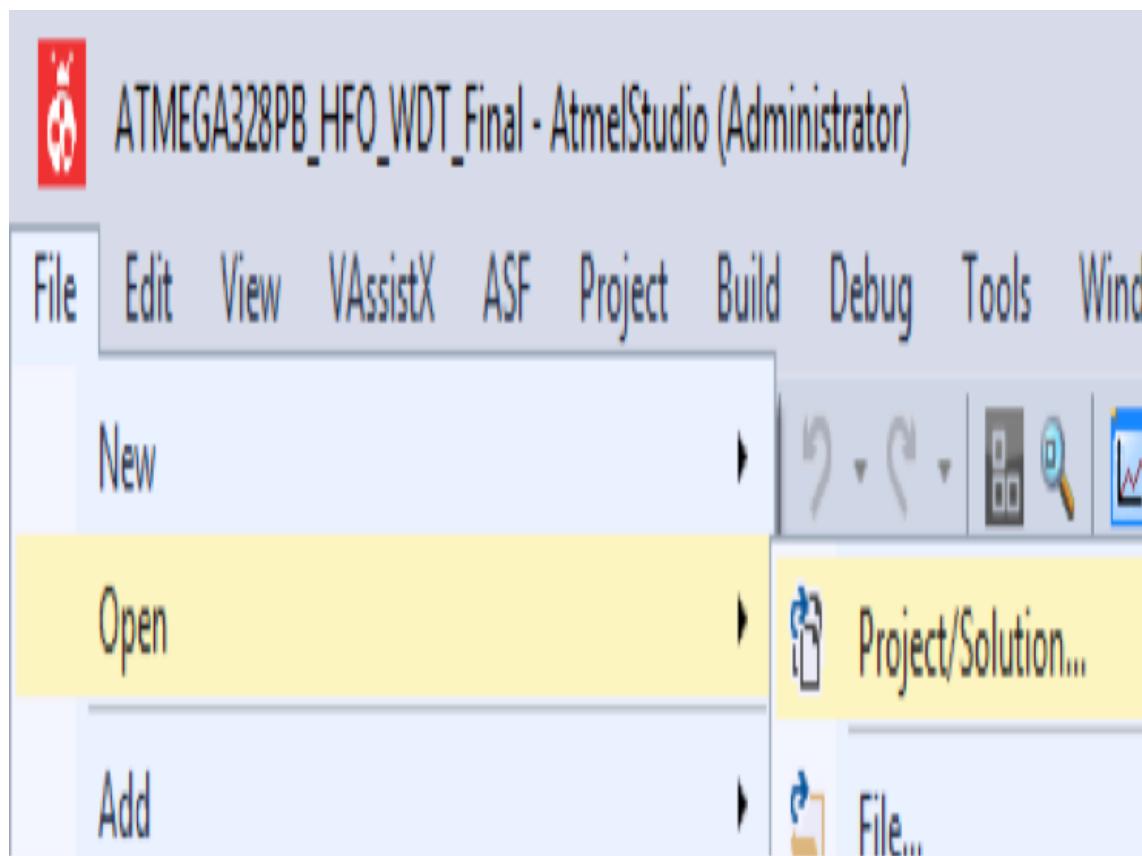


## Procedimiento

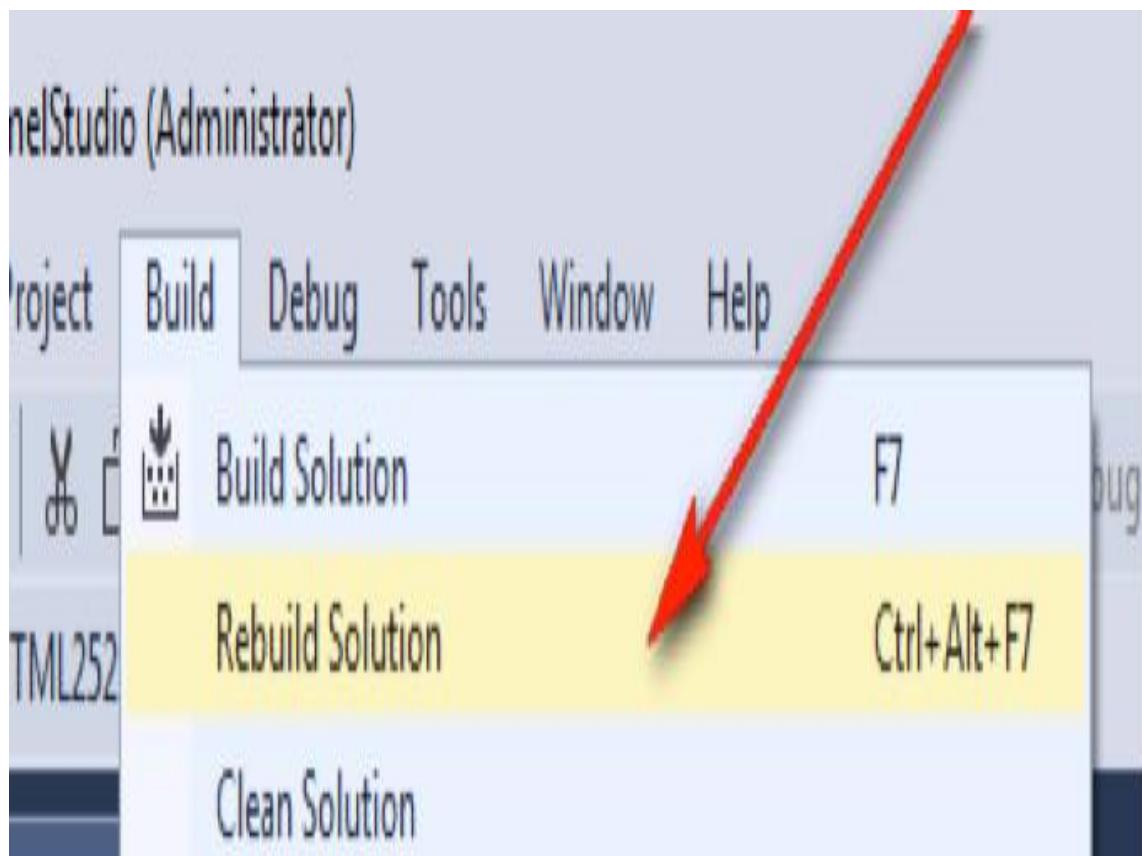
1

### Tarea 1: abrir y compilar el proyecto

- Descargue los archivos fuente del proyecto de la sección anterior Archivos de ejercicios y descomprimalos en su computadora.
- Abrir Atmel Studio 7
- Seleccione **Archivo > Abrir > Proyecto/Solución**



- Seleccione ATMEGA328PB\_HFO\_WDT\_Final.atsln de los archivos de proyecto descargados.
- En el menú Generar, seleccione 'Reconstruir solución'.

