

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_{cc} mínimo. El reloj de la fuente seleccionada se introduce en el generador de reloj AVR® y se enruta a los módulos apropiados.

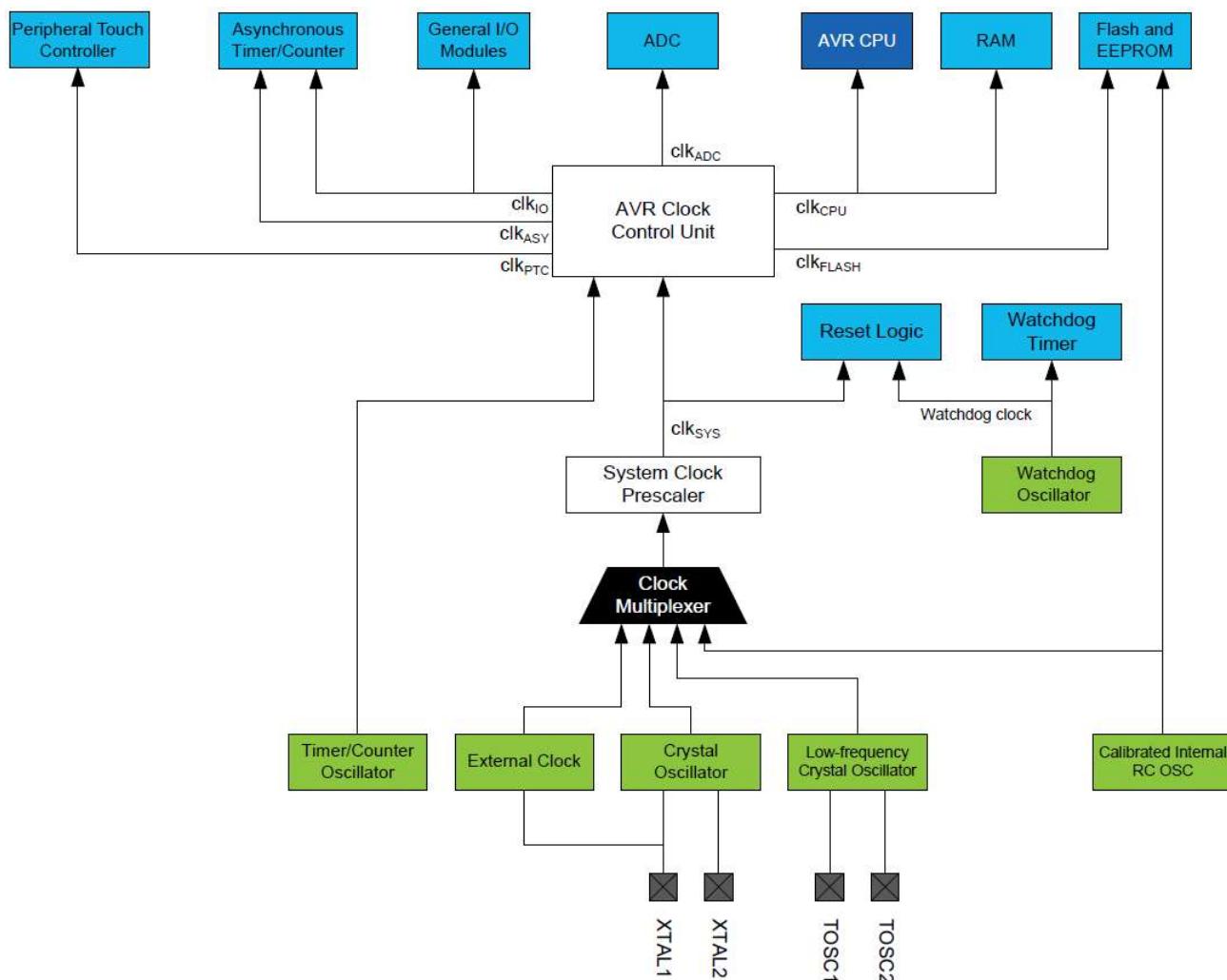


La **frecuencia máxima de funcionamiento de megaAVR® depende de V_{cc}**. El software de aplicación debe asegurarse de que la frecuencia de la fuente de reloj seleccionada se encuentra dentro del área de operación segura (ver sección 33.4

