GUÍA ATMEL STUDIO 7

Desarrollado por Raúl Alzate & David Sterling en agosto - 2018 para el curso de B-Learning de señales y sistemas.

# Introducción a Atmel Studio:

Atmel Studio es un **IDE** desarrollado por la compañía Atmel Corporation. Este software fue creado para la programación de todos los productos de microcontroladores de esta empresa. Hasta la fecha la versión 7 es la última desarrollada antes de que Atmel fuera comprada por Microchip, también fabricante de microcontroladores.

Al igual que los demás IDE’s, Atmel Studio 7 facilita el desarrollo de software para microcontroladores ya que realiza **highlight** de las palabras claves, realiza funciones de **autocompletar** y además **indica errores de sintaxis** antes de realizar la compilación.

Sin embargo, ofrece otras características adicionales muy importantes para el desarrollo de microcontroladores como son las funciones de **depuración** (**debug on chip**), **simulación**, entre otros.Estos últimos temas se explorarán mas adelante en esta mima guía.

## Instalación de Atmel Studio 7

El desarrollo para microcontroladores involucra en primer lugar un compilador apto para la arquitectura de microcontroladores que se desea trabajar. En segundo lugar, se debe disponer de una herramienta hardware que permite la programación de estos dispositivos. En el caso de Atmel los protocolos más populares para esta labor son JTAG y SPI. Es muy común en las tarjetas de desarrollo que el hardware de programación este junto con el microcontrolador en la misma board. De esta manera se simplifica el trabajo del desarrollador ya que basta con hacer una conexión mediante USB para poder programarlo. Sin embargo, no siempre es el caso y por eso es muy importante tener presente que siempre se necesita disponer de un hardware para llevar a cabo la programación de los microcontroladores.

Este hardware de programación implica a su vez la necesidad de unos drivers, los cuales también son obligatorios en el proceso de instalación. Finalmente, el IDE propiamente hablando (Atmel Studio 7), el cual facilitara la integración de los elementos anteriormente citados.

Cuando se desea desplegar una aplicación para microcontroladores, el IDE primero debe invocar al compilador el cual convertirá el código fuente en lenguaje de máquina.

El Instalador se puede descargar en la pagina oficial de microchip: <http://www.microchip.com/mplab/avr-support/atmel-studio-7>. Una vez se ejecuta se mostrará la ventana que se muestran en la Ilustración 1. En esta ventana se puede configurar la ruta de instalación.



Ilustración 1. Inicio del instalador de Atmel Studio 7.

La siguiente ventana (Ilustración 2) permite seleccionar el tipo de arquitecturas con las que se desea trabajar. Esto esta relacionado con el compilador que usara el IDE ya que la obtención del código de maquina a partir de un programa escrito en C, depende de la arquitectura para la cual se esta compilando. Se recomienda que se seleccione soporte para las 3 arquitecturas de Atmel.

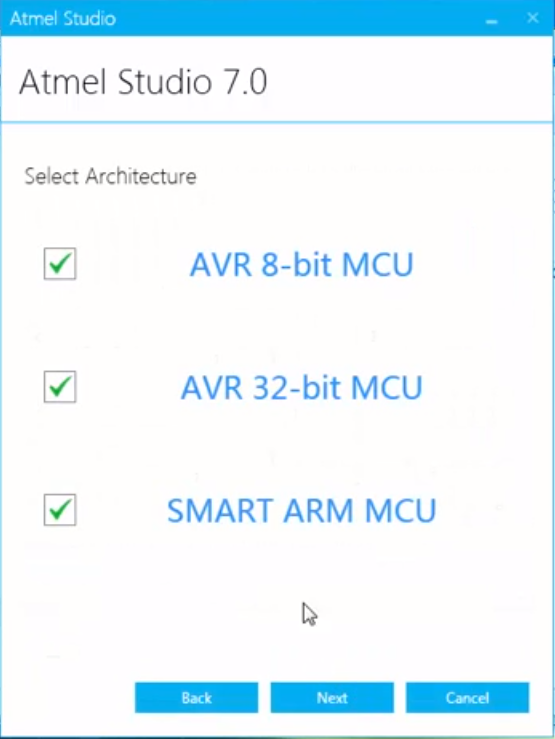


Ilustración 2. Selección de soporte para arquitecturas de Atmel.

En la tercera ventana, que se muestra en la Ilustración 3, se puede seleccionar si se instaran los proyectos de ejemplo y frameworks para el IDE.

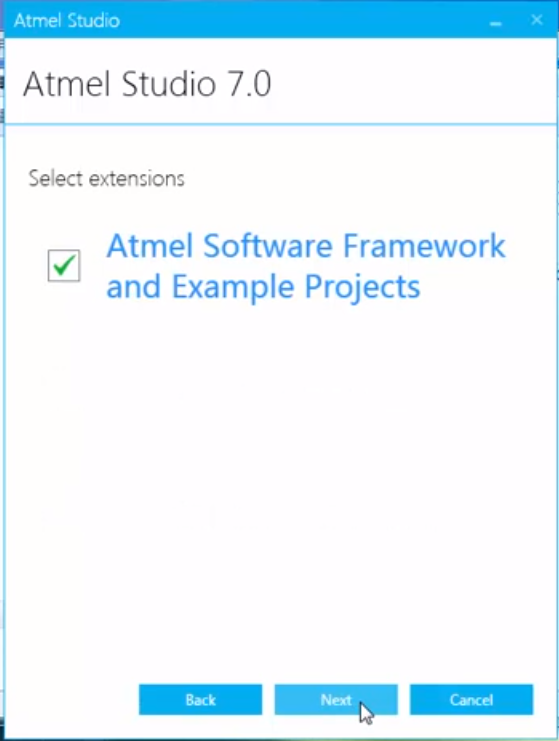


Ilustración 3. Adición de los proyectos de ejemplo y FrameWorks.

En la siguiente ventana se hace una validacion de los recursos del equipo para ver si soporta Atmel Studio 7.

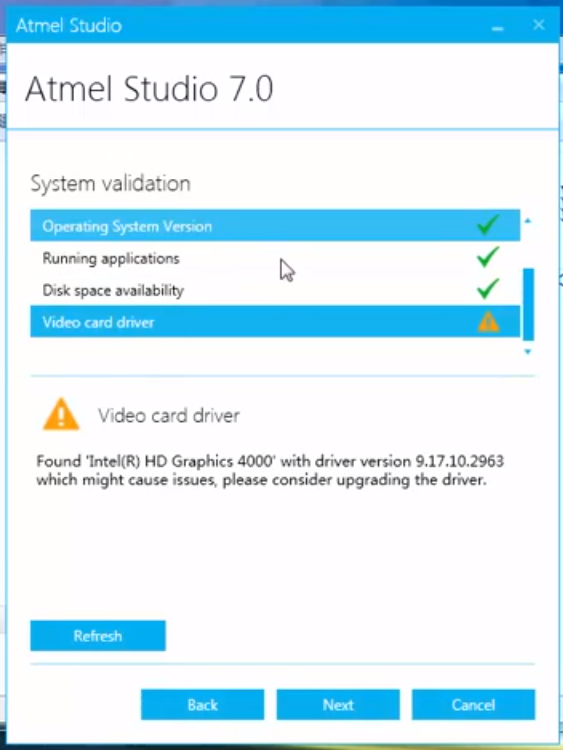


Ilustración 4. Validación de los requerimientos del sistema.

Se puede ver que hay una advertencia relacionada con la tarjeta gráfica del equipo, sin embargo, se prosigue con la instalación ya que no es impedimento. Luego se mostrará la ventana que se muestra en la Ilustración 5, indicando información sobre la instalación de usb drivers.

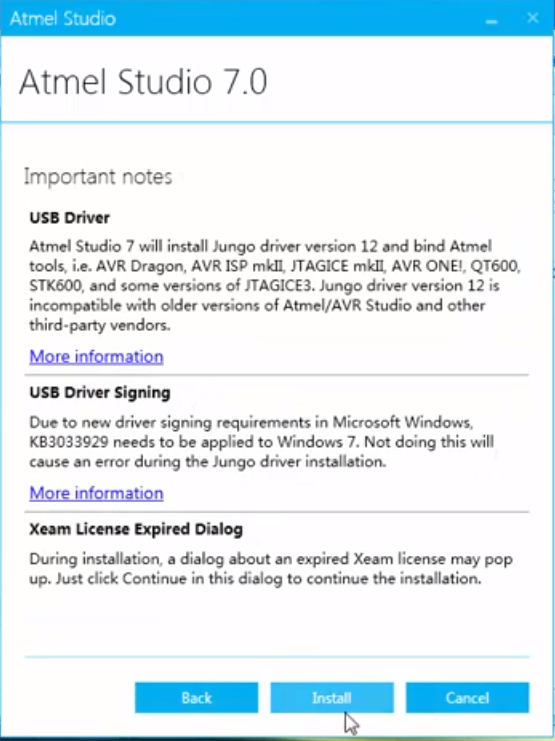


Ilustración 5. Notas sobre la instalación de Atmel Studio 7.

Cuando se procede a realizar la instalación se puede observar una ventana como la que se muestra en la Ilustración 6.

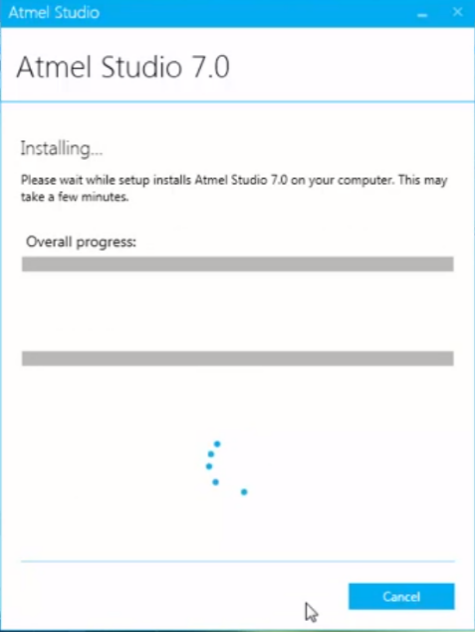


Ilustración 6. Proceso de instalación de Atmel Studio 7.

Una vez se termina la instalacion, se requerira que se haga un reinicio de la maquina tal como se muestra en la Ilustración 7.

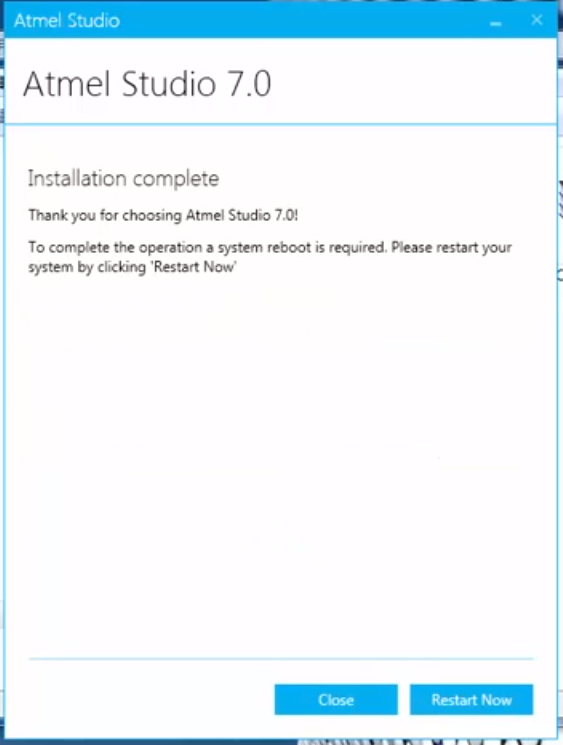


Ilustración 7. Finalización de la instalación de Atmel Studio 7.

# Creación de Proyectos en Atmel Studio 7

Se pueden varios tipos de proyectos diferentes. Estas opciones incluyen la creación de un proyecto usando lenguaje ensamblador, proyecto de C/C++ para librería o para aplicación y finalmente proyecto con una tarjeta de desarrollo.

## Proyectos En Ensamblador

En la Ilustración 8 se muestra la creación de un proyecto para usar ensamblador.