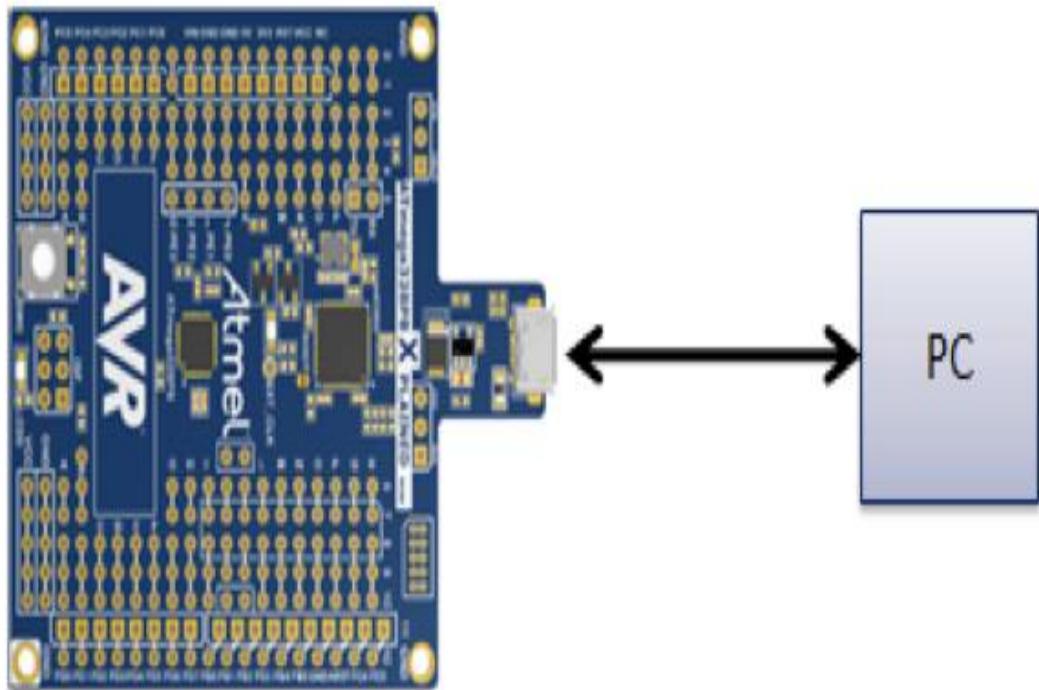


Seleccione 'GCC C Executable Project' y asígnele el nombre `Project1` . Elija una ubicación para guardar el proyecto en su computadora.

**2**

## Tarea 2: preparación de la placa

Asegúrese de que el cable USB esté conectado entre la placa y la PC.