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

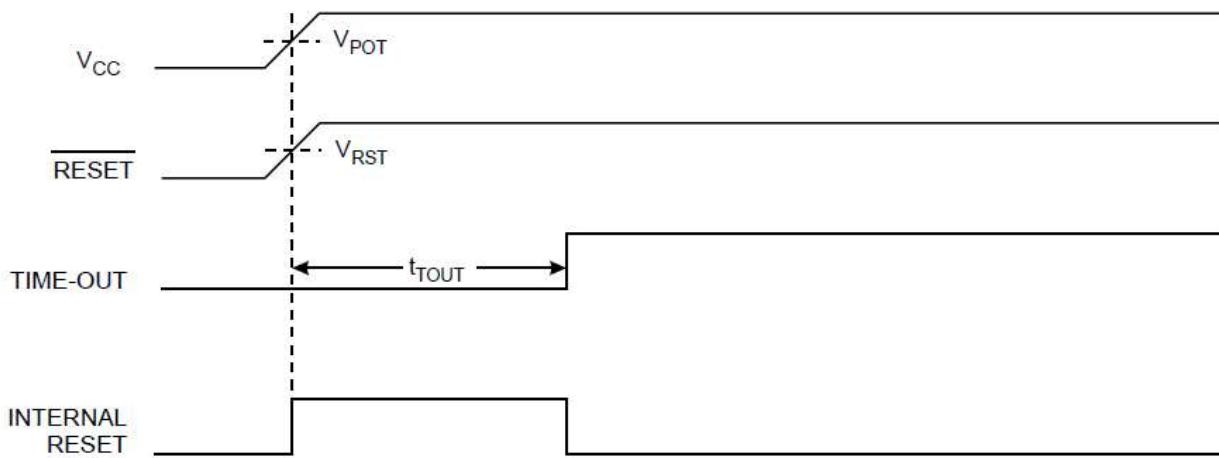
El dispositivo se envía con el oscilador RC interno seleccionado a 8.0 MHz y con el fusible CKDIV8 programado, lo que resulta en un reloj del sistema de 1.0 MHz. El tiempo de inicio se establece en máximo y el período de tiempo de espera está habilitado: CKSEL=0010, SUT=10, CKDIV8=0. Esta configuración 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_{Cc} 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_{Cc}, 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 ($V_{\text{CC}} = 5.0\text{V}$)	Typ. Time-out ($V_{\text{CC}} = 3.0\text{V}$)
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

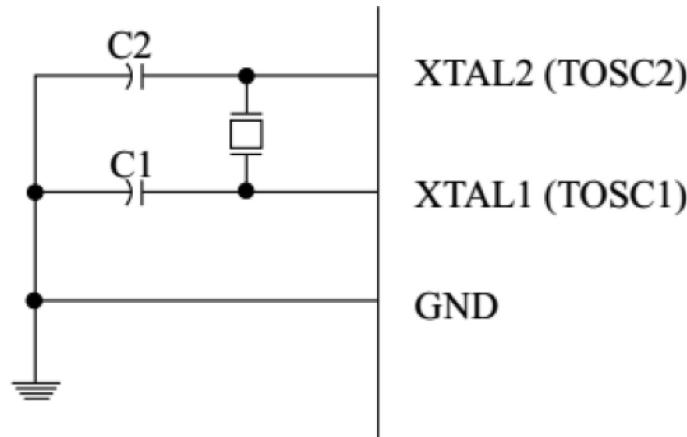
Se requiere que el oscilador oscile durante un número mínimo de ciclos antes de que el reloj se considere estable. Un contador de ondulación interno monitorea el reloj de salida del oscilador y mantiene activo el restablecimiento interno durante un número determinado de ciclos de reloj. El restablecimiento 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] ⁽²⁾	Range for total capacitance of C1 and C2 [pF] ⁽⁴⁾
0.4 - 0.9	100 ⁽³⁾	–
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), que describe el procedimiento para volver a calibrar el oscilador RC interno.

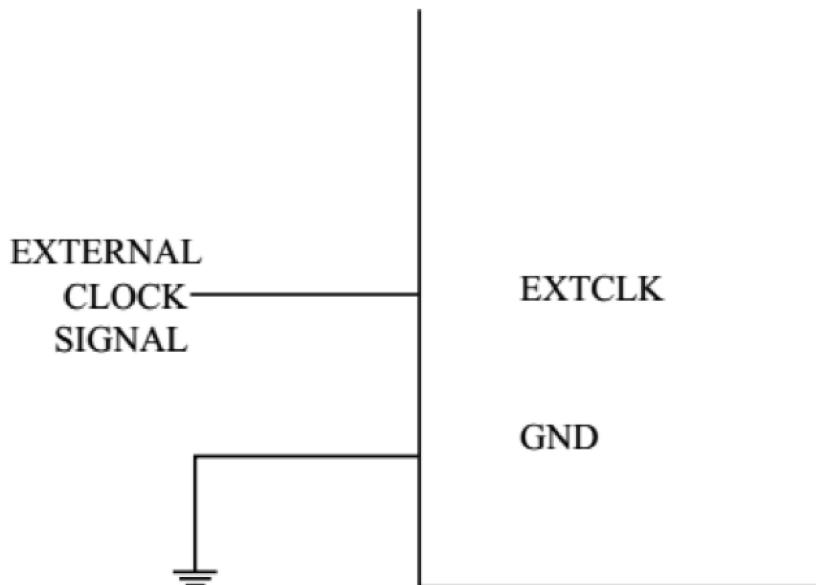
Oscilador interno de 128 kHz

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. $\text{Clk}_{\text{E/S}}$, Clk_{Adc} , Clk_{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)

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

(/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
ATmega328PB Xplained Mini Kit de evaluación	(http://www.atmel.com/tools/MEGA328PB-XMINI.aspx) https://www.microchipdirect.com/prc