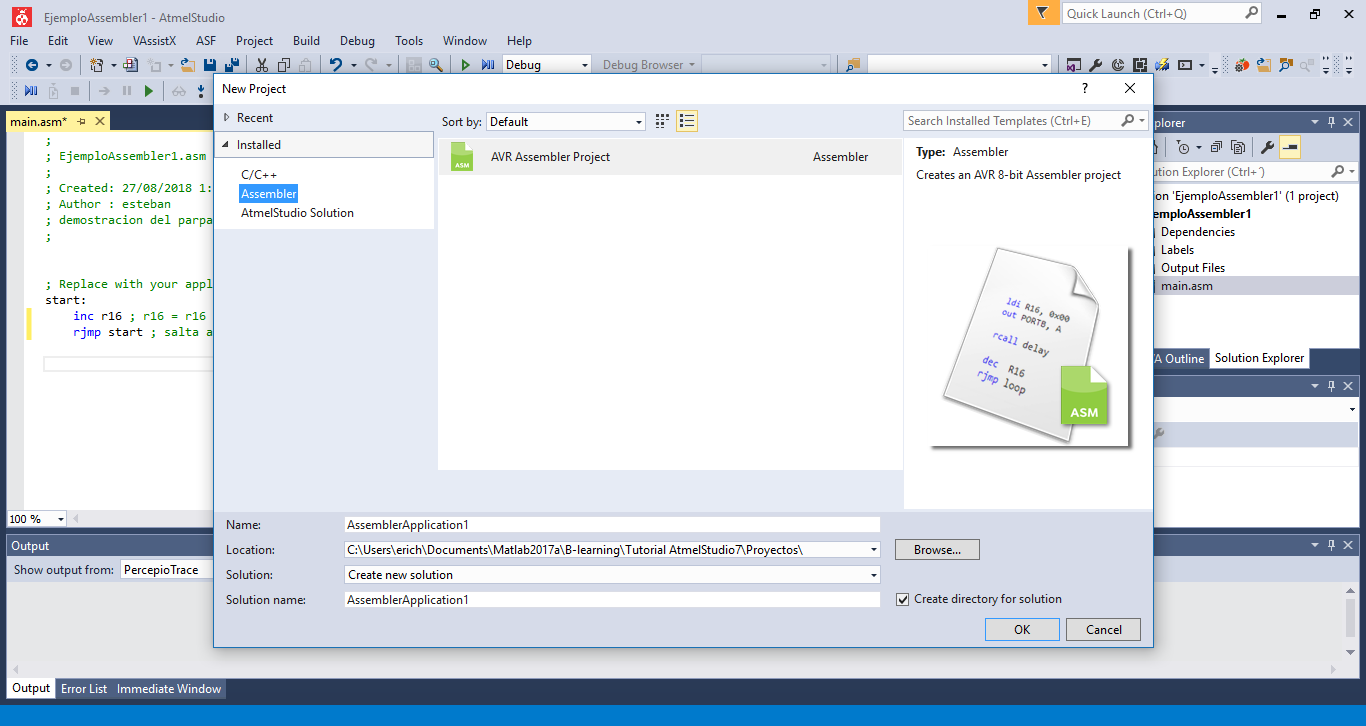


Ilustración 8. Creación de un proyecto para ensamblador usando atmel studio 7.

Basta con establecer el nombre y la ruta de localización del mismo. La opción de solución es una opción que se puede usar para agrupar varios proyectos que hacen parte de una misma solución. En la Ilustración 9 se puede el template que genera por defecto el IDE. Lo primero que se puede notar es que el archivo main es de extensión .asm, indicando que se trata de un source code para assembler. También se puede notar que los comentarios se escriben con “ ; ” a diferencia de C donde los comentarios se agregan con “ // “ o “ /\* \*/ “.

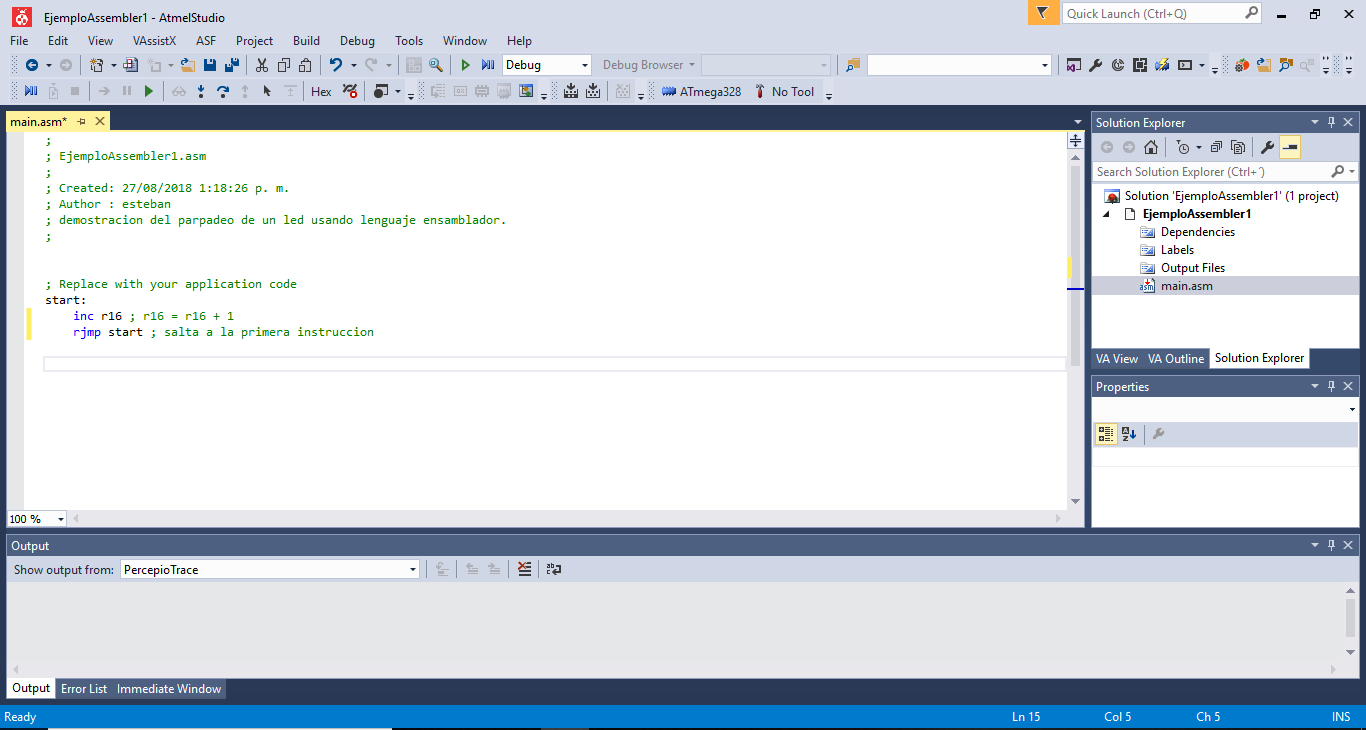


Ilustración 9. Plantilla de inicio cuando se crea un proyecto para ensamblador.

Los programas desarrollados en ensamblador son específicos del hardware para el cual se desarrollan debido a que el set de instrucciones varia de fabricante a fabricante. En el caso de AVR el set de instrucciones se puede encontrar en el siguiente enlace:

<http://www.avr-tutorials.com/sites/default/files/Instruction%20Set%20Summary.pdf>

más adelante en esta guía se muestra un ejemplo de un programa desarrollado en ensamblador.

## Proyecto Para Creación De Librería C/C++

Los proyectos que son tipo librería son muy similares los proyectos para aplicación con la diferencia que no se genera un ejecutable. También es muy importante tener en cuenta que si la librería se involucra directamente con los periféricos del microcontrolador es muy posible que la librería pierda generalidad. Lo anterior se debe a que los periféricos disponibles pueden variar de acuerdo a la referencia de los microcontroladores e incluso los nombres de los registros pueden variar. Esto se vera mas adelante en los proyectos de ejemplo.

## Proyecto Para Creación De Aplicación C/C++

En la Ilustración 10 se puede ver la creación de un proyecto tipo c/c++. Es similar a la creación de todos los proyectos, se debe seleccionar un nombre y una ubicación. En este tipo proyectos el objetivo final es generar un ejecutable para desplegar en el microcontrolador de destino.

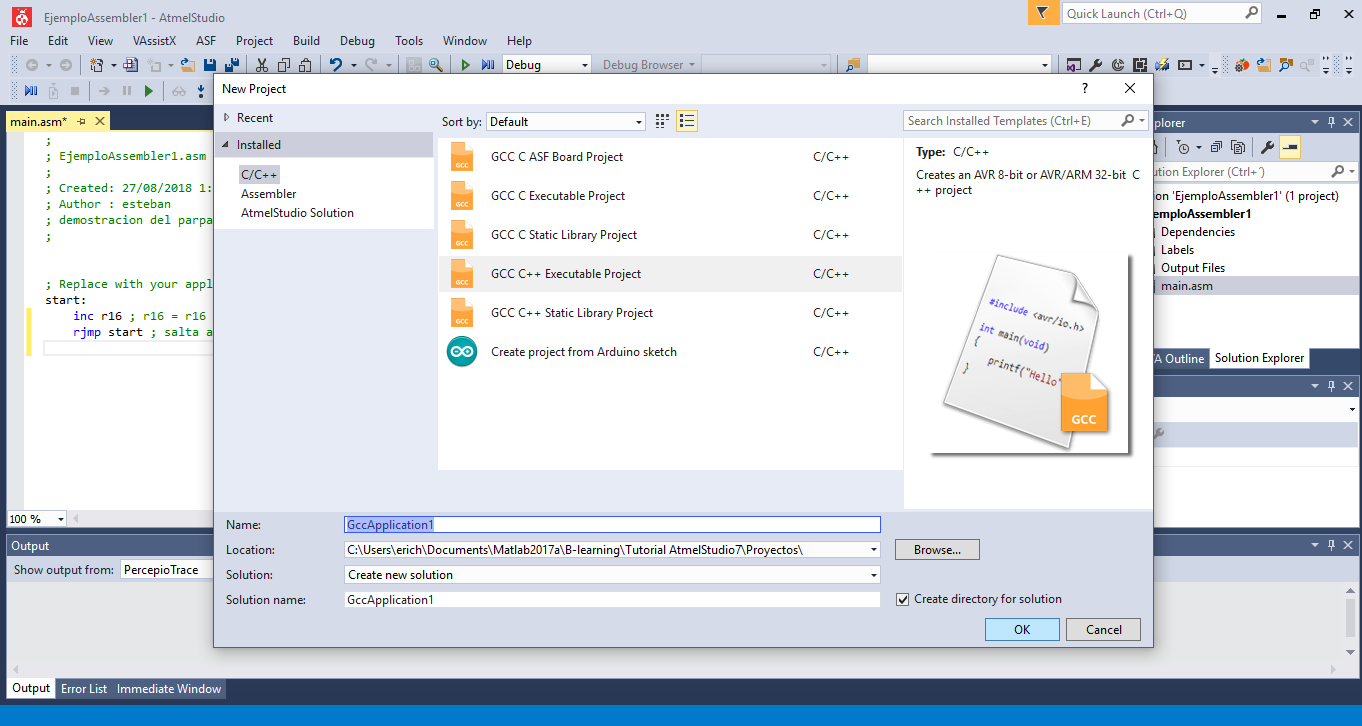


Ilustración 10. Creación de un proyecto con c/c++.

## Proyecto Para Trabajar Con Boards Basadas En Microcontroladores De Atmel

Cuando se trabaja directamente con microcontroladores de Atmel, es muy común el uso de programadores. Sin embargo, en el caso de tarjetas como Arduino, que se basan en microcontroladores de Atmel, se suele hacer uso de chips adicionales en la misma tarjeta para realizar comunicación serial entre el microcontrolador y la maquina de desarrollo y también para realizar la programación del microcontrolador. Lo anterior tiene la ventaja de permitir un manejo muy compacto del hardware ya que basta con conectarlo por usb a la máquina de desarrollo. Para esta clase de hardware se debe seleccionar la opción ASF BoardProject o Project From Arduino Sketch. En la Ilustración 11 se muestra una lista con las tarjetas de desarrollo disponibles para trabajar con Atmel studio.

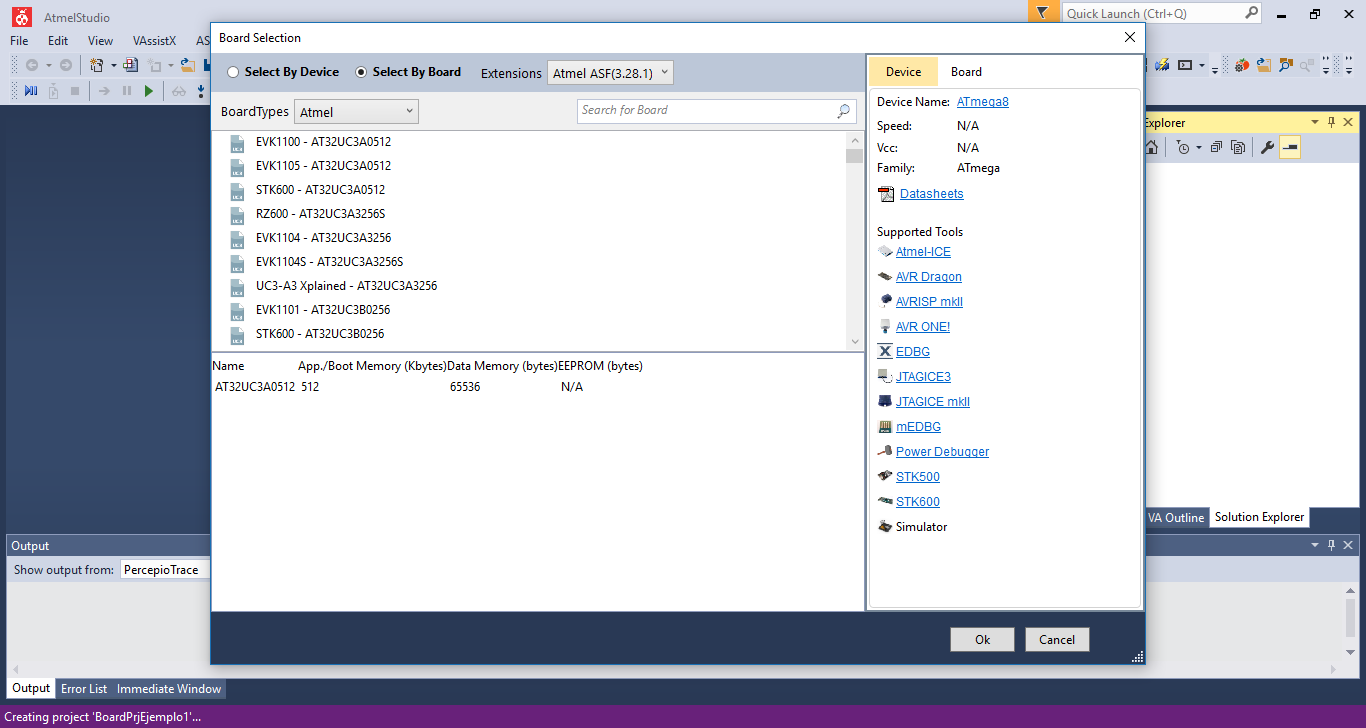


Ilustración 11. Creación de un proyecto usando una tarjeta de desarrollo.