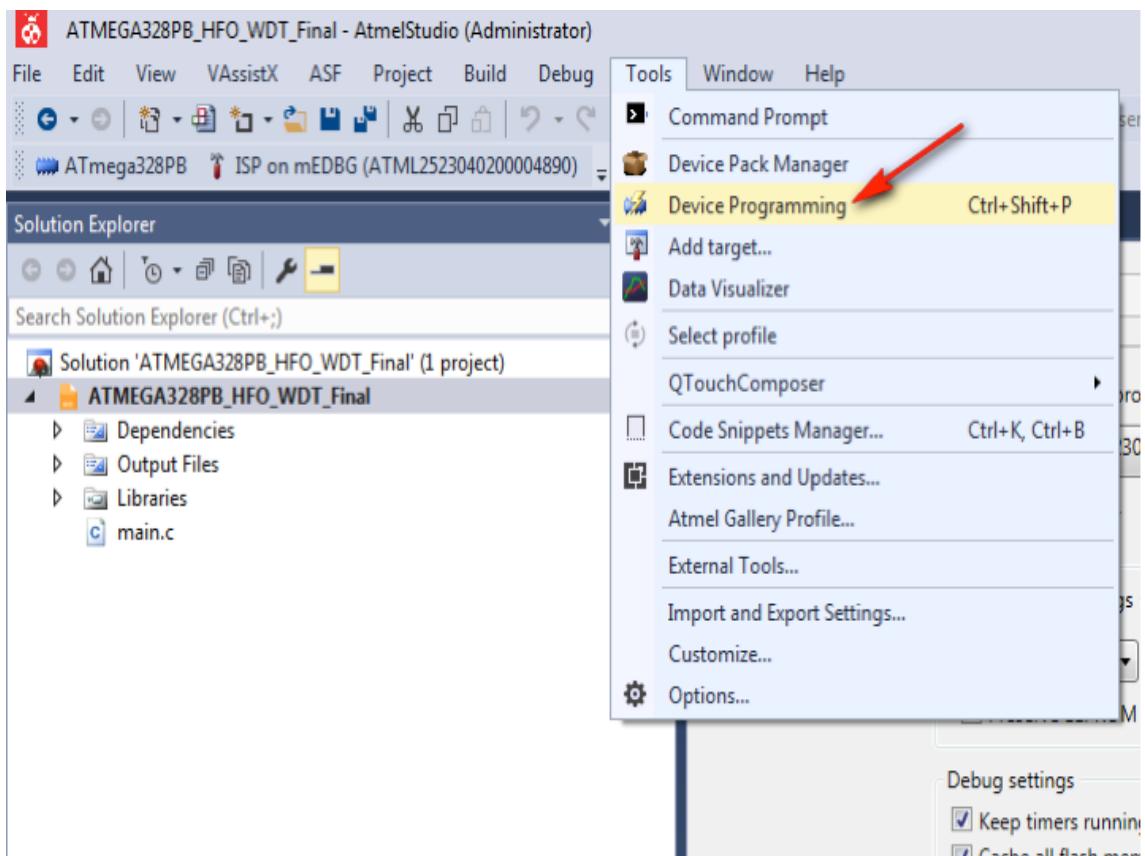
**3**

### Tarea 3 - Configuración del programador

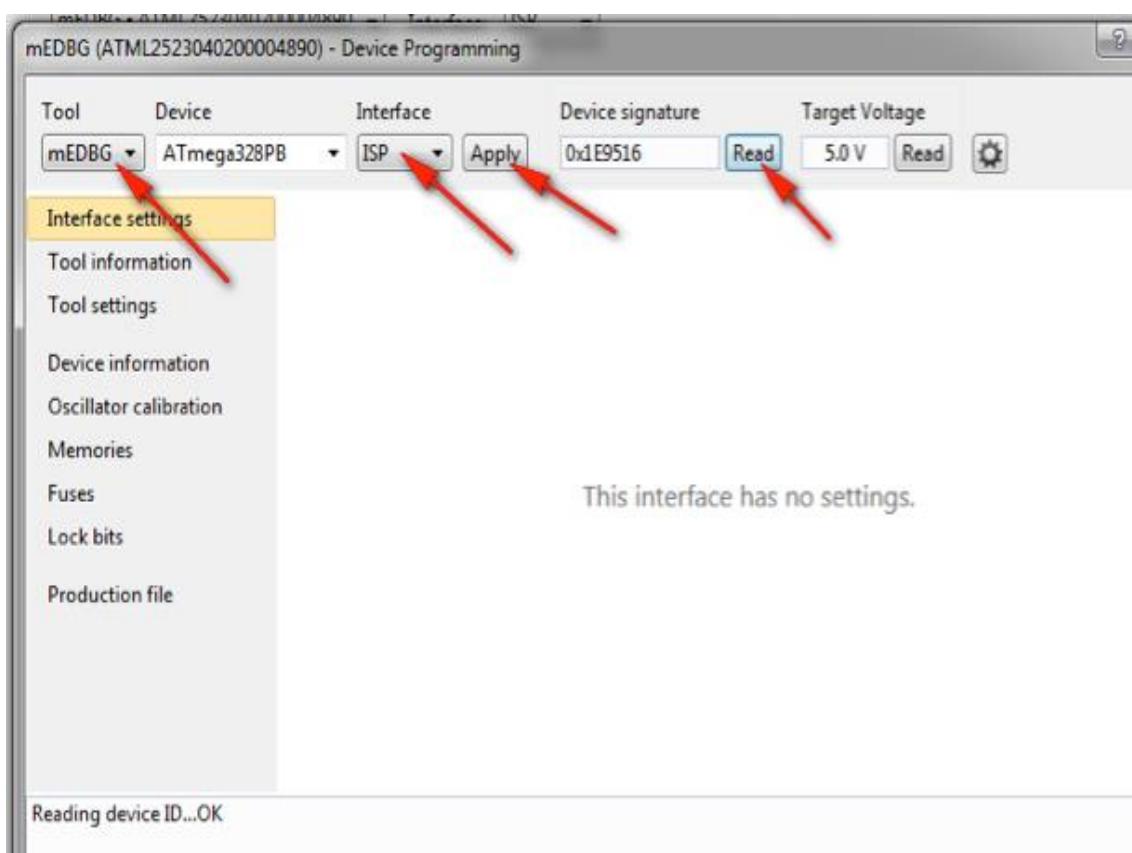
Si debugWire está habilitado, desactívelo.

[Deshabilitar debugWire \(DWEN\) ►](#)

- Haga clic en **Herramientas > Programación de dispositivos** :



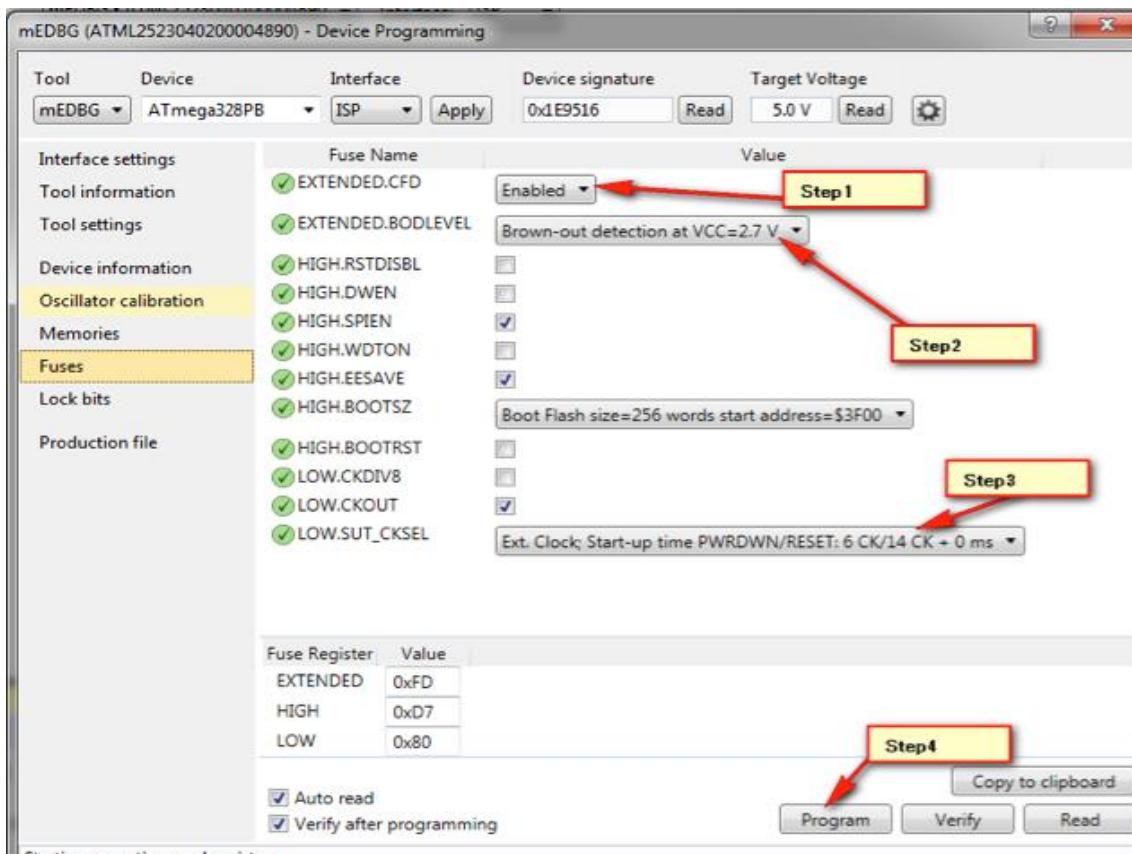
- Haga clic en los botones que están marcados como flechas rojas en la captura de pantalla.