Herramientas de software

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

Archivos de ejercicio

Archivo	Windows	Linux
https://microchipdeveloper.com/8avr:osc-mega-example		

Proyecto
de
ejemplo

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

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

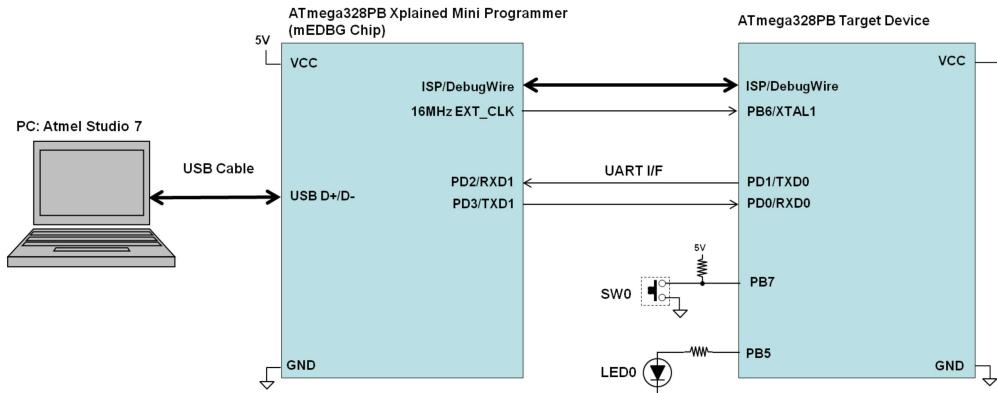


Recomendamos extraer el archivo .zip a su C:\ carpeta.

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)

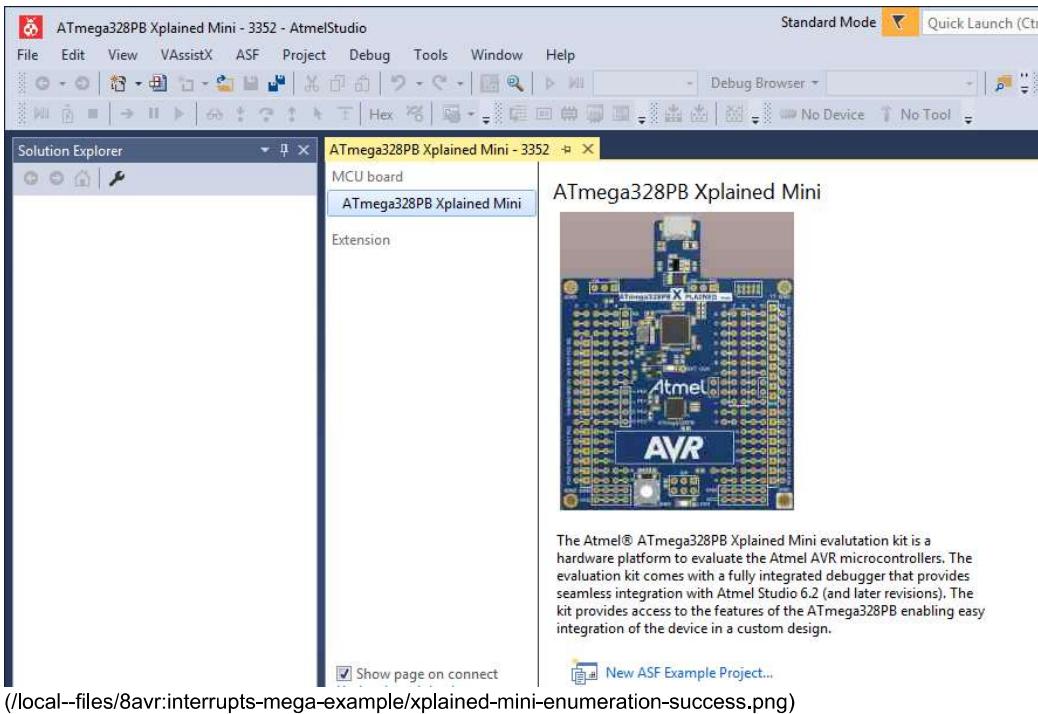


The target ATmega328PB CKSEL fuse bits are initially unchangeable on the Xplained Mini board from within Atmel Studio. "Verification" errors will be displayed if programming CLSEL bits with any other setting other than "external clock".

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).

❗ Procedure

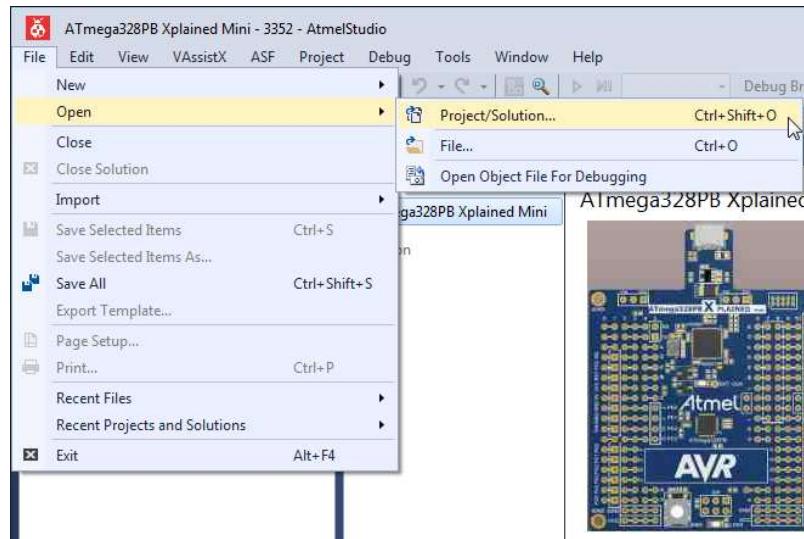
Attach the **ATmega328PB Xplained Mini board** to your computer using a USB-A-male-to-Micro-B-male cable. Start Atmel Studio 7. If the board has been successfully enumerated, you should see the board image come up in Studio as shown:



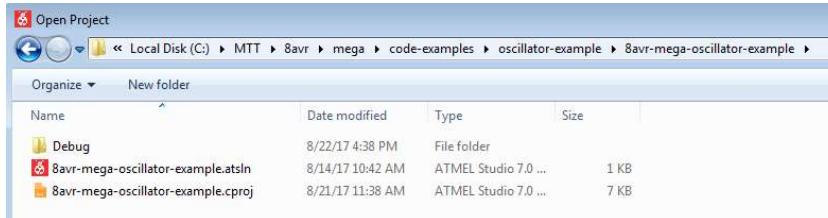
i The Xplained mini board is identified by the last four digits in its serial number (see sticker on bottom of board). In the above example, the last four digits are "3352"

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`:

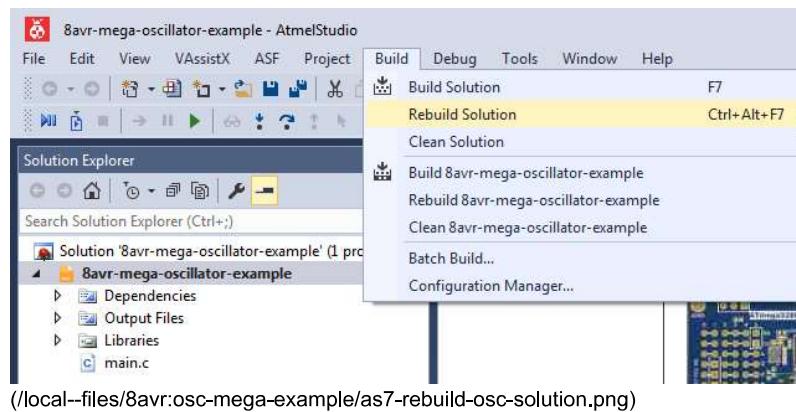


(/local--files/8avr:interrupts-mega-example/as7-open-solution.png)



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

2 Rebuild the Solution

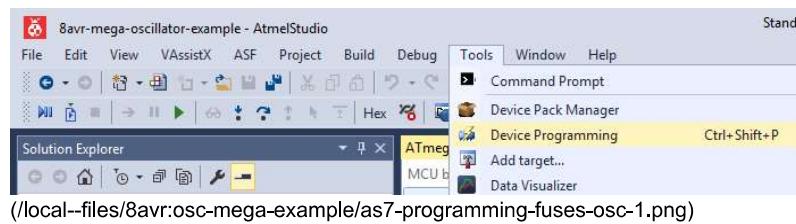


3 Program the Fuses

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

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

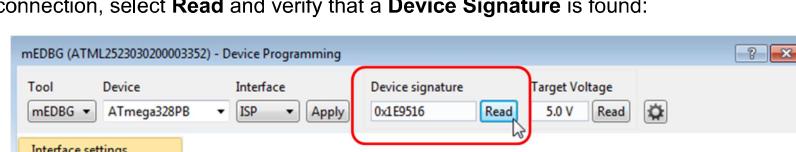
Enter the Device Programming dialog as shown:



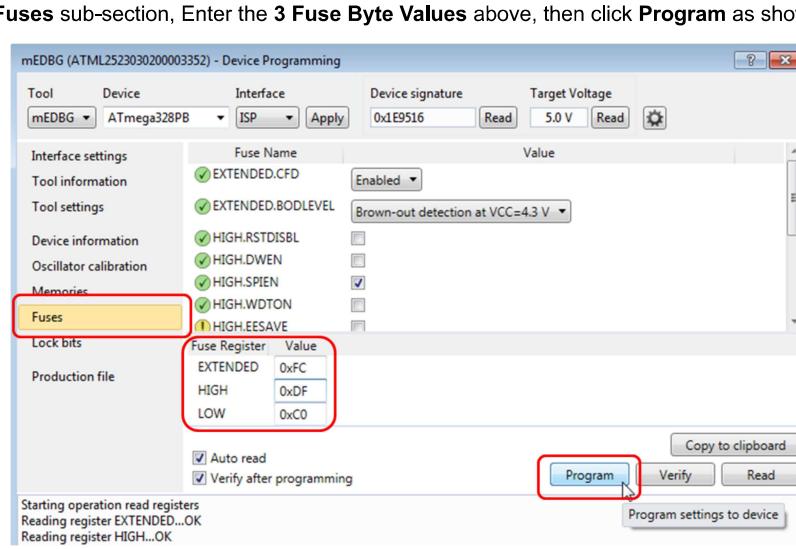
In the **Device Programming** dialog box, select **Tool**, **Device** and **Interface** as shown, then click **Apply**:



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

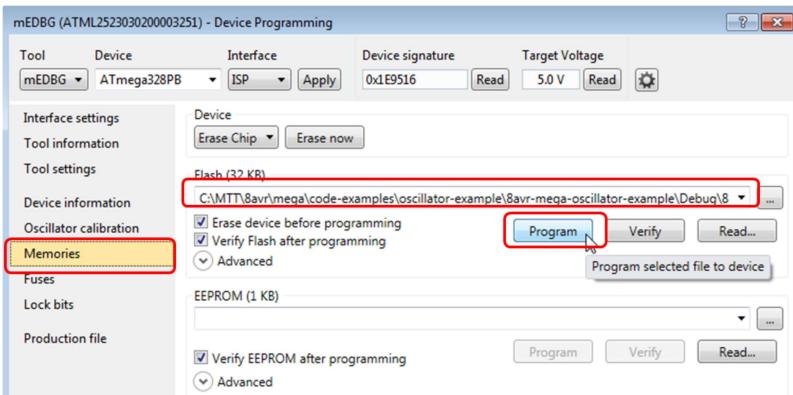


Select the **Fuses** sub-section, Enter the 3 **Fuse Byte Values** above, then click **Program** as shown:



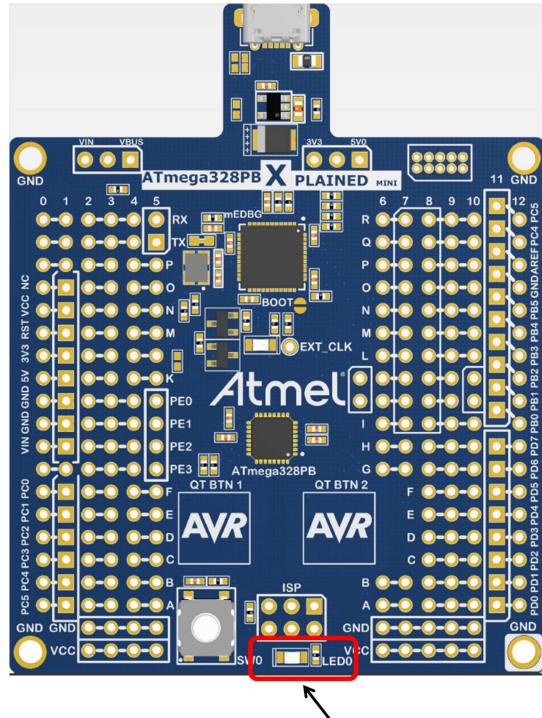
4 Program the Hex File

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 > (/avr:osc-mega-overview)

**megaAVR® Timer Overview**

Learn more > (/avr:avrtimerover)

**megaAVR® Interrupts Overview**

Learn more > (/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).

Aprende más



Configuración

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



megaAVR® USART Ejemplo (sondeado)

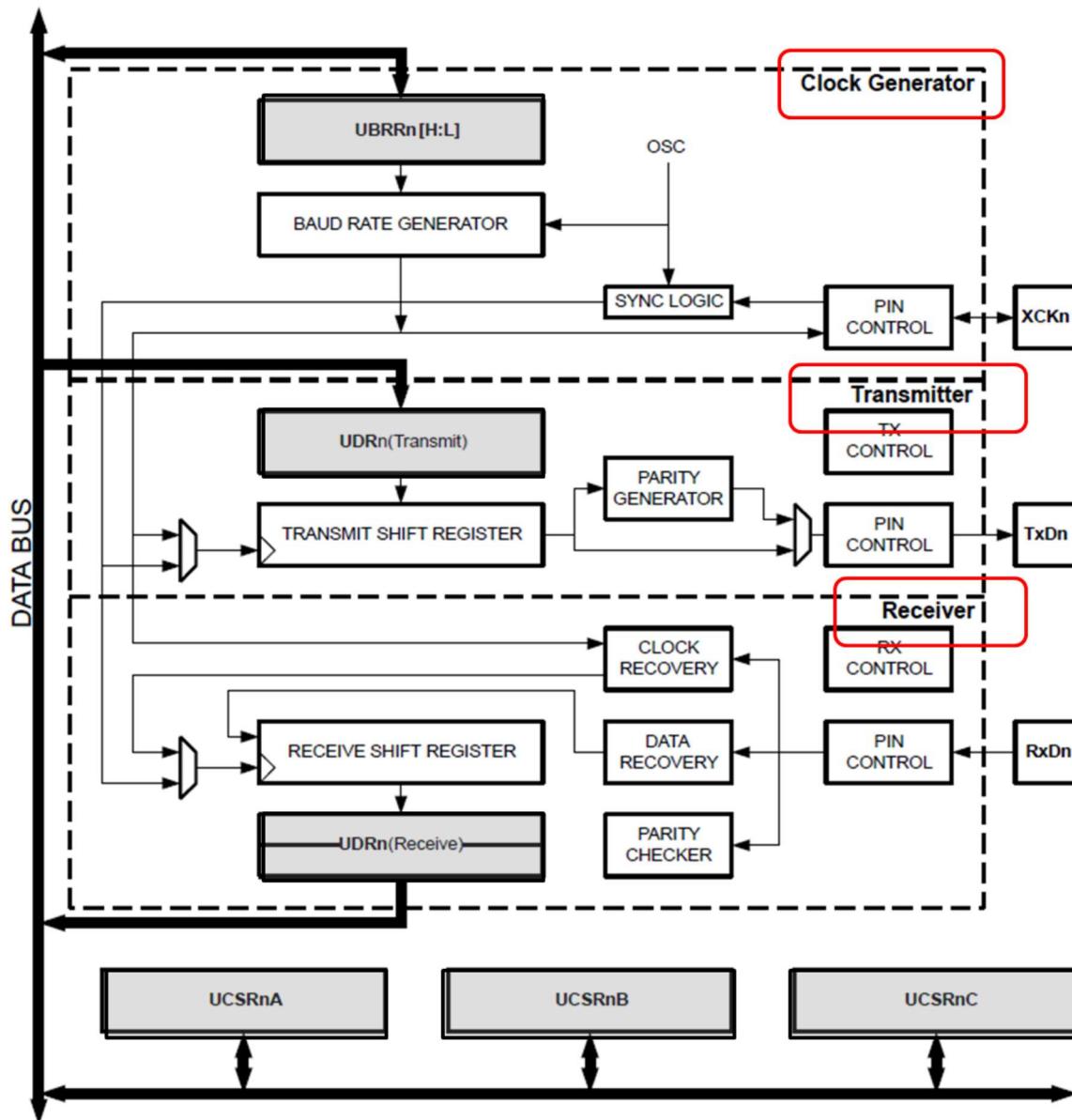
Más información > (/8avr:uart-mega-example-polled)

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.