# Programación De Los Microcontroladores

Para llevar a cabo la programación de estos dispositivos normalmente se requiere de un protocolo de comunicaciones entre el programador y el microcontrolador. El protocolo más común es el JTAG sin embargo también se dispone del protocolo SPI. El SPI tiene la ventaja de que requiere menos pines del microcontrolador, pero a su vez no es apto para realizar depuración, lo cual se discutirá más adelante en esta guía. Por el contrario, el protocolo JTAG permite hacer programación y depuración.

En cualquier caso, cuando se usa Atmel Studio, lo primero es conectar los pines del microcontrolador al programador. Posteriormente se accede a la pestaña de herramientas (tools) y se selecciona la opción Device Programming como se muestra en la Ilustración 12.

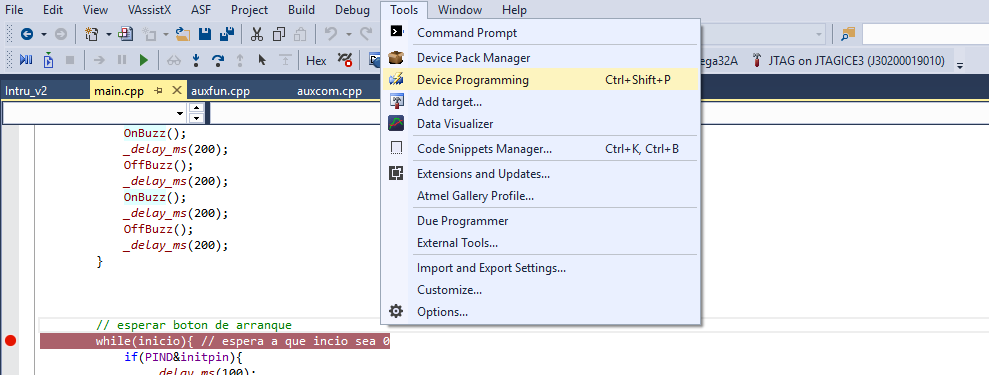


Ilustración 12. Apertura de la ventana de programación de microcontroladores.

## Conexión Hardware Para La Programación

La forma de conexión depende del protocolo que se va a emplear para hacer la programación de los microcontroladores. En este guía vamos a considerar los protocolos principales: JTAG y SPI pero se debe tener en cuenta que hay muchos más. El protocolo empleado también trae limitaciones a la velocidad máxima posible de programación, lo cual seria un factor importante cuando se desea producir boards en serie.

JTAG:

En la Ilustración 12 se muestran los pines que se usan para hacer la programación con JTAG de los microcontroladores.

Si se compara con la Ilustración 15, que corresponde al caso con programación SPI, se puede apreciar que el JTAG requiere mayor cantidad de pines para hacer la programación. Sin embargo, la gran ventaja de este protocolo es que también se puede usar para depuración.

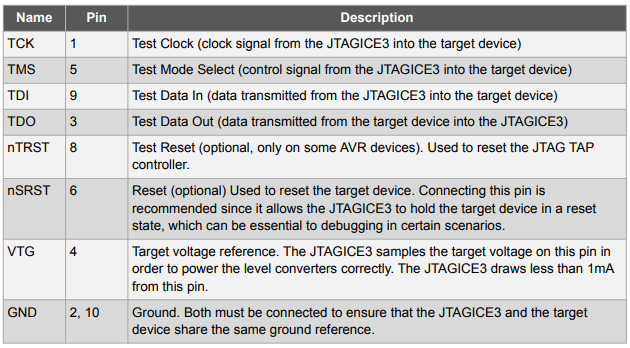


Ilustración 1. Tabla de pines para conexión con JTAG usando el JTAGICE3

Normalmente los microcontroladores de la familia tiny, que poseen pocos pines, no implementan interfaz por JTAG. En su lugar implementan la interfaz SPI, la cual requiere menos pines, sin embargo SPI no soporta debugging.

Para hacer la demostracion se empleo el ATmega162, el cual soporta ambas interfacez, JTAG y SPI. En la Ilustración 13 se muestra el PinOut del ATmega162, adema se resaltan los pines relacionados con la programacion por JTAG.

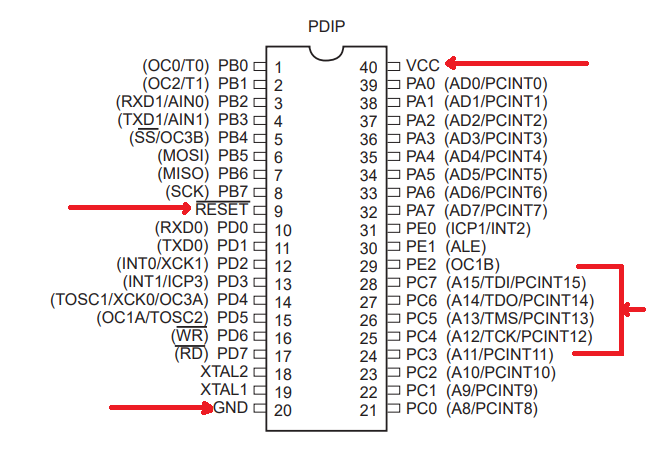


Ilustración 13. Pin Out del ATmega162.

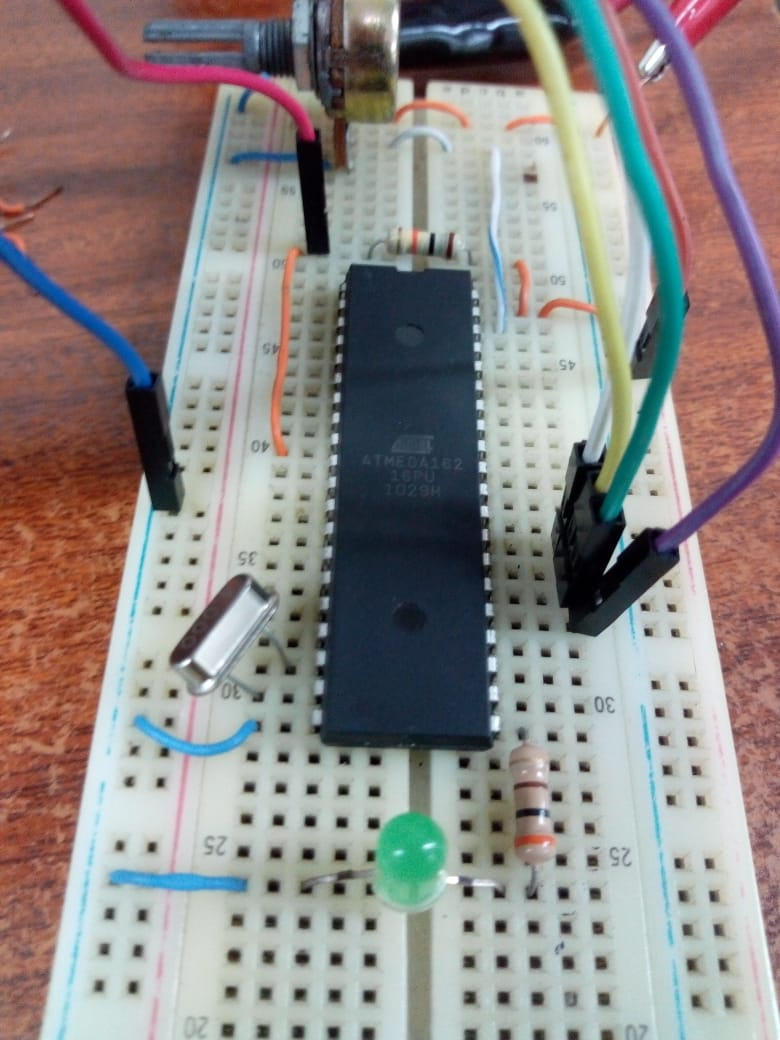


Ilustración 14. Conexion por JTAG al ATmega162

SPI: En la Ilustración 15 se muestran los pines del JTAGIC3 que se usan para realizar la programacion. Al igual que ocurre cuando se va a conectar dos circuitos entre sí, es necesario que compartan la tierra. GND corresponde a la tierra del circuito del microcontrolador. VTG hace referencia a la tensión de alimentación del microcontrolador. /Reset corresponde al pin de reset del microcontrolador, el cual es necesario que pueda ser controlador por el programador. Lo anterior se debe a que el programador puede usar este reset para asegurarse que el CPU este detenido. SCK corresponde a la señal de reloj que se usa para sincronizar la comunicación entre el microcontrolador y el programador. MISO y MOSI son los pines que se usan para el envío y recepción de datos. El protocolo JTAG se basa en el modelo maestro-esclavo. MISO significa Master Input Slave Output mientras que MOSI es Master Output Slave Input.

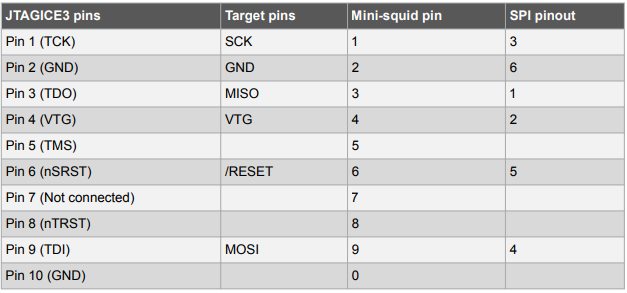


Ilustración 15. Tabla de pines para programación con SPI usando el JTAGICE3.

En la Ilustración 16 se muestran las funciones de cada uno de los pines del microcontrolador ATtiny4313. Se puede observar que la mayoria de los pines desempeñan mas de una funcion a excepcion de los pines de alimentacion VCC y GND. Los pines necesarios para hacer la programacion del microcontrolador con el protocolo SPI son: PA2, GND, PB5, PB6, PB7 y VCC. Se puede observar que estos pines ademas de servir para la programacion del microcontrolador tambien pueden ser usados como salidas o entradas digitales

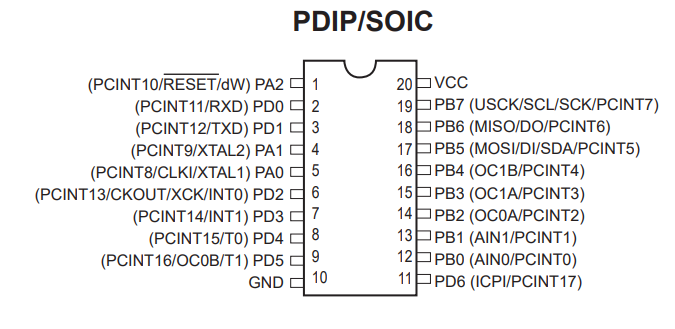


Ilustración 16. PinOut del microcontrolador AVR-8 bits ATtiny4313.

En la Ilustración 17 se muestra la conexión de la alimentación y un cristal de cuarzo a un microcontrolador ATtiny4313. En la Ilustración 16 se puede ver que los pines donde se debe conectar el cristal son los que están etiquetados con XTAL, es decir los pines PA0 y PA1. Para llevar a cabo la programación de un microcontrolador de la familia AVR es necesario que estos dispongan de una señal de reloj. Normalmente los microcontroladores de esta familia están configurados por defecto para usar un oscilador de cristal, sin embargo, esta configuración se puede cambiar. Esto se explicará más adelante en la sección de clock fuses.

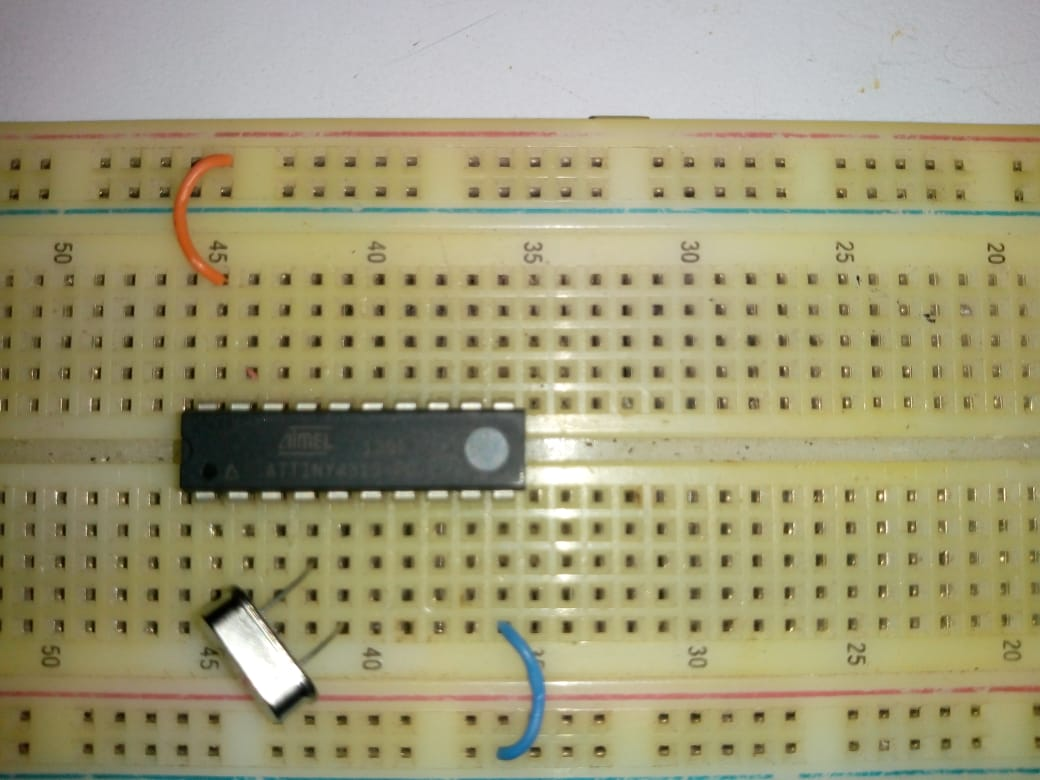


Ilustración 17. Conexión de la alimentación y el cristal de cuarzo para el microcontrolador ATtiny4313.