**4**

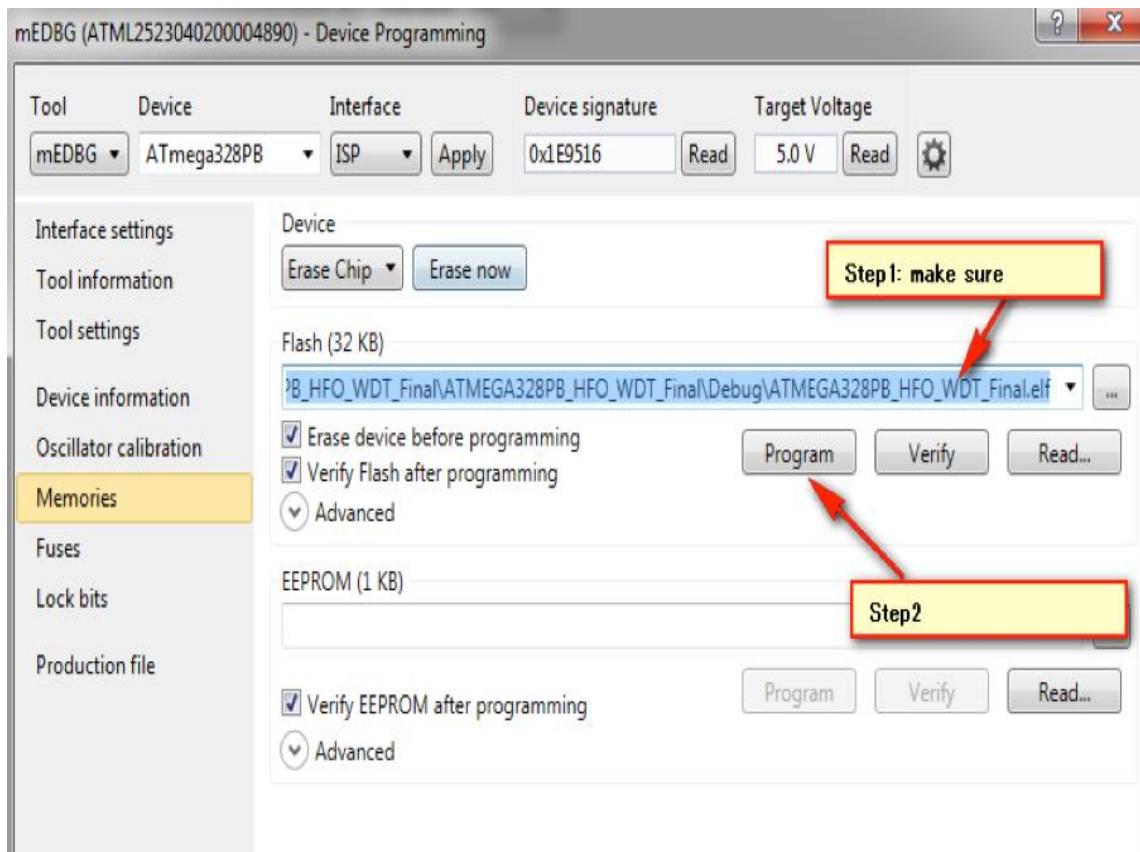
## Tarea 4 - Programación de los bits de fusible

- Seleccione las opciones de 'Fusibles' en la barra de opciones de la izquierda y establezca los bits como se muestra en la figura, lo que habilitará CFD, umbral de DBO como 2,7 V y reloj externo.

**5**

## Tarea 5 - Programación de dispositivos

- Seleccione las opciones de 'Memorias' en la barra de opciones de la izquierda y termine los pasos como se muestra en la figura, que programa el archivo binario compilado en ATMEGA328PB.



## 6

### Tarea 6 - Código del proyecto

Aquí está el código completo al que hacemos referencia. También puedes descargarlo en la Sección de Ejercicios.

```
/*
 * ATMEGA328PB_HFO_WDT_Final.c
 *
 * Created: 2017/03/08 17:15:35
 * Author : A17582
 */
```

```
#define F_CPU 8000000UL
```

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/wdt.h>
#include <util/delay.h>
```

```

//initialize watchdog
void WDT_Init(void)
{
    //disable interrupts
    cli();

    WDTCSR = (1<<WDCE) | (1<<WDE);           // Enable configuration change.

    WDTCSR = (1<<WDIE) |                         // Enable Watchdog
    Interrupt Mode.

    (1<<WDCE) | (1<<WDE) |                      // Enable Watchdog
    System Reset Mode if unintentionally enabled.

    (0<<WDP3) | (1<<WDP2) | (1<<WDP1) | (1<<WDP0); // Set Watchdog Ti
    meout period to 4.0 sec.

    //Enable global interrupts
    sei();
}

//Watchdog timeout ISR
ISR(WDT_vect)
{
    //Burst of 0.1Hz pulses
    for (uint8_t i=0;i<4;i++)
    {
        //LED OFF
        PORTB &= ~(1 << PINB5);                  // Set PORTB5 Low
        _delay_ms(80);

        //LED ON
        PORTB |= (1 << PINB5);                  // Set PORTB5 On
        _delay_ms(20);
    }
}

#define BORFbit 2

```

```

#define PORFbit 0

int main(void)
{
    unsigned char i;
    DDRB |= (1 << PINB5); // Set PORTB5 as output ,
    DDRB &= ~(1<<PINB7); //Set PORTB7 as input

    if(MCUSR & 1 ) {
        MCUSR=0;
        for ( i=0;i<4;i++)
        {
            //LED OFF
            PORTB &= ~(1 << PINB5); // Set PORTB5 Low
            _delay_ms(300);
            //LED ON
            PORTB |= (1 << PINB5); // Set PORTB5 On
            _delay_ms(300);
        }
    }

    else if(MCUSR & 4) {
        MCUSR=0;
        for (i=0;i<8;i++)
        {
            //LED OFF
            PORTB &= ~(1 << PINB5); // Set PORTB5 Low
            _delay_ms(100);
            //LED ON
            PORTB |= (1 << PINB5); // Set PORTB5 On
            _delay_ms(100);
        }
    }
}

```

```

else if(MCUSR & 8) {
    MCUSR=0;
    WDT_Init();
    for (i=0;i<8;i++)
    {
        wdت_reset();
        //LED OFF
        PORTB &= ~(1 << PINB5);           // Set PORTB5 Low
        _delay_ms(20);
        //LED ON
        PORTB |= (1 << PINB5);          // Set PORTB5 On
        _delay_ms(80);
    }
    MCUSR=0;
}