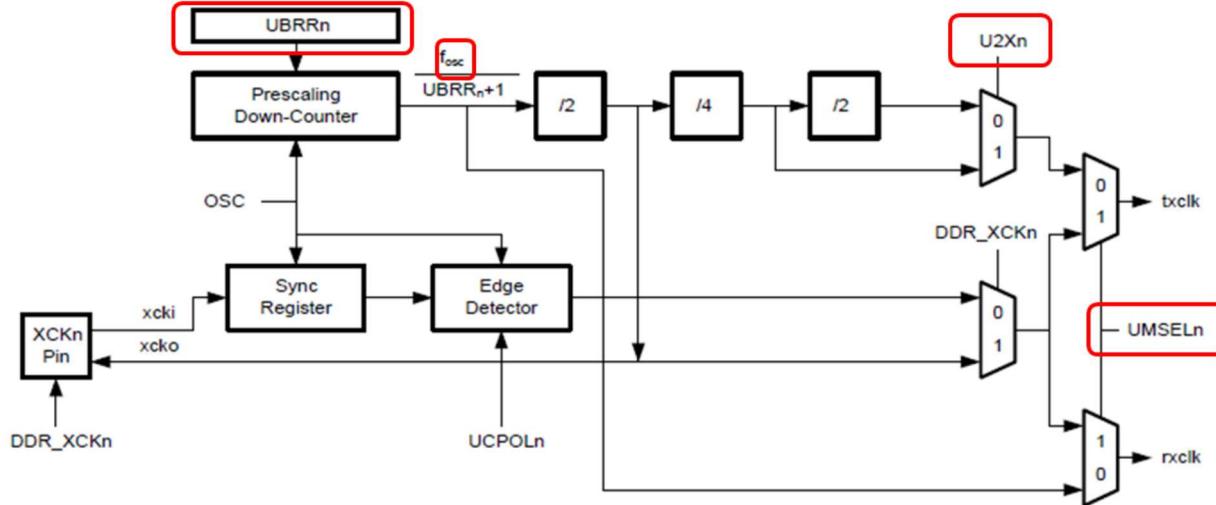
Inicialización

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_{osc}:** 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) 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

Reset values: 0, 0, 0, 0, 0, 1, 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) 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(void)
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
 (http://www.atmel.com/tools/MEGA328PB-XMINI.aspx)	(http://www.atmel.com/tools/MEGA328PB-XMINI.aspx) (https://www.microchipdirect.com/prc/XMINI.aspx)

Herramientas de software

Herramienta	ⓘ Acerca de	Instaladores		
		Windows	Linux	Mac OSX
Estudio Atmel®	 Entorno de desarrollo integrado	(/atstudio:start) Download (http://studio.download.atmel.com/7.0.1931/as-installer-7.0.1931-full.exe)	Download	Download
Emulador de terminal de Término Tera	 (/swtools:terminal-emulator)	Download (https://en.osdn.jp/projects/ttssh2/releases/)	Download	Download

Archivos de ejercicio

Archivo	Windows	Linux	ⓘ Descargar

Proyecto de ejemplo

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

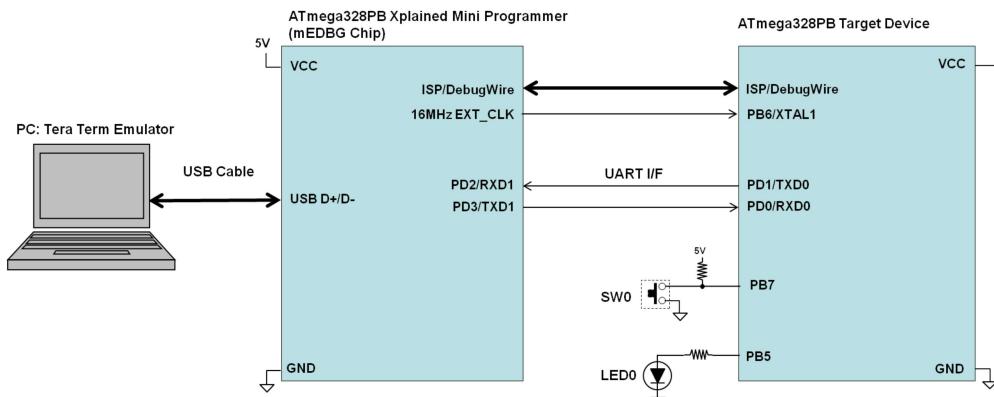
[\(local--files/8avr:uart-mega-example-polled/8avr-mega-uart-example-polled.atsln\)](#)