En la Ilustración 19 se muestra la conexión del programador al microcontrolador para poder programarlo. Se puede observar que se agrega una resistencia de pull-up entre Vcc (alimentación positiva) y el pin de reset. El esquema eléctrico se muestra en la Ilustración 18.

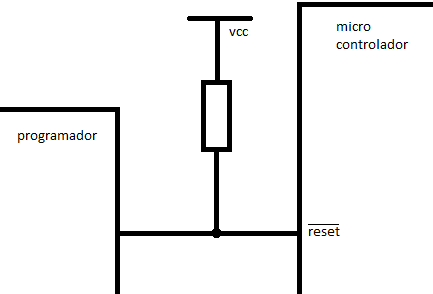


Ilustración 18. Conexión de la resistencia de Pull-Up para el pin de reset.

Para hacer un reset del microcontrolador se debe establecer un nivel lógico bajo (GND). Cuando se desea operación normal del microcontrolador se debe poner un nivel lógico alto en el pin de reset. Y que el programador debe hacer un hacer uso de este pin, será necesario imponer en el mismo niveles altos y bajos durante la fase de programación. Por lo anterior se usa la resistencia de pull-up. En la figura Ilustración 19 se puede ver que el programador posee 3 leds, pero solo el de la mitad esta encendido. Este led indica que el programador esta conectado a una fuente de alimentación.

|  |  |
| --- | --- |
|  |  |

Ilustración 19. Conexión del programador JTAGICE3 usando el protocolo JTAG.

En la Ilustración 20 se muestra cómo se enciende el led verde del programador cuando se hace tiene un valor correcto de alimentación para el chip.

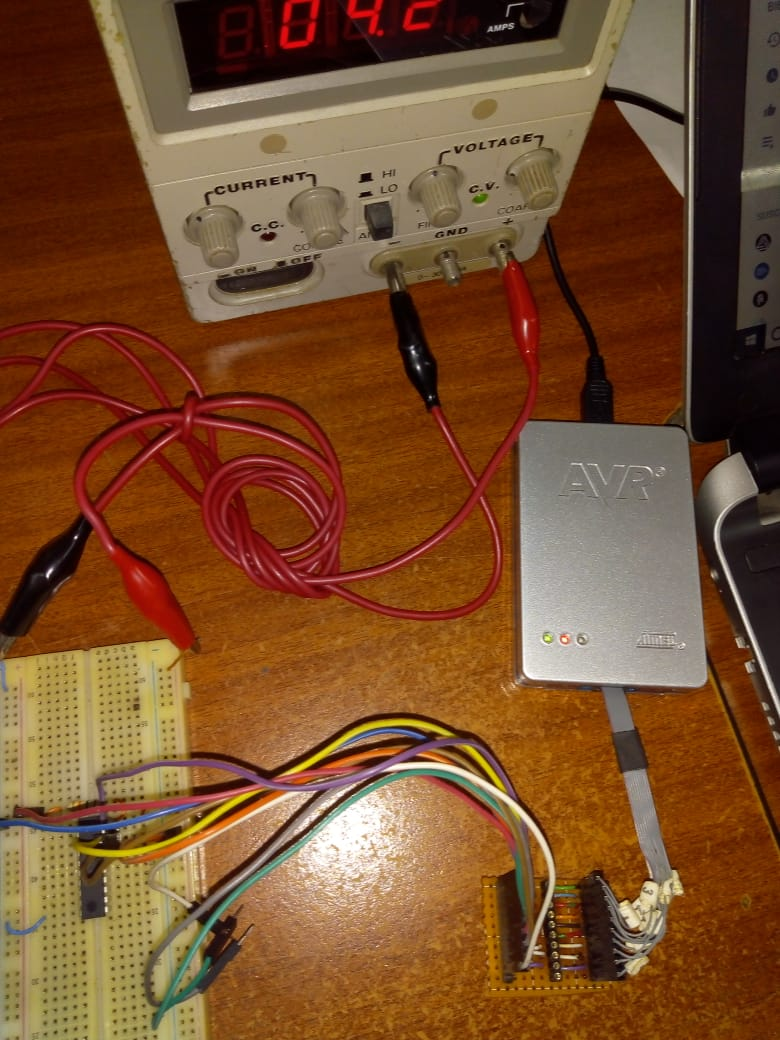


Ilustración 20. Programador y microcontrolador correctamente conectados para llevar a cabo la programación.

Una vez se han hecho las conexiones necesarias, se procede a ejecutar atmel studio y abrir la ventana de programacion de dispositivos (Ctrl+shift+p). En la Ilustración 21 se muestra en la esquina superior izuierda que en la opcion tool se seleccióna el programador, en este caso el JTAGICE3. Ademas se debe seleccionar la referencia del microcontrolador que es ATtiny4313. Este microcontrolador solo soporta programacion por SPI. Sin embargo la mayoria de las referencias de AVR que no son de la familia Tiny soportan JTAG y SPI. Tambien se puede observar en la misma figura que se selecciona SPI como interfaz de programacion. Una vez se selecciona la interfaz de programacion se debe dar click en el boton de aplicar (apply). Seguido de esto, donde dice device signature se debe dar click en read. Si la conexión esta bien hecha y el microcontrolador tiene el cristal de la frecuencia correcta entonces se mostrara un numero en hexadecimal en la casilla device signaure. Esto indicara que el disposittivo esta listo para ser programado.

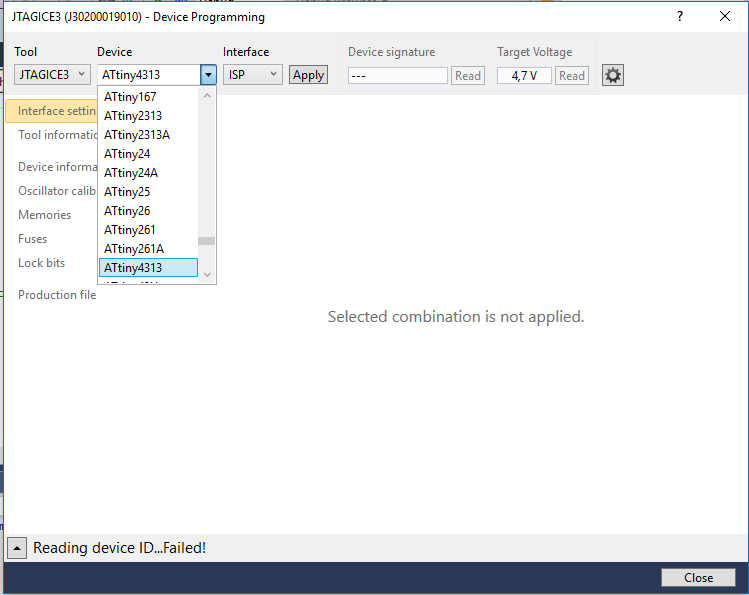


Ilustración 21. Selección del dispositivo a programar.

Debido a que la interfaz SPI no soporta funciones de depuración, es necesario usar la interfaz debug wire.

## Escribiendo la Memoria De Programa

Lo primero que se debe hacer es compilar la aplicación sin errores y preferiblemente sin advertencias (warnings). La compilación se puede lograr con el shortcut Ctrl + p. Una vez se logra conexión con el microcontrolador como se muestra en la sección anterior, se debe abrir la sección memoria como se muestra en la Ilustración 22. Finalmente basta con hacer click en el botón “Program”, tal como se muestra en la ilustración. El programa empezara a ejecutarse tan pronto termine el proceso de programación, por lo anterior es importante tener precauciones en caso de que el microcontrolador este accionando actuadores de alta potencia o dispositivos susceptibles de dañarse debido a errores en el programa.

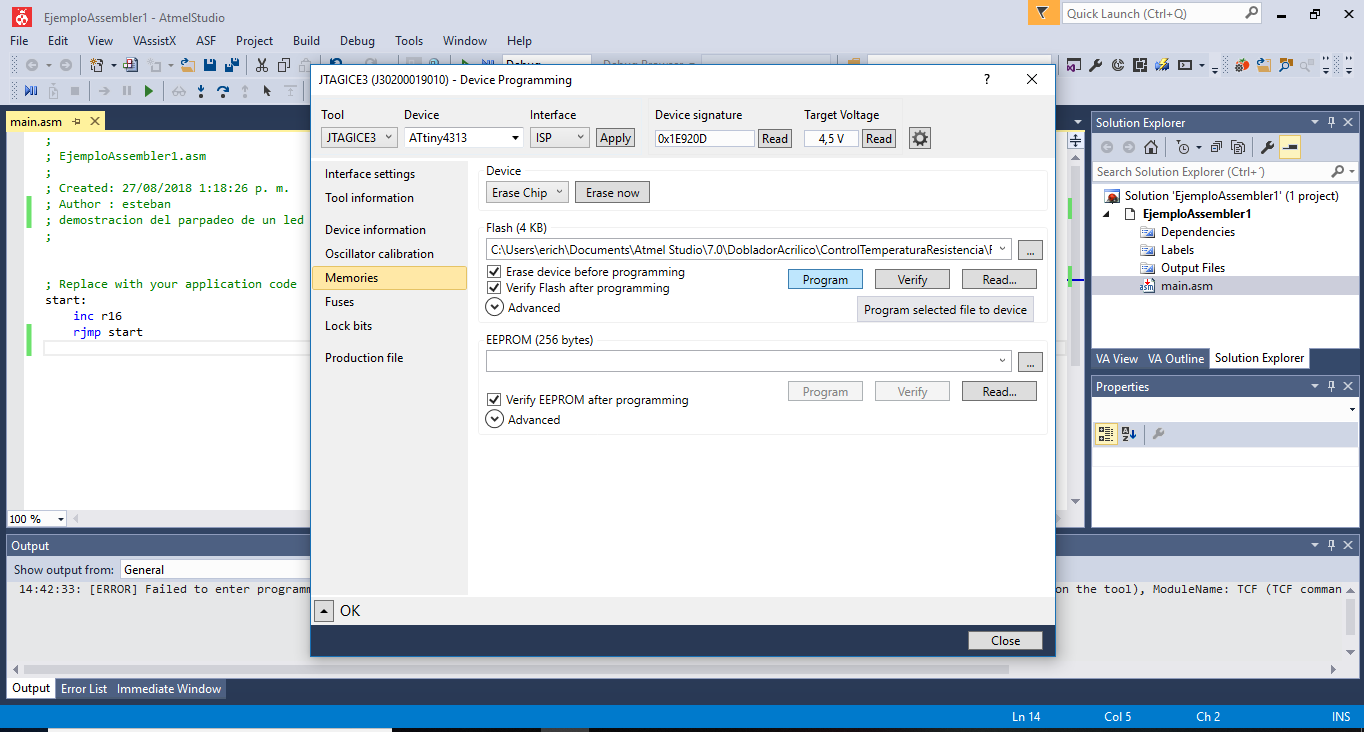


Ilustración 22. Programación del microcontrolador usando JTAGICE3.

## Programación de Fuses

La mayoría de las funcionalidades del microcontrolador se configuran estableciendo valores en los registros del mismo. Sin embargo, en el caso de Atmel hay algunas características que se configuran en unas memorias separadas a las que llaman “fuses” (de ahora en adelante fusibles). Estas regiones de memoria no son accesibles durante ejecución, por consiguiente, no es posible modificarlos mediante instrucciones de máquina.

Entre las características más importantes que se configuran con los fuses están:

* Fuente de reloj para CPU (CPU Clock source)
* Frecuencia de reloj (CPU Clock)
* Brown-Out Detector
* Watch Dog Timer
* Modo Debug Wire
* Memory lock
* Entre otros.

La fuente de reloj puede ser generada usando un cristal de cuarzo o puede generarse internamente en el microcontrolador con un oscilador RC (que no tan preciso como el oscilador de cuarzo). También es posible establecer que la frecuencia de reloj no se genere, sino que esta se recibirá externamente.