//initialize watchdog
WDT_Init();

//delay to detect reset
//_delay_ms(500);

while(1){

    PORTB |= (1 << PINB5);           // Set PORTB5 high
    _delay_ms(250);
    _delay_ms(250);
    _delay_ms(250);
    _delay_ms(250);

    PORTB &= ~(1 << PINB5);         // Set PORTB5 Low
    _delay_ms(250);
    _delay_ms(250);
}

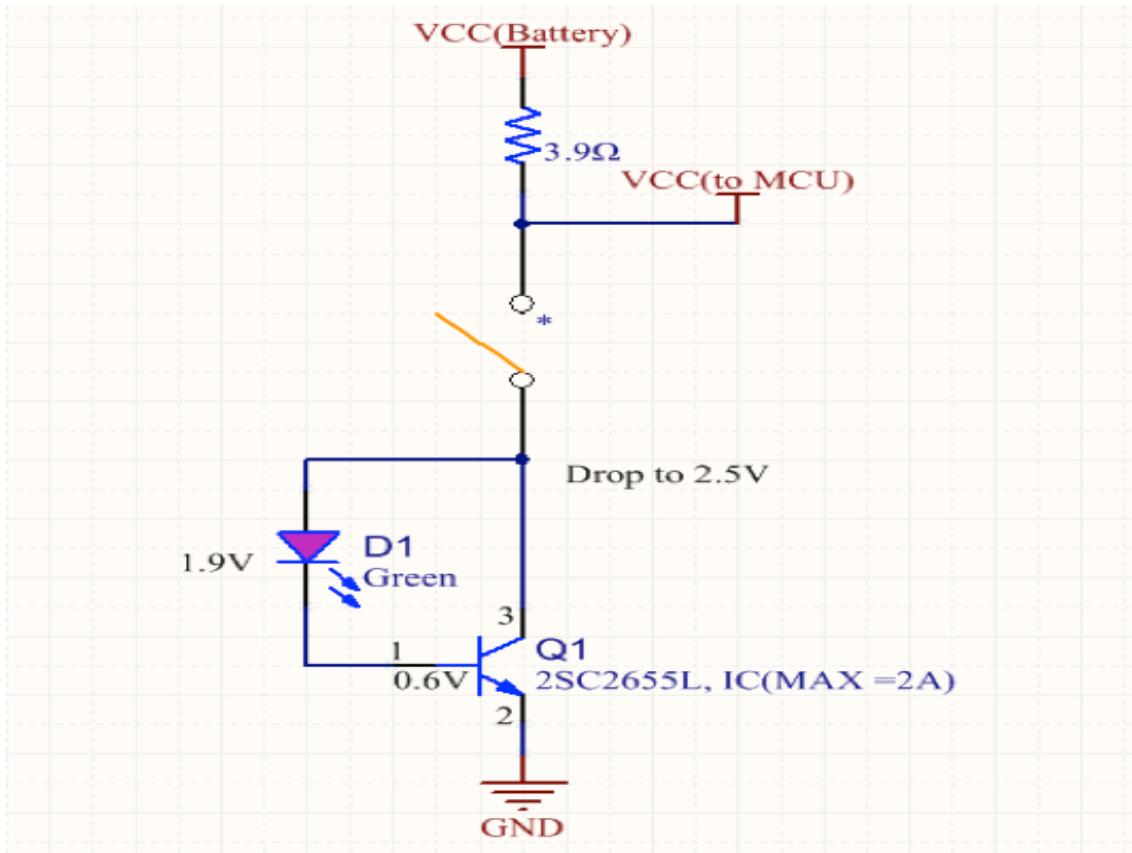
```

```
_delay_ms(250);  
  
_delay_ms(250);  
  
if((PINB&1<<PINB7)==0){  
    PORTB |= (1 << PINB5); // Set PORTB5 high  
    _delay_ms(500);  
    _delay_ms(500);  
    _delay_ms(500);  
    _delay_ms(500);  
    _delay_ms(500);  
    _delay_ms(500);  
    _delay_ms(500);  
    _delay_ms(500);  
    _delay_ms(500);  
    _delay_ms(500);  
}  
wdt_reset();  
}  
}
```

## 7

### Tarea 7: reinicio del circuito de prueba

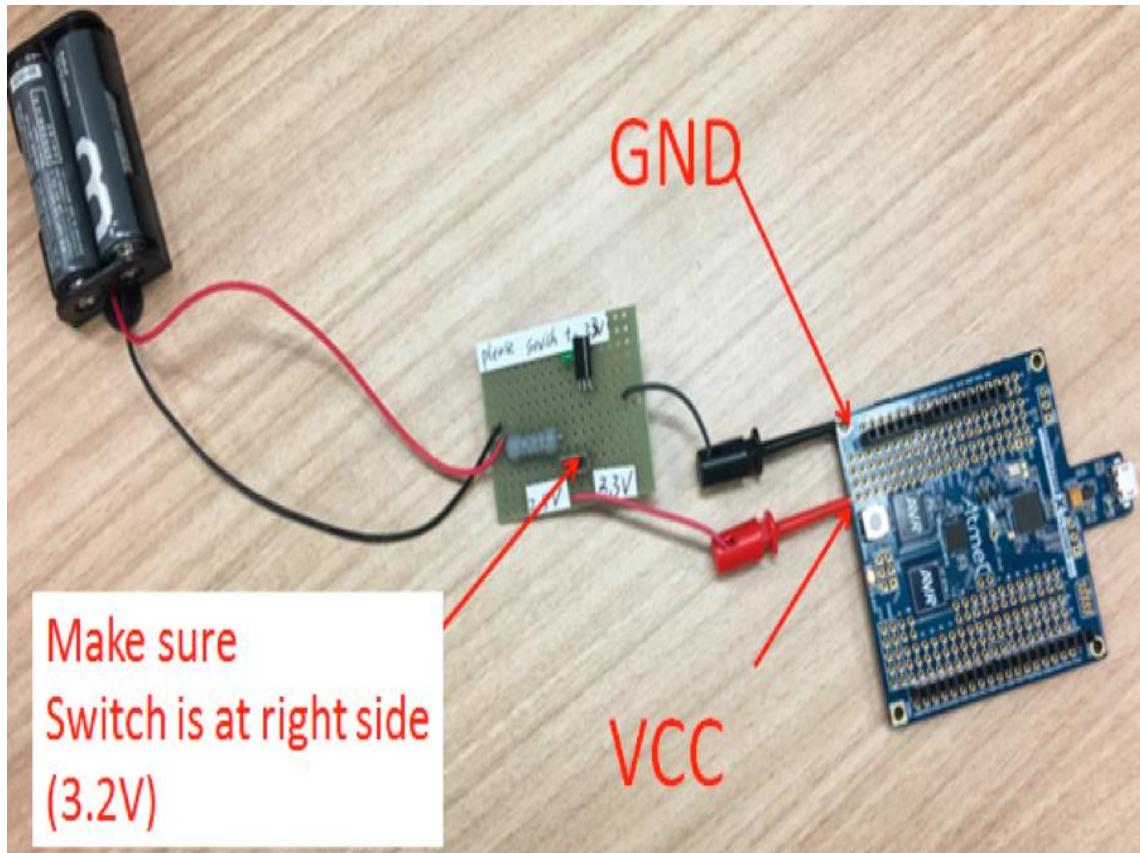
- Construya el circuito de prueba de reinicio que se muestra en el esquema.



7

## Tarea 8 - Prueba POR

- Saque el cable USB
- Asegúrese de que el interruptor del cable VCC ajustable externo esté en el lado derecho (3,2 V)
- Sujete el cable VCC ajustable externo a la placa



- Resultado: el LED naranja responderá parpadeando (0,3 ms, cuatro veces) según la sección de código a continuación porque se detectó el restablecimiento de POR.