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

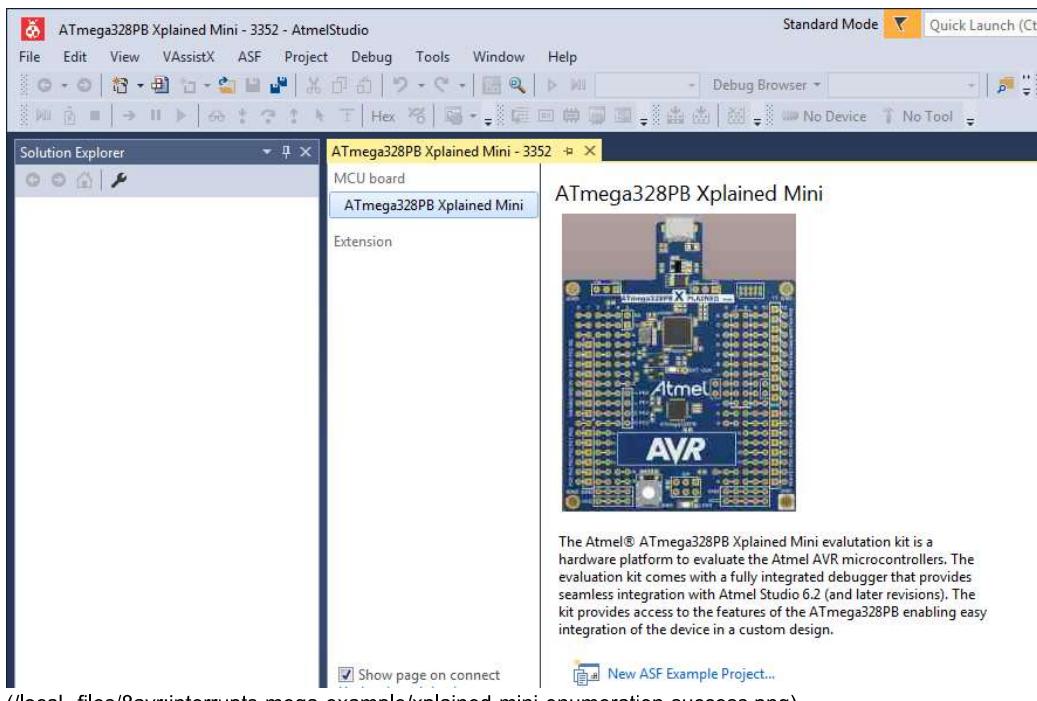
El módulo USART0 del dispositivo ATmega328PB de destino está conectado a la interfaz USART del chip mEDBG. El chip mEDBG realiza la conversión serie USB enumerando como un puerto COM virtual de clase CDC en la PC y presentando los datos USART de destino en esta interfaz. Tenga en cuenta que el mEDBG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



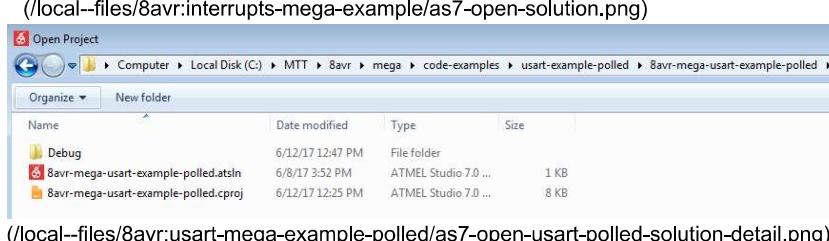
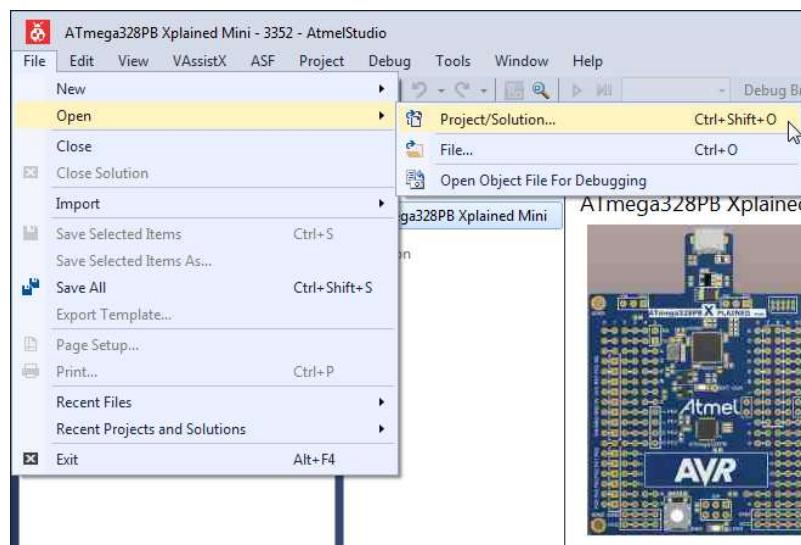
The Xplained mini board is identified by the last four digits in its serial number (see sticker on bottom of board). In the above example, the last four digits are "3352".

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:



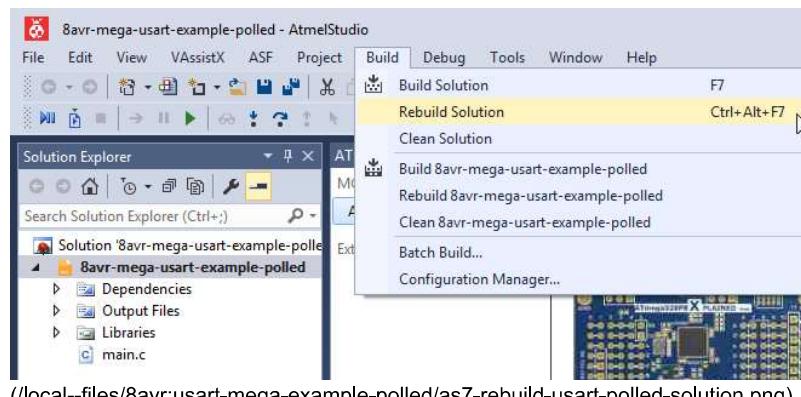
1 Open the Solution

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

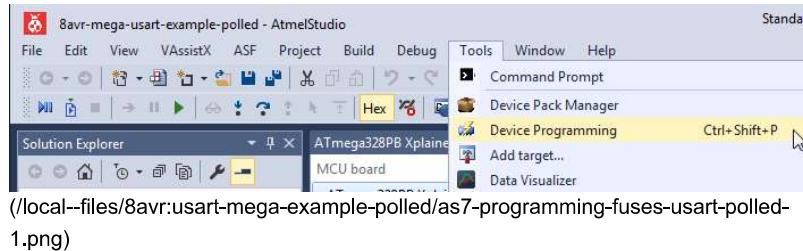


3 Program the Fuses

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:



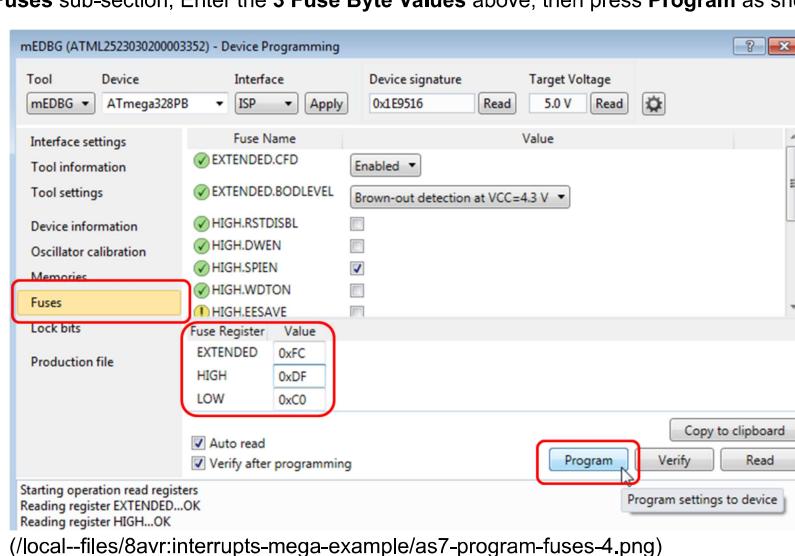
In the Device Programming dialog box, select the **Tool**, **Device** and **Interface** as shown, then press **Apply**:



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

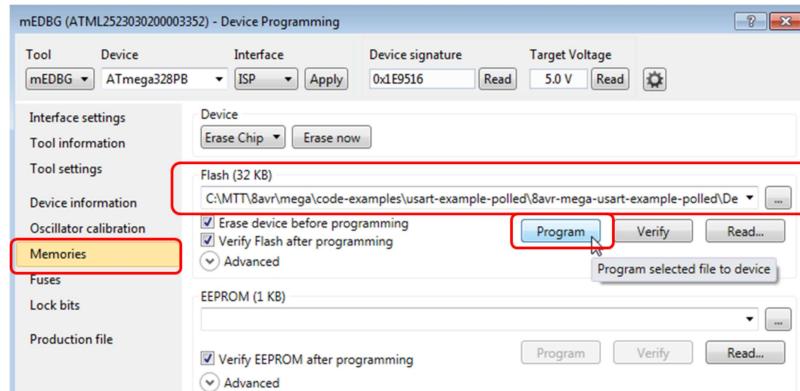


Select the **Fuses** sub-section, Enter the **3 Fuse Byte Values** above, then press **Program** as shown:



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:



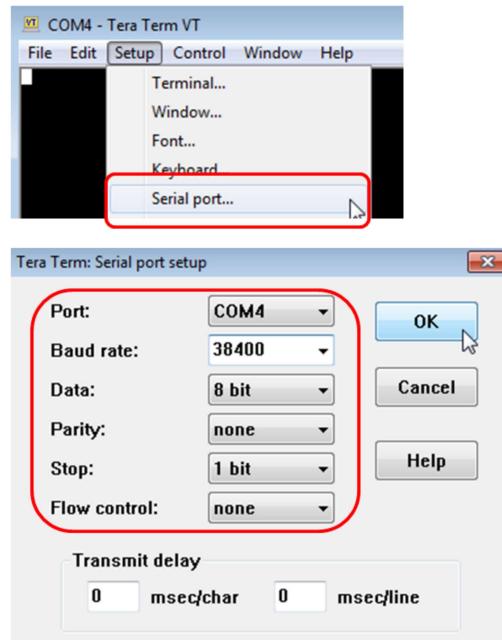
(/local--files/8avr:uart-mega-example-polled/as7-program-hex-usart-polled-1.png)

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



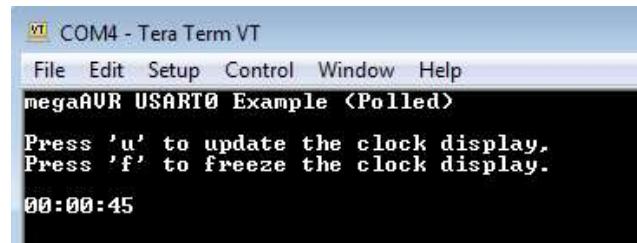
(/local--files/8avr:uart-mega-example-polled/teraterm-select-com-port.png)

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)