El Brown-Out detector es un hardware del microcontrolador que se encarga de detectar cuando ocurre una reducción de la alimentación del microcontrolador. Cuando esto ocurre es posible que algunos de los registros del microcontrolador experimenten cambios deseados en sus bits lo cual puede ser peligroso dependiendo de la aplicación que se esté trabajando. Por este motivo cuando ocurre un bajón de tensión del microcontrolador, lo más seguro es realizar un reset. Cuando el fusible de Brown-Out Detection está habilitado, el microcontrolador hace un reset automático si ocurre una reducción de la tensión por debajo de un nivel especificado.

El Watch-Dog-Timer es un temporizador que realiza un reset del microncontrolador cuando este se desborda. Esta característica es muy importante cuando se está implementado una aplicación que hace uso de operaciones matemáticas muy complejas. En este caso puede ocurrir que en medio de los cálculos matemáticos provoquen que el programa se quede enganchado en un ciclo infinito, y puede ser preferible hacer un reset. Cuando se hace uso del Watch-Dog-Timer siempre se debe hacer un clear del temporizador al finalizar cada ciclo del programa.

Otra característica muy importante a nivel de producción es el Memory-Lock. Cuando se activa este fusible ya no es posible leer ni escribir sobre la memoria de programa y los fuses. Esto puede ser muy útil para evitar que otras empresas realicen espionaje industrial del software programado en los microcontroladores. Sin embargo, si esto se hace en fase de pruebas sería equivalente a perder el microcontrolador pues ya no se podrá reprogramar para hacer ninguna clase de mejoras.

Algunos de los registros mencionados se muestran en la Ilustración 23 e Ilustración 24, se muestra un ejemplo de la configuración de los clock fuses.

# 

Ilustración 23. ventana de configuración de clock fuses.

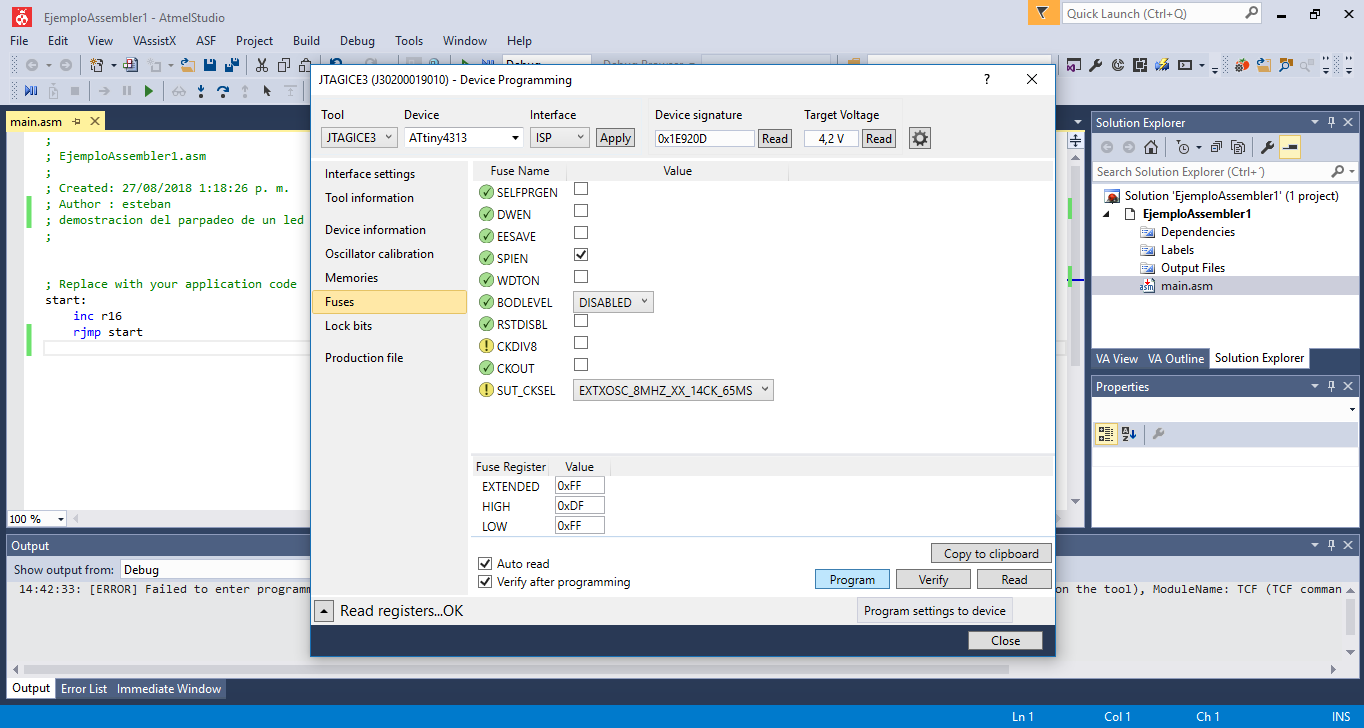


Ilustración 24. Configuración de la fuente de reloj.

En este ejemplo se puede observar que la fuente de reloj esta configurada inicialmente para usar el oscilador interno y además usar un pre-escalador por división 8. Es decir que si el oscilador interno es de 8 Mhz entonces la CPU funcionará a 1 Mhz. En la Ilustración 24 se puede ver que estos parámetros se cambian para que el microcontrolador emplea una señal de reloj de una fuente externa en este caso de un oscilador de cristal. Se puede observar también en la ilustración que el puntero del ratón esta sobre el botón de programar. Una vez se configuran los clock fuses con los valores apropiados es necesario programar el microcontrolador para que estos cambios tengan efecto. Una vez programados estos valores permanecen después de que se pierda la alimentación, es decir que corresponden a la memoria persistente del microcontrolador.

# Proyecto de Ejemplo

Para poner en práctica los conceptos vistos hasta el momento se llevará a cabo el parpadeo de un led usando los 4 tipos de proyectos presentados hasta el momento. Este tipo de proyectos se les suele dar el nombre: “hola mundo de la electrónica”, ya que corresponde al ejemplo más sencillo para el caso hardware.

Mas adelante en esta guía se presentan ejercicios más avanzados.

Independientemente del tipo de proyecto requerido hay una serie de pasos comunes que conviene seguir. En primer lugar, se debe descargar el datasheet del microcontrolador con el que se desea trabajar para revisar que periféricos dispone y su número de pines. De esta forma se puede determinar si el microcontrolador posee los recursos suficientes para la aplicación que se desea desarrollar.

Una vez se sabe con claridad cuáles de los periféricos del microcontrolador se desean utilizar se procede a revisar en el datasheet los registros relacionados y como deben configurarse. Una vez se posee esta información ya sse procede a escribir el programa para el microcontrolador seleccionado. Lo anterior también se ilustrara en los siguientes ejemplos para que el lector se familiarice con el desarrollo de soluciones con sistemas embebidos y pueda generalizar los conceptos de esta guía a otros fabricantes tales como Microchip, STM , NPX, Renesas, Freescale entre muchos otros.

## Parpadeo De Un Led En Ensamblador

Para hacer parpadear un led lo más sencillo es hacer uso de las salidas digitales del microcontrolador. La estructura de los registros puede cambiar de acuerdo al fabricante. En este caso se va a tomar como ejemplo el caso de Atmel. Para las entradas y salidas digitales se usan los siguientes registros: DDRx, PINx y PORTx. El registro DDRx establece que pines van a hacer entradas o salidas digitales. El registro PORTx establece los valores lógicos de los pines que funcionan como salidas digitales. Por último, el registro PINx se usa para leer los niveles lógicos de los pines que operan como entradas digitales.

Normalmente las salidas digitales se agrupan de acuerdo al tamaño de los registros. En el caso de las arquitecturas de 8 bits, se agruparían 8 pines en un puerto. En el caso de Atmel estos grupos de pines se designan con una letra, empezando desde la A. Para este ejemplo se usará el ATmega8A, que es un microcontrolador de 8 bits. El PinOut de este microcontrolador se muestra en la Ilustración 25.

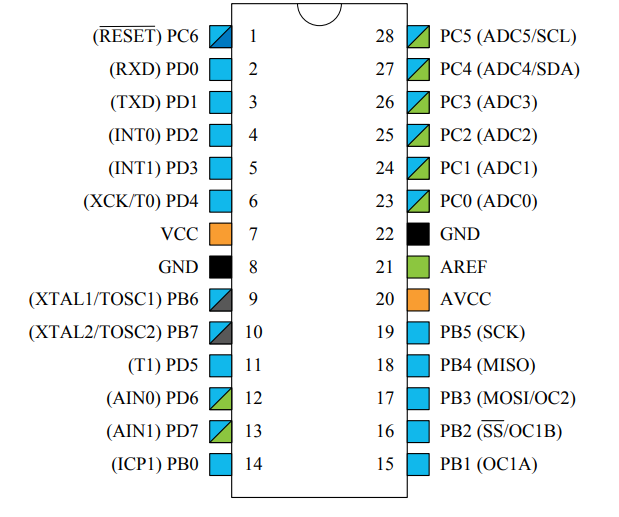


Ilustración 25. PinOut del ATmega8A de atmel.

Este microcontrolador solo soporta el protocolo SPI de programación. En la Ilustración 26 se muestra la conexión del ATmega8A a un led, cristal de 4Mhz y al JTAGIC3 para programarlo. También se puede observar su respectiva resistencia de pull-up a pin de reset. Una vez hecha la conexión se procede a revisar la ficha técnica del ATmega8A, la cual se puede encontrar en la siguiente URL: <http://ww1.microchip.com/downloads/en/DeviceDoc/Microchip%208bit%20mcu%20AVR%20ATmega8A%20data%20sheet%2040001974A.pdf> .

Para este Ejercicio es claro que solo es necesario hacer uso de una salida digital del microcontrolador. En este caso se escogió PB1. (Pin 1 del puerto B)

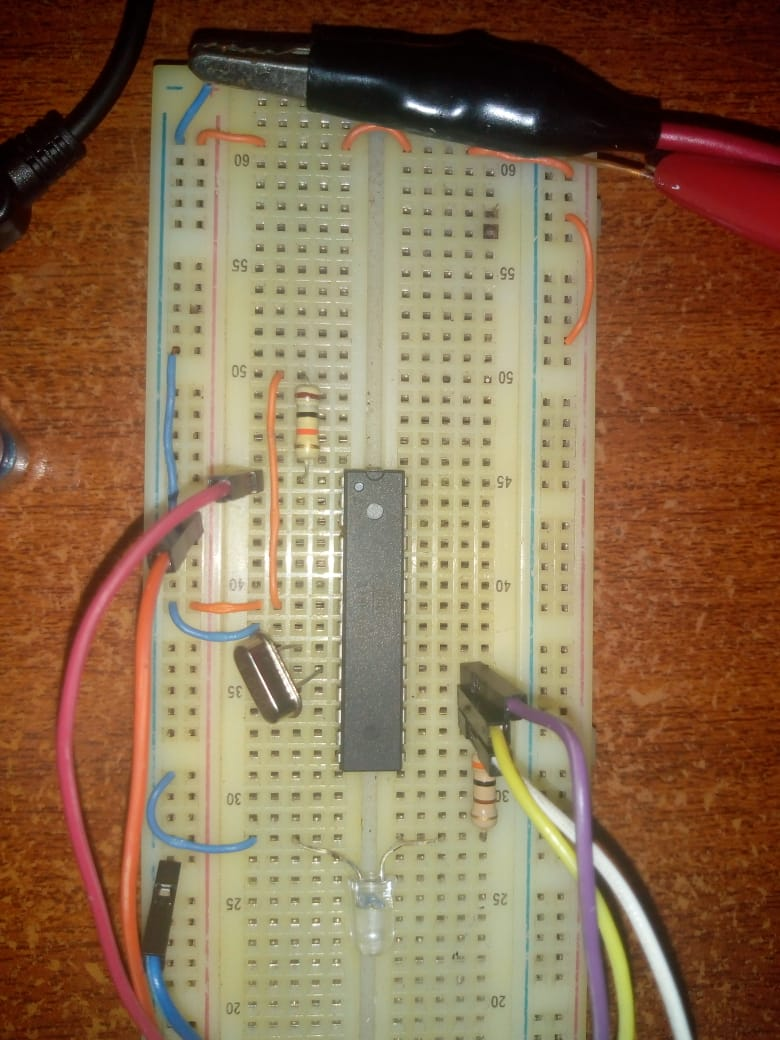


Ilustración 26. Ejercicio de parpadeo de un led con el ATmega8A.

En la página 83 de la ficha técnica se puede ver una explicación del funcionamiento de las entradas y salidas digitales del microcontrolador (I/O). Leyendo la ficha técnica se puede observar que es necesario manipular los registros PORTB y DDRB. Mas abajo se muestra el código implementado en Ensamblador, se puede encontrar este proyecto en el repositorio de B-Learning bajo el nombre “EjemploAssembler1” en la carpeta de proyectos de Atmel Studio 7.

;

; EjemploAssembler1.asm

;

; Created: 27/08/2018 1:18:26 p. m.

; Author : esteban

; YoutubeChannel:

; Instagram: RoughGentleman77

; demostracion del parpadeo de un led usando lenguaje ensamblador.