```

int main(void)
{
    unsigned char i;
    DDRB |= (1 << PINB5); // Set PORTB5 as output ,
    DDRB &= ~(1<<PINB7); //Set PORTB7 as input

    if(MCUSR & 1 ){ //POR detected
        MCUSR=0;
        for ( i=0;i<4;i++)
        {
            //LED OFF
            PORTB &= ~(1 << PINB5); // Set PORTB5 Low
            _delay_ms(300);
            //LED ON
            PORTB |= (1 << PINB5); // Set PORTB5 On
            _delay_ms(300);
        }
    }
    else if(MCUSR & 4) {
        MCUSR=0;
    }
}

```

9

## Tarea 9 - Prueba BOR

- Despues de la prueba POR, deslice el interruptor del cable VCC ajustable externo hacia el lado izquierdo (2,5 V)
- Deslice el interruptor del cable VCC ajustable externo hacia el lado derecho (3,2 V)
- Resultado: el LED naranja responderá parpadeando (0,1 ms, ocho veces) segun el código resaltado a continuación porque se detectó el restablecimiento de BOR.

```

else if(MCUSR & 4) { //BOR reset detected
    MCUSR=0;
    for (i=0;i<8;i++)
    {
        //LED OFF
        PORTB &= ~(1 << PINB5); // Set PORTB5 Low
        _delay_ms(100);
        //LED ON
        PORTB |= (1 << PINB5); // Set PORTB5 On
        _delay_ms(100);
    }
}
else if(MCUSR & 8) { // WDT reset detected
    MCUSR=0;
    WDT_Init();
    for (i=0;i<8;i++)
    {
        wdت_reset();
        //LED OFF
        PORTB &= ~(1 << PINB5); // Set PORTB5 Low
        _delay_ms(20);
        //LED ON
        PORTB |= (1 << PINB5); // Set PORTB5 On
        _delay_ms(80);
    }
}

```

## 10

### Tarea 10 - Prueba WDT

- Presione el botón en el tablero durante aproximadamente 1 segundo y luego suelte el botón. Se activarán retrasos prolongados en la rutina principal, lo que provocará un tiempo de espera de WDT.

Nota: El temporizador WDT está configurado en 2 segundos.

```

if((PINB&1<<PINB7)==0){
    PORTB |= (1 << PINB5);           // Set PORTB5 high
    _delay_ms(500);
    _delay_ms(500);
    _delay_ms(500);
    _delay_ms(500);
    _delay_ms(500);
    _delay_ms(500);
    _delay_ms(500);
    _delay_ms(500);
}

wdt_reset();
}

```

- Resultado: el LED naranja responderá con un parpadeo rápido (0,02 ms encendido, 0,08 ms apagado, cuatro veces) al principio cuando el WDT interrumpe el ISR:

```
//Watchdog timeout ISR
ISR(WDT_vect)
{
    //Burst of 0.1Hz pulses
    for (uint8_t i=0;i<4;i++)
    {
        //LED OFF
        PORTB &= ~(1 << PINB5);           // Set PORTB5 Low
        _delay_ms(80);
        //LED ON
        PORTB |= (1 << PINB5);          // Set PORTB5 On
        _delay_ms(20);
    }
}
```

- Resultado: el LED naranja volverá a responder con un parpadeo rápido (0,02 ms encendido, 0,08 ms apagado, cuatro veces) a medida que se detecta el restablecimiento del WDT:

```

else if(MCUSR & 8) { // WDT reset detected
    MCUSR=0;
    WDT_Init();
    for (i=0;i<8;i++)
    {
        wdت_reset(); //LED OFF
        PORTB &= ~(1 << PINB5); // Set PORTB5 Low
        _delay_ms(20);
        //LED ON
        PORTB |= (1 << PINB5); // Set PORTB5 On
        _delay_ms(80);
    }
    MCUSR=0;
}

//Initialize watchdog

```

## Análisis

El proyecto muestra tres formas en que puede ocurrir un reinicio: POR, BOR y WDT Timeout. Cada uno tiene una aplicación única y todos pueden ejecutarse en la misma aplicación. Los ejemplos de código son solo una referencia de cómo se pueden configurar e implementar estos tipos de restablecimientos.

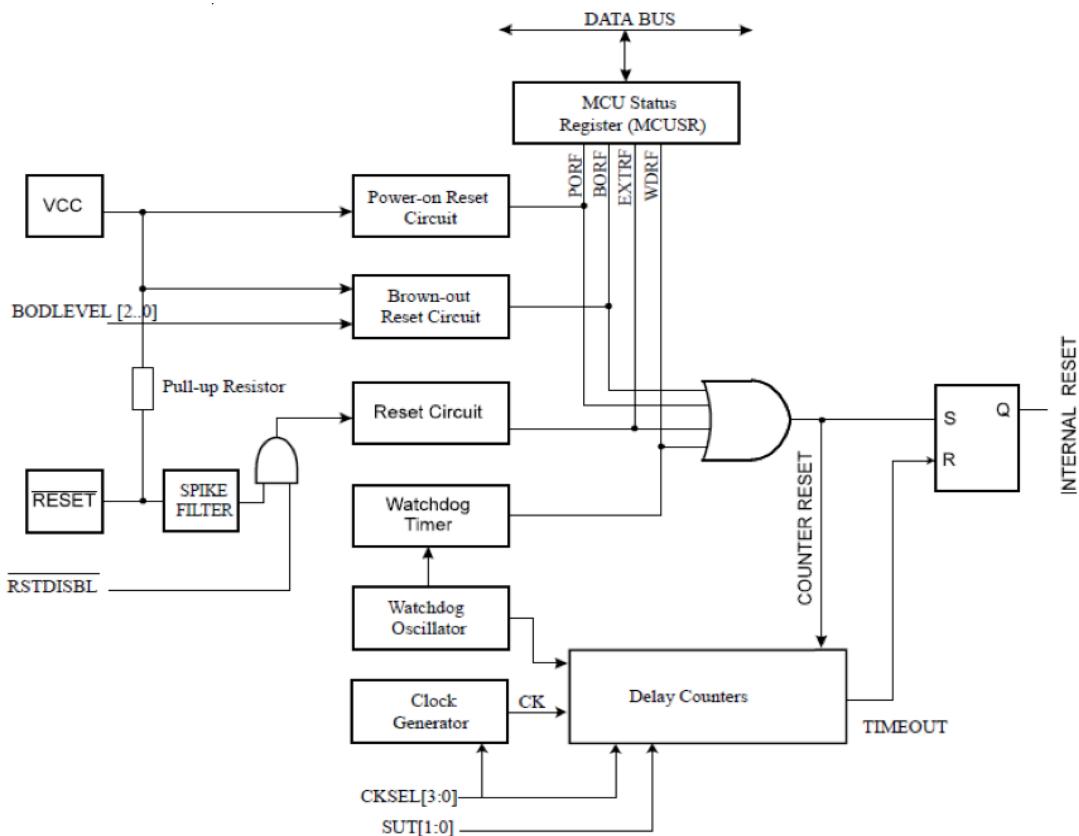
## Conclusiones

El proyecto ayuda a explicar cómo funciona el circuito de reinicio dentro del AVR y cómo implementarlo. La sección de código se puede reutilizar en aplicaciones futuras que pueden requerir una estructura de reinicio similar. De ninguna manera es esta la única forma de diseñar restablecimientos en el dispositivo AVR, este es solo un proyecto de muestra simple que ayuda a explicar la operación y le permite aplicar su conocimiento y comprensión de la estructura de restablecimiento a una aplicación específica.

# Fuentes de restablecimiento de AVR

El dispositivo AVR tiene cuatro fuentes de reinicio:

- **Power-on Reset** - The Microcontroller (MCU) is reset when the supply voltage is less than the Power-on Reset threshold (VPOT).
- **External Reset** - The MCU is reset when a low level is present on the RESET pin for longer than the minimum pulse length.
- **Watchdog System Reset** - The MCU is reset when the Watchdog Timer period expires and the Watchdog System Reset mode is enabled.
- **Brown-out Reset** - The MCU is reset when the supply voltage  $V_{CC}$  is less than the Brown-out Reset.



## MCU Status Register (MCUSR)

To make use of the reset flags to identify a reset condition, the user should read and then reset the MCUSR as early as possible in the program. If the register is cleared before another reset occurs, the source of the reset can be found by examining the reset flags.

**Name:** MCUSR

**Offset:** 0x54

**Reset:** 0x00

**Property:** When addressing as I/O Register: address offset is 0x34

Bit	7	6	5	4	3	2	1	0
					WDRF	BORF	EXTRF	PORF
Access					R/W	R/W	R/W	R/W

**Bit 3 – WDRF: Watchdog System Reset Flag**  
This bit is set if a Watchdog System Reset occurs. The bit is reset by a Power-on Reset, or by writing a '0' to it.

#### **Bit 2 – BORF: Brown-out Reset Flag**

This bit is set if a Brown-out Reset occurs. The bit is reset by a Power-on Reset, or by writing a '0' to it.

#### **Bit 1 – EXTRF: External Reset Flag**

This bit is set if an External Reset occurs. The bit is reset by a Power-on Reset, or by writing a '0' to it.

#### **Bit 0 – PORF: Power-on Reset Flag**

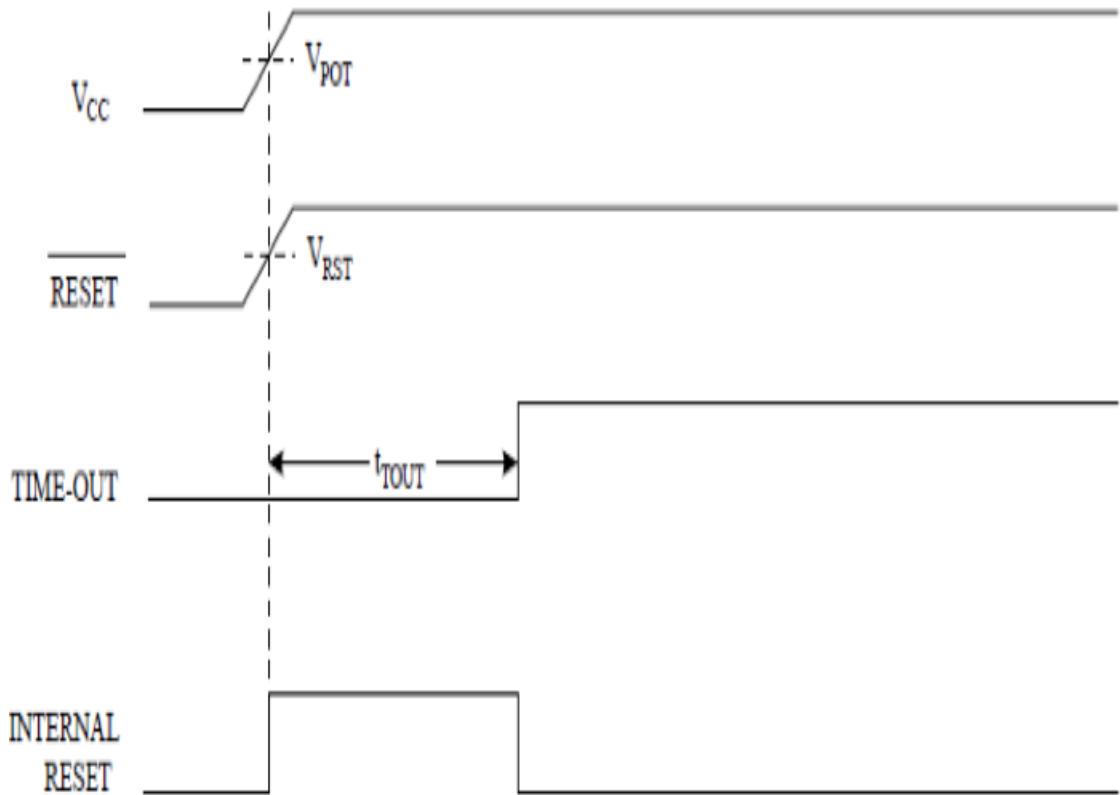
This bit is set if a Power-on Reset occurs. The bit is reset only by writing a '0' to it.

## **Power-on Reset (POR)**

A POR pulse is generated by an On-chip detection circuit. The POR is activated whenever  $V_{cc}$  is below the detection level. The POR circuit can be used to trigger the start-up Reset, as well as to detect a failure in supply voltage.