; para mas informacion ver la guia en este mismo repositorio

.equ PB = 2

start:

ldi r16, PB

ldi r17, 0x00

out DDRB, r16 ; se establece el pin 1 como salida digital.

nop

out PORTB, r16 ; se pone el pin 1 del puerto B en nivel logico alto.

nop

; se implementa un retardo 1048576 ciclos de reloj que corresponde aproximadamente a 1.048576 segundos

eor r18, r18 ; or exclusiva A xor A = 0. esta instruccion hace clear del registro r19 (proposito general).

eor r19, r19

eor r20, r20

loop:

inc r18 ; incrementa +1 al valor del registro r18

cpi r18, 0xff ; se compara r18 con 255

brlo loop ; si en la comparacion da r18 menor a 255 se regresa el program counter a loop:

eor r18, r18 ; cuando r18 ya no es menor a 0xff (255) entonces se pone r18 nuevamente en 0

inc r19 ; se incrementa r19 con +1

cpi r19, 0xff ; se compara r19 con 255

brlo loop ; se regresa el program counter a loop

eor r19, r19 ; se hace clear de r19

inc r20

cpi r20, 0x10 ; se compara r20 con 16

brlo loop ; se regresa el program counter a loop si r20 < 16

out 0x18, r17 ; se pone la salida en 0

; esta es un copy and paste de la porcion de codigo para hacer el retardo

eor r18, r18

eor r19, r19

eor r20, r20

loop2:

inc r18

cpi r18, 0xff

brlo loop2

eor r18, r18

inc r19

cpi r19, 0xff

brlo loop2

eor r19, r19

inc r20

cpi r20, 0x10

brlo loop2

rjmp start ; salta a la primera instruccion

; se define una rutina de delay

La primera línea de código “.equ PB = 2” no se trata realmente de una instrucción ejecutable por el microcontrolador. Corresponde a una directiva para el IDE que le indica cuando encuentre la palabra PB la reemplace por el valor 2. Esta clase de definiciones es muy útil cuando se desarrollan proyectos complejos en lenguaje ensamblador ya que recordar las direcciones de los registros es más difícil que recordar nombres. En este caso se está guardando el número que representa el pin. Esto se explicará más adelante, por ahora solo téngase en cuenta que el valor de PB va a indicar que pin del puerto B es el que se desea configurar. La etiqueta start: corresponde a un nombre que designa una posición en memoria de una instrucción. La primera instrucción ejecutable es ldi r16, PB. En este caso start: sirve para designar la dirección en la memoria de programa donde esta guardada esta instrucción. Se debe tener en cuenta que esta etiqueta no hace parte del programa, simplemente se trata de una característica del lenguaje ensamblador que facilita el trabajo de los desarrolladores. La primera línea de instrucción guarda en el registro r16 el valor 2 (el valor de PB). Luego se asigna el valor de 0 en el registro r17.

Luego con la instrucción out se asigna valores a los registros de salida, DDRB (para configurar el PB1 como salida digital) y PORB (poner un valor lógico alto en PB1).

El comando nop indica al procesador no realizar ninguna tarea por un ciclo de reloj.

Una vez se se asigna a PB2 un valor lógico alto, se debe mantener el led encendido por un periodo de tiempo que permita que el parpadeo del led pueda ser percibido por el ojo humano.

En este ejemplo el microcontrolador emplea un cristal de 4Mhz, la cual es la frecuencia de CPU. Por tanto, será necesario que el microcontrolador espere 4 millones de ciclos de reloj para provocar un retardo de 1 segundo.

Para lograr un retardo visible no es necesario un valor muy preciso de tiempo. En este proyecto se implementó el retardo contando hasta 1048576. Debido a que se trata de un microcontrolador de 8 bits, el número máximo que se puede contar con un registro es 255. Por este motivo para lograr un retardo significativo se hizo uso de 3 registros de 8 bits, r18, r19 y r20. Para iniciar el conteo primero se establecen estos registros en 0. Para esto se usa la instrucción eor, que implementa la operación lógica or exclusiva (xor) bit a bit. Por definición, esta operación lógica convertirá todos los bits en cero cuando se aplica a 2 números iguales. Lo anterior es aprovechado para hacer clear de los 3 registros que se usaran para contar.

La implementación del retardo continúa realizando un incremento del registro r18. Luego se usan 2 instrucciones para verificar si r18 ha llegado a su valor máximo. La primera instrucción es cpi, la cual realiza la comparación de 2 números de 8 bits e indica en un registro aparte 1 o 0 dependiendo de que numero es mayor. Luego se usa el comando brlo el cual hace uso directo del resultado generado por el comando cpi. Si r18 es menor, la instrucción brlo asigna al program counter la posición en la memoria de programa correspondiente a la etiqueta loop. Si, por el contrario, r18 ha alcanzado 0xff, entonces continua con la siguiente instrucción en la memoria de programa. Las siguientes instrucciones repiten el mismo patrón, pero alterando el registro r19. Es decir que r19 está contando el número de veces que r18 llega a 0xff (255). En otras palabras, con r18 y r19 se logra contar hasta 255\*255 = 65025. Por último, se repite este mismo patrón con el registro con r20, con la diferencia que la comparación se hace con 0x10 (16 en decimal). Es decir que r20 cuenta el número de veces que se llega a 65025 con los registros r18 y r19. Al final de esta parte de código se ha contado hasta 255\*255\*16 con ayuda de 3 registros de 8 bits. Durante este conteo se ha mantenido al procesador ocupado para que el led permanezca encendido el suficiente tiempo y el ojo humano pueda percibirlo.

Luego se usa nuevamente la instrucción out para asignar el valor lógico 0 al pin correspondiente al led en el puerto B (registro PORTB).

Seguido de esto se repite la porción de código para llevar a cabo el retardo.

Se debe tener en cuenta que para este microcontrolador las direcciones de DDRB y PORTB son 0x17 y 0x18 respectivamente. En el código de ensamblador presentado anteriormente, el IDE identifica automáticamente las direcciones correspondientes a PORTB y DDRB, sin embargo, sin esta característica del IDE sería necesario escribir estas direcciones en lugar del nombre de los registros.

## Parpadeo de un led usando C/C++

En esta demostración se podrá observar, los programas desarrollados en C/C++ son mucho más sencillos desde el punto de vista del programador en comparación a los proyectos desarrollados en ensamblador. Esto se debe a que C/C++ es un lenguaje de alto nivel que le permite al programador desarrollar algoritmos con instrucciones que ya no dependen del procesador con el que se está trabajando. Una vez el algoritmo esté terminado, ya será trabajo del compilador convertirlo a lenguaje de máquina. Esta capa de abstracción libera en parte al programador de los aspectos minuciosos del manejo de memoria. A continuación, se presenta la versión en código C++ para lograr el parpadeo de un led. Para este ejemplo se usa el mismo microcontrolador, y la misma configuración del ejemplo pasado.

#include <avr/io.h>

#define *F\_CPU* 4000000UL //(1Mhz) macro para definir la frecuencia de trabajo del microcontrolador. se usa en la implementacion del comando delay.

//#define \_BV( bit ) ( 1<<(bit) ) // macro para facilitar la escritura de los comandos para leer y escribir sobre registros del microcontrolador.

#include <util/delay.h> // libreria de AVR para hacer retardos.

int main(void)

{

DDRB |= \_BV(1); // se asigna el pin 1 del puerto B como salida digital

while (1)

{

PORTB |= \_BV(1); // se asigna un valor logico alto a PB1

*\_delay\_ms*(1000); // un retardo de 1000 ms

PORTB &= ~\_BV(1); // se asigna un nivel logico bajo a PB1

*\_delay\_ms*(1000); // se hace un retardo de 1000 ms

}

}

En la primera línea se incluye la librería estándar para trabajar con microcontroladores AVR. Esta librería contiene los nombres de los registros y algunas funciones. Esto permite que se puedan usar los nombres de los registros, que son más fáciles de reconocer que las direcciones.

La siguiente línea es la definición de la frecuencia de CPU. Esta corresponde a una directiva para el ide y es usada por la librería útil/delay. Esta librería contiene funciones que permiten implementar retardos. En la sección anterior se pudo observar que es necesario conocer la frecuencia de trabajo del procesador para poder escribir el código que implementa el retardo. Por este motivo es necesario definir la frecuencia de reloj antes de importar la librería de delay. Esto permite que las funciones de la librería implementen el retardo por el tiempo correcto.

Por último se puede observar en la sección de cabecera una definición para el ide que esta comentada, #define \_BV( bit ) ( 1<<(bit) ) . Esta definición esta comentada ya que está incluida en la librería delay. Si no se incluye esta librería se recomienda incluir esta definición ya que facilita la escritura de números binarios. La convención que se emplea consiste en usar 0b al inicio si se trata de un numero en formato binario o 0x su se trata de un número en formato hexadecimal. Se puede considerar a manera de ejemplo el numero 0xff, que en formato decimal es 255. En formato binario seria 0b11111111, que corresponde a todo un registro de 8 bits. El número \_BV(0) es igual a (0b00000001). De la misma forma \_BV(1) = 0b00000010, \_BV(2) = 0b00000100 … \_BV(7) = 0b10000000. Se puede observar que \_BV corresponde a un número binario con un solo bit en estado activo. El argumento de \_BV indica que bit es el que debe quedar con un nivel lógico alto. Para ver porque esta representación puede ser conveniente (aunque no necesaria) vamos a analizar el registro DDRB. Como se ha mencionado antes los registros son de 8 bits en este caso, por tanto, el registro DDRB puede controlar hasta un máximo de 8 pines. Si numeramos estos pines de 0 a 7, \_BV(n) establecería el pin n como salida digital, donde n seria un numero de 0 a 7.

Esto es lo que se muestra en la primera línea de código del main. Se puede observar que se hace uso de |= en lugar de =. Para este proyecto daría el mismo resultado usar cualquiera de las asignaciones, sin embargo, es más común la primera forma, que es la que se usa en este ejemplo. El símbolo “ | “ denota la operación OR bit a bit. De esta forma \_BV(0) | \_BV(1) = 0b00000011 . La instrucción DDRB |= X realiza la operación OR bit a bit de lo que hay actualmente en DDRB con X, el resultado se asigna nuevamente a DDRB. Se puede recordar que la operación OR solo tiene efecto si se realiza con el bit 1, de lo contrario el bit con el que se hace or queda inalterado como se muestra en la tabla más abajo.

|  |  |  |
| --- | --- | --- |
| A | B | Salida |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 0 | 1 |

Este tipo de asignación se usa cuando se desea alterar solo un bit de un registro sin modificar los demás bits. En este caso como solo se desea modificar los valores relacionados con el pin 1 del puerto B no hay ningún problema con asignar a los demás bits del registro el valor 0. Sin embargo, en programas muy largos esto podría dar lugares a errores, ya por descuido, el programador podría alterar bits equivocados y hacer que su programa no funcione correctamente. Otro motivo para usar el operador OR en lugar de la asignación directa es que hay ciertos registros en los cuales no todos los bits se pueden modificar ya que parte de estos bits pueden ser de uso reservado para el procesador y como consecuencia el sistema podría comportarse de manera no deseada. A continuación, se presenta otro ejemplo. Supongamos que es necesario establecer el los bits 5 y 2 de un registro sin alterar los demás bits. Esto se lograría con la instrucción [Registro] |= \_BV(5) | \_BV(2). En la siguiente tabla se muestra esto de una forma gráfica.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Registro | x7 | x6 | x5 | x4 | x3 | x2 | x1 | x0 |
| \_BV(5)|\_BV(2) | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| resultado | x7 | x6 | 1 | x4 | x3 | 1 | x1 | x0 |

Hacer operación OR con 0 permite dejar el bit inalterado, mientras que hacerlo con 1 permite establecer valor lógico alto sin importar el valor del bit.

Se puede observar en el código que después de establecer el pin PB1 como salida digital y un asignar un valor lógico alto, se usa la función \_delay\_ms la cual lleva a cabo un retardo en milisegundos del tiempo especificado. En este ejemplo se estable el retardo de 1 segundo. Luego de esto es necesario llevar nuevamente el bit correspondiente a PB1 a cero. Para esto se usa la operación AND en lugar de OR. Cuando se hace AND con 1 el bit del registro que inalterado, pero cuando se hace and con 0 el bit del registro queda en cero sin importar su valor. En este caso si se desea hacer clear del bit 1 del registro PORTB. El carácter “ ~ “ corresponde a la negación bit a bit. Por tanto ~\_BV(1) = 0b11111101. Al hacer la operación AND del registro PORTB con este número binario se logra que el bit correspondiente a PB1 tome el valor de 0 mientras que los demás permanecen inalterados.

Se puede observar que la versión en C++ es mucho más compacta y fácil de leer que su equivalente en lenguaje ensamblador. Esto será incluso más evidente cuando se trate de proyectos de mayor envergadura. Normalmente se recomienda el uso de C/C++ para la mayoría de los proyectos debido a que el tiempo de desarrollo es más corto y fácil de entender.

# Uso del simulador con Atmel Studio 7

## Conceptos básicos para el uso del simulador

Uno de los primeros pasos para validar el programa diseñado para un microcontrolador dado es realizar una simulación. Esto puede servir para verificar que los comandos alteren los registros correctos del microcontrolador bajo ciertas condiciones de operación.

Para hacer un primer acercamiento a la herramienta se va a analizar a continuación el ejemplo más sencillo que es parpadear un led. En la Ilustración 27 se muestran resaltadas las opciones de herramienta y la opción de “start debugging and break”. Inicialmente, para el caso de un proyecto en blanco, la opción de herramienta aparecerá en blanco. Al dar click en ella se abrirá una nueva pestaña en la que se deberá seleccionar la opción simuladora (simulator). Seguido de esto se debe dar click en la el botón “start debugging and break”. El IDE iniciara la compilación del proyecto. Si la compilación es exitosa entonces se lanzará el simulador.