A POR circuit ensures that the device is reset from Power-on. Reaching the Power-on Reset threshold voltage invokes the delay counter, which determines how long the device is kept in Reset after  $V_{cc}$  rise.

The Reset signal is activated again, without any delay, when  $V_{cc}$  decreases below the detection level.

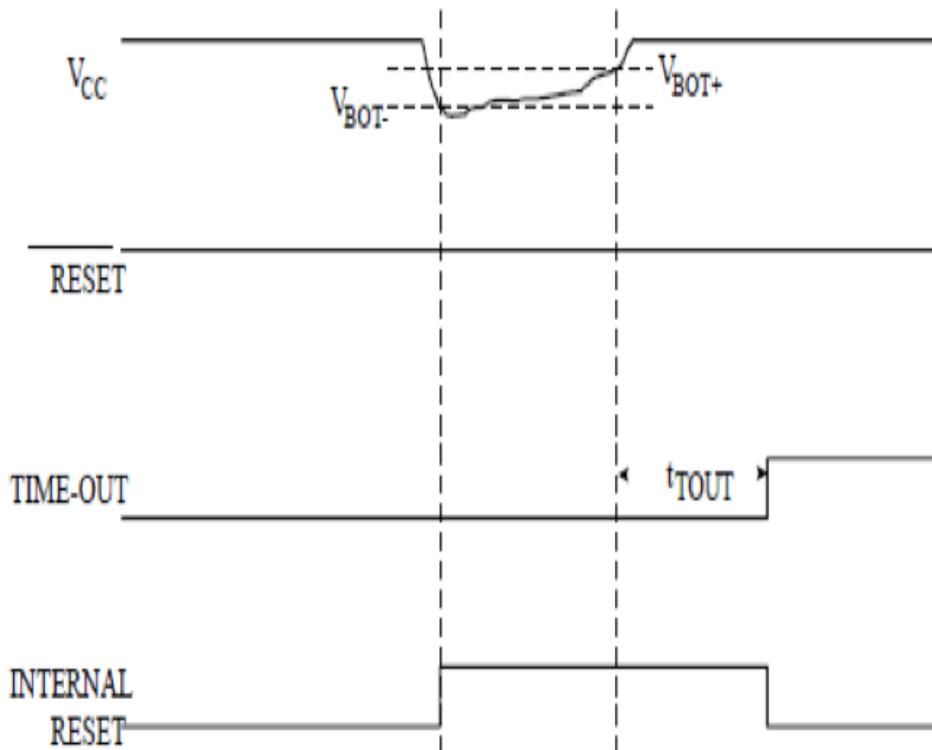


## Brown-out Detection (BOD) and Brown-out Reset (BOR)

The device has an On-chip BOD circuit for monitoring the  $V_{CC}$  level during operation by comparing it to a fixed trigger level. The trigger level for the BOD can be selected by the BODLEVEL Fuses.

The BOR circuitry has hysteresis on the detection level. The BOD circuit will only detect a drop in  $V_{CC}$  if the voltage stays below the trigger level ( $V_{BOD-}$ ) for longer than  $t_{BOD}$ . When that occurs, the BOR is immediately activated.

Cuando  $V_{CC}$  aumenta por encima del nivel de activación ( $V_{BOD+}$  en la siguiente figura), el contador de retardo inicia la MCU después de que haya expirado el período de tiempo de espera  $t_{TOUT}$ .



## Temporizador de vigilancia (WDT)

El WDT se ejecuta independientemente del resto del sistema, lo que hace que el sistema se reinicie cada vez que se agote el tiempo de espera. Sin embargo, el software de la aplicación debe garantizar que nunca se agote el tiempo de espera reiniciando el WDT periódicamente siempre que el software se encuentre en un estado saludable conocido. Si el sistema se bloquea o la ejecución del programa se corrompe, el WDT no recibirá su reinicio periódico y, finalmente, expirará y provocará un reinicio del sistema.

El WDT mejorado en algunos dispositivos AVR también tiene la capacidad de generar interrupciones en lugar de reiniciar el dispositivo. Dado que el WDT funciona con su propio reloj independiente, se puede utilizar para activar el AVR desde todos los modos de suspensión. Esto lo convierte en un temporizador de despertador ideal, que se combina fácilmente con el funcionamiento normal como fuente de reinicio del sistema. La interrupción también se puede utilizar para obtener una advertencia temprana de un próximo restablecimiento del sistema Watchdog para que los parámetros vitales se puedan respaldar en una memoria no volátil.

## Detección de fallas de reloj (CFD)

El CFD permite al usuario monitorear el oscilador de cristal de baja potencia o la señal del reloj externo (XOSC). El XOSC es monitoreado por el circuito CFD que opera con el oscilador interno de 128kHz. CFD supervisa el reloj XOSC y, si falla,

cambiará automáticamente a un reloj RC interno seguro. Cuando se produce un encendido o un restablecimiento externo, el dispositivo volverá al reloj XOSC y continuará monitoreando el reloj XOSC en busca de fallas.

El reloj seguro se deriva del reloj del sistema RC interno de 8 MHz. Esto permite configurar el reloj seguro para satisfacer las necesidades de seguridad de la aplicación.