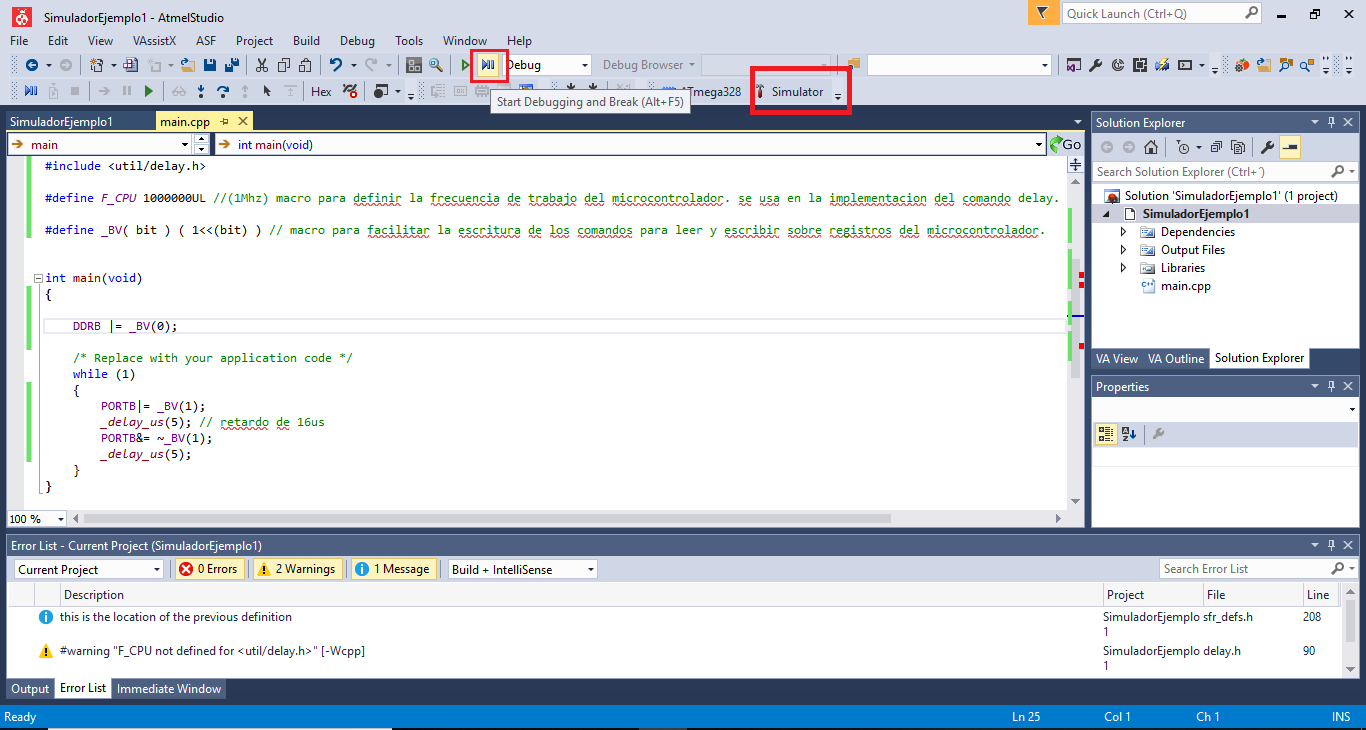


Ilustración 27. Ejemplo para el uso del simulador con Atmel Studio 7.

En la Ilustración 28 se puede ver lo que ocurre cuando se entra en el modo de simulación. En primer lugar, se puede observar que aparecen 2 secciones adicionales, **Memory** e **I/O.** La sección memory muestra la memoria de programa en números hexadecimales. La sección I/O muestra los registros de la referencia de microcontrolador seleccionada (esto se configura cuando se crea el proyecto, pero se puede modificar en la opción device).

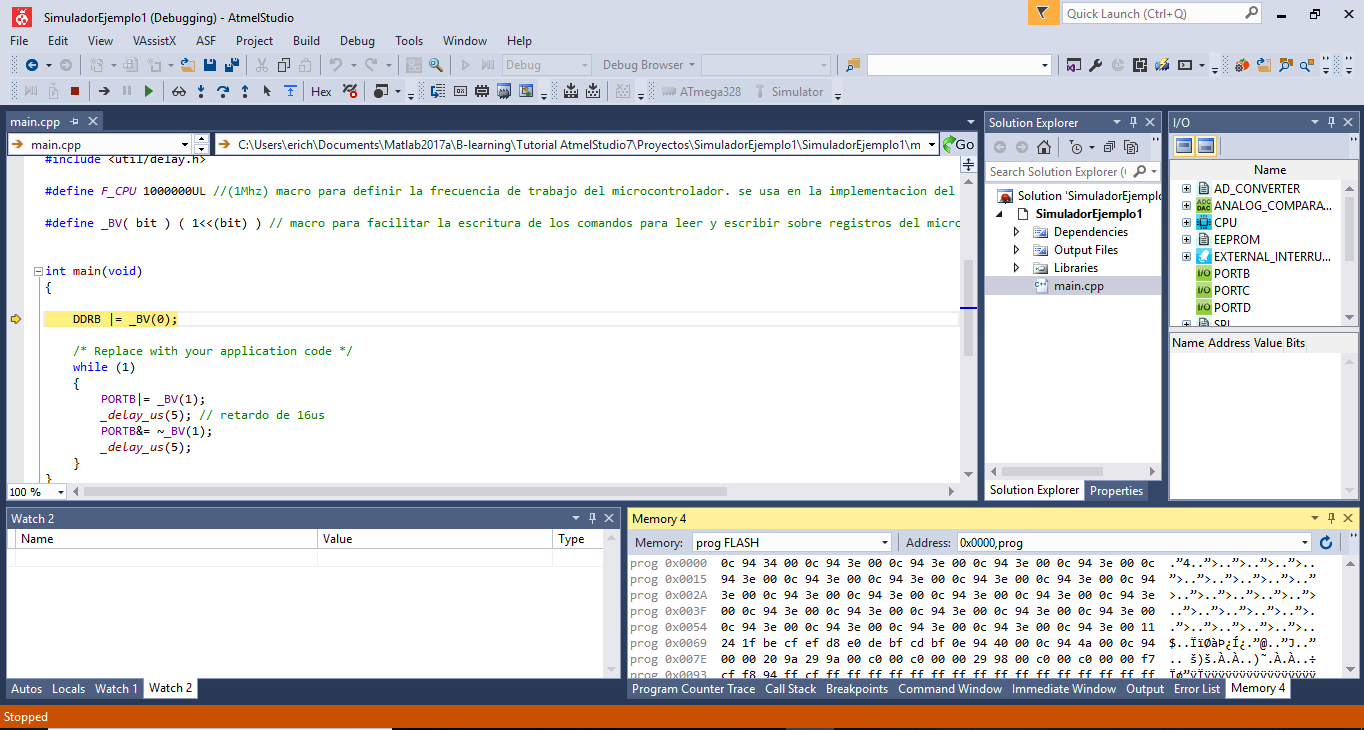


Ilustración 28. Atmel Studio en modo simulación.

También se puede ver en el método main, en la sección del source code, queda una línea resaltada con una flecha amarilla. Esta línea representa en este caso la primera línea de código ejecutable del programa, ya que apenas se inicio la simulación. Este primer comando establece el pin 0 del puerto B como salida digital. En la Ilustración 29 se muestra el uso del botón step over, el cual permite avanzar entre líneas de código. En dicha ilustración se puede ver que el simulador avanza al siguiente comando, el cual establece un nivel lógico alto en el pin 1 del puerto B. Si se ejecuta nuevamente un step over se pasa a la línea del delay. Delay es un comando que se usa para que el procesador espere una cantidad de tiempo dada. La siguiente línea del programa pone un nivel lógico bajo en el pin 1 del puerto B. Por defecto en la parte derecha del IDE se muestran los registros especiales del microcontrolador. Estos registros son los que permiten configurar periféricos del microcontrolador y algunas de sus funciones especiales. En este ejercicio de simulación se estudiarán solo los registros de entradas y salidas digitales.

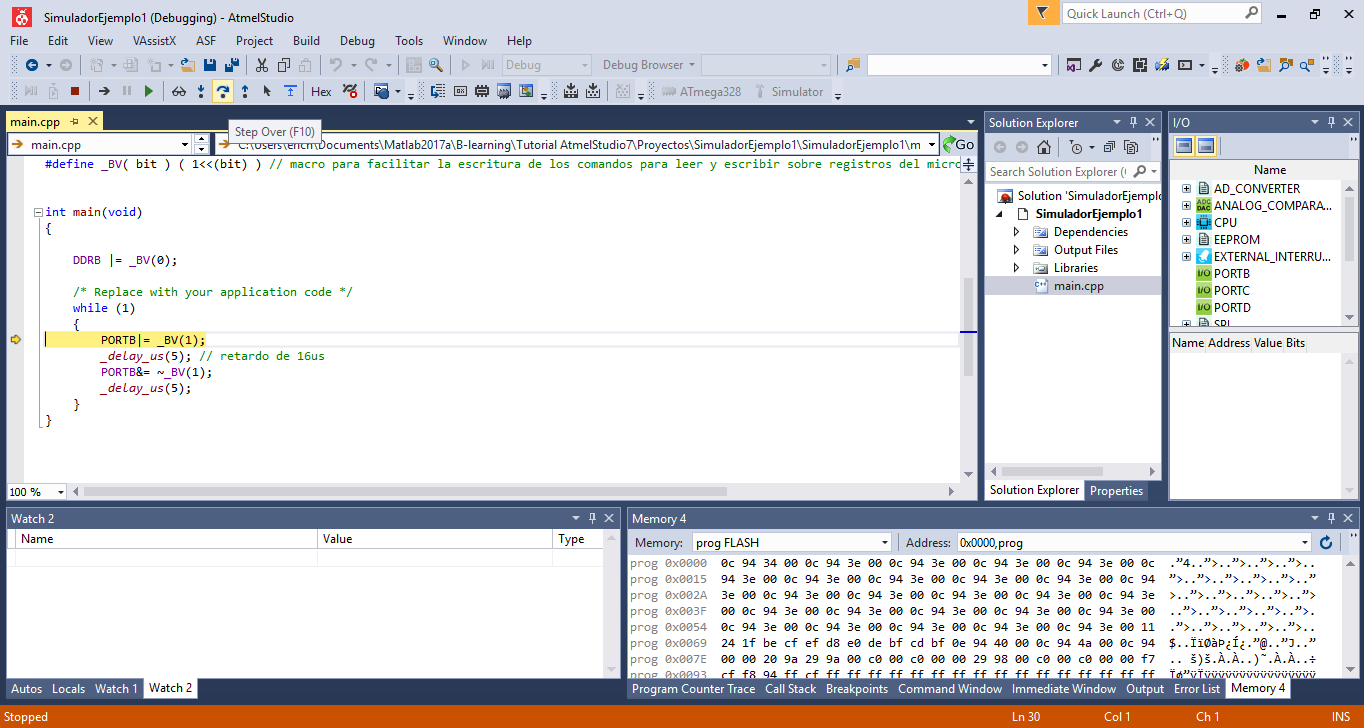


Ilustración 29.Demostración del comando Step Over en Modo simulación.

En la Ilustración 30 e Ilustración 31 se puede ver en el panel izquierdo como cambian los valores de los registros del puerto B conforme se alterna entre las líneas de código dentro del ciclo while. También se pueden observar en ambas ilustraciones unos círculos de color rojo en el lado izquierdo del IDE. Estos círculos representan Break Points. Estos puntos se configuran al hacer click sobre la barra izquierda de color gris en la ventada del source code. Estos puntos se usan solo para simulación o depuración y le indican al IDE que una vez la ejecución del programa llegue a estos puntos, la simulación o la depuración debe hacer una pausa.

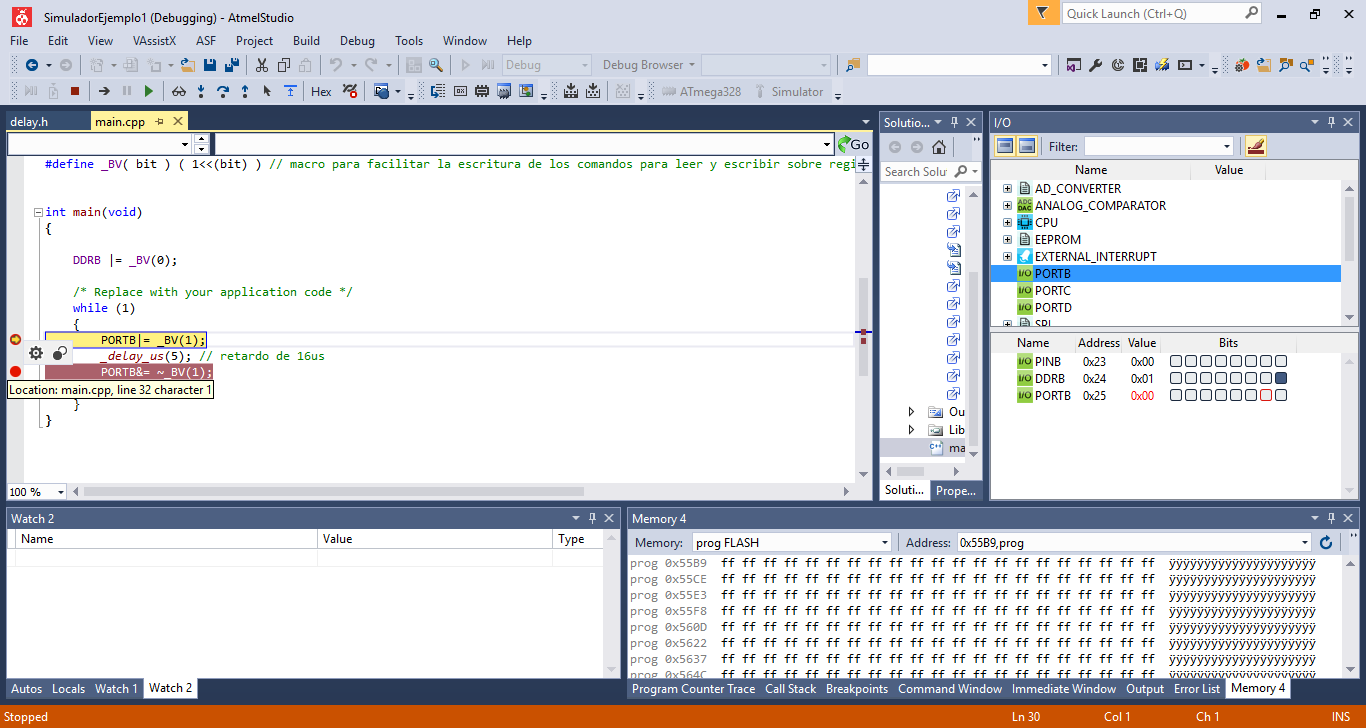


Ilustración 30. Demostración de Break Points en modo simulación.

En la Ilustración 31 se muestra señalado el comando de continuar (continue) representado con un triángulo verde. Este comando le indica al IDE que en caso de

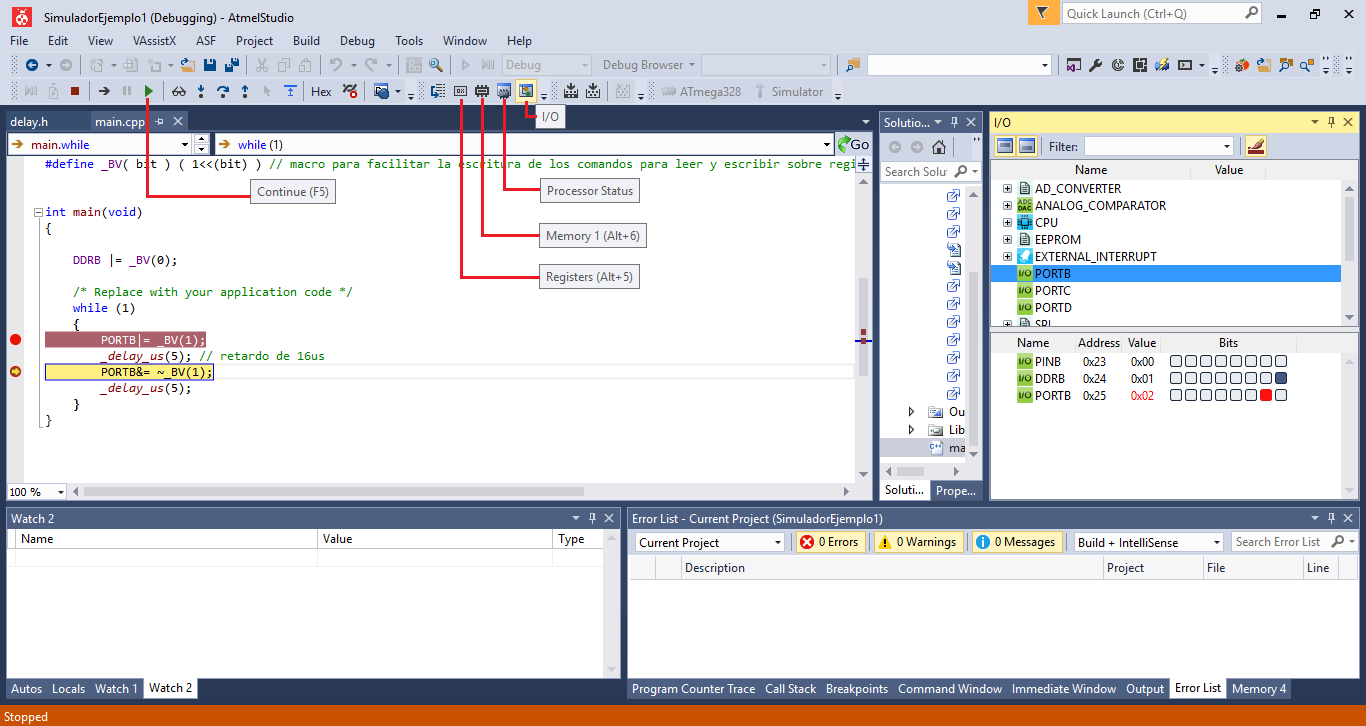


Ilustración 31. Alteración de los registros de entradas y salidas digitales durante una simulación.

Hacer una simulación o depuración, debe continuar con la ejecución completa del programa, en lugar de parar en la siguiente línea de código. Cuando se usa esta opción es necesario hacer uso de break points ya que de lo contrario la ejecución no se detendrá. En la misma ilustración también se resalta la opción processor status, Ilustración 32. En esta ventana se puede ver información importante sobre el procesador, ya sea en modo simulación o depuración. Uno de estos parámetros es el program counter, el cual indica en que punto que instrucción de la memoria de programa se encuentra en procesador.



Ilustración 32. Ventana de processor status.

Por último se debe notar que en este ejemplo se estableció como salida digital el pin 0 mientras que el pin al que se le modificaba el valor de tensión era el pin 1. Esto significa que, al ejecutar este programa en la vida real, el pin 1 no cambiara de valor a pesar de que efectivamente el valor correspondiente en el registro es alterado. Para que este ejemplo permita hacer el parpadeo de un led es necesario que se modifique el valor del pin 0 en lugar del pin 1. Otra opción seria establecer el pin 1 como salida digital en lugar del pin 0. Esto Permite mostrar que, aunque el simulador es una gran herramienta para encontrar errores en el algoritmo diseñado, también es necesario que el programador cuente con los conceptos fundamentales.

# Proyectos De Ejemplo Atmel Studio 7

## Temporizadores

Los temporizadores son circuitos digitales que permiten contar eventos. Cuando esos eventos son el flanco de subida de una señal cuadrada periódica entonces el temporizador permite medir el tiempo transcurrido. Es muy importante distinguir que a pesar de que un temporizador se encuentre en el mismo chip del microcontrolador, se trata de entidades diferentes. Precisamente esto los hace muy útiles ya que al tratarse de hardware independiente a la CPU, permite contar eventos sin llevar a cabo ninguna instrucción máquina directamente. De manera muy similar ocurre con los demás periféricos de los microcontroladores, se introducen como entidades separadas dentro del mismo chip para extender las funcionalidades del microcontrolador. En la siguiente sección se introducirá el concepto de interrupción y se mostrará un ejemplo sobre el uso de temporizadores.

## Configuración de interrupciones

Las interrupciones son llamados que se hacen al procesador para que atienda un evento que requieren atención inmediata. Normalmente las interrupciones son adecuadas para atender eventos cuyo momento exacto de ocurrencia no es conocido, que pueden ocurrir esporádicamente o con una frecuencia baja en comparación con otras tareas de mayor interés y que además son de alta importancia para que aplicación funcione correctamente. Gracias a las interrupciones es posible distribuir de una manera mas eficiente el tiempo que el procesador debe dedicar a las diferentes tareas

Esta sección se limita a la implementación de las interrupciones con microcontroladores de AVR usando la herramienta Atmel Studio 7, los aspectos teóricos de las interrupciones deberán abordarse en cursos de fundamentos de computación.

La configuración de interrupciones en Atmel involucra en primer lugar la configuración de varios registros de acuerdo a la fuente de interrupción requerida. Las fuentes de interrupción más comunes son las relacionadas con temporizadores, pero también hay interrupciones relacionadas con otros periféricos como el ADC.

Para este ejemplo de interrupciones se usará el microcontrolador ATmega8A junto a uno de sus temporizadores. Se realizará el mismo ejercicio de hacer parpadear un led pero sin hacer uso de los retardos. Uno de los principales problemas con los retardos es que estos desperdician tiempo de computo del procesador. Por el contrario, el uso de interrupciones permitirá encender y apagar el led durante intervalos de tiempo precisos mientras que se el procesador permanecerá disponible para hacer otras operaciones. Abajo se muestra el programa de demostración para interrupciones. Este puede ser encontrado en el repositorio de B-learning bajo el nombre InterrupcionesEjemplo1.

#include <avr/io.h>

#include <avr/interrupt.h> // libreria de AVR que facilita uso de interrupciones

#define *F\_CPU* 4000000UL //(1Mhz) macro para definir la frecuencia de trabajo del microcontrolador. se usa en la implementacion del comando delay.

// #define \_BV( bit ) ( 1<<(bit) ) // macro para facilitar la escritura de los comandos para leer y escribir sobre registros del microcontrolador.

#define COUNTER\_INIT 61630 // valor a partir del cual debe empezar a contar el temporizador.

int main(void)

{

DDRB |= \_BV(1); // se establece el pin PB1 como salida digital

sei(); // funcion que configura registros para habilitar interrupciones.

TCNT1 = COUNTER\_INIT; // se inicializa el temporizador en 616630

TIMSK |= \_BV(2); // se habilita la interrupcion de overflow en temporizador 1

TCCR1B |= \_BV(2) | \_BV(0); // se establece fuente de reloj con preescalamiento de 1024

while (1)

{

// se queda enganchado en el while sin hacer nada.

}

}

// rutina de interrupcion cuando se desborda el temporizador 1

ISR(TIMER1\_OVF\_vect){

TCNT1 = COUNTER\_INIT; // se recarga nuevamente el temporizador en 61630

PORTB ^= \_BV(1); // se articula el valor del bit 1 en el puerto B.

}

Lo primero que se debe notar es la importación de la librería avr/interrupt. Esta librería contiene funciones y definiciones que facilitan el trabajo con interrupciones. En el caso de Atmel estudio se debe crear una función o rutina llamada ISR, que significa Interrupt Service Routine. Es obligatorio que la función sea llamada ISR, de lo contrario el IDE marcara error. En el argumento de la función ISR se pone la fuente de interrupción que se desea capturar. En este caso se trata del desbordamiento del temporizador 1. El Desbordamiento ocurre cuando el temporizador está en su valor máximo (todos sus bits en 1) y cuenta un siguiente evento. Dado que todos los bits están en 1 el carry hace que todos los bits retornen 0, al no disponerse de un bit mas significativo, ya que el temporizador tiene un numero finito de bits entonces el valor del temporizador retorna a 0. A lo anterior se le llama desbordamiento.

El temporizador 1 del ATmega8a es de 16 bits. Por tanto, puede contar 2^16 ciclos de máquina. Adicionalmente el temporizador posee unos divisores de frecuencia que se configuran con el registro TCCR1B. A estos divisores se les llama pre-escaladores. Para este ejemplo se utilizó el pre-escalador de 1024 bits. Debido a que la frecuencia de reloj es de 4Mhz, la frecuencia que le llega al temporizador después del divisor de frecuencia es 4e6/1024 = 3906.25. Es decir que cuando el temporizador cuente 3906 pulsos habrá transcurrido aproximadamente 1 segundo. Se puede observar que habrá un error debido a la parte decimal que no se puede cuantificar con el temporizador. Por este motivo cuando se necesitan tiempos muy exactos se debe seleccionar la frecuencia de cristal correcta de forma que el tiempo que se desea medir pueda expresarse en un múltiplo entero de la frecuencia de reloj.

Debido a que el temporizador puede contar 2^16, lo que se hará es asignar al temporizador el valor 2^16 – 3906 = 61630. De esta forma el temporizador se desbordará después de contar 3906 pulsos. Una vez se desborde el temporizador se, en la rutina de interrupción se prendera/apagará el led y además se asignará nuevamente 61630 al temporizador. Es importante asignar el valor del temporizador tan pronto se ejecuta la rutina de interrupción para no incurrir en imprecisiones de tiempo. Además, se debe tener en cuenta el número de ciclos que toma asignar el valor al temporizador para tenerlo en cuenta. Debido a que en este ejercicio no se requiere precisión en los tiempos al tratarse del parpadeo de un led no se tuvo en cuenta estos detalles.

Se puede ver que el argumento de la rutina de interrupción ISR es TIMER1\_OVF\_vect. Los nombres y direcciones para cada interrupción se deben buscar en la ficha técnica del microcontrolador. Para el caso del ATmega8A se pueden encontrar los vectores de interrupción en la paginas 68 y 69. Cuando se empieza a escribir el nombre del vector de interrupción en ISR se puede usar el shortcut CTRL + Space Bar para usar la función de autocompletar de Atmel studio. Con este shortcut se puede ver con más facilidad el nombre correcto que se debe usar en el argumento de la rutina de interrupción.

Para activar las interrupciones se usa la función sei(). Esta función habilita las interrupciones globales.

Además, es necesario configurar los registros del temporizador, lo cual se debe verificar también en la ficha técnica. Para este microcontrolador se pueden ver los registros del temporizador 1 a partir de la página 134. Para este ejercicio los registros más importantes son TCNT1L y TCNT1H, que son 2 registros de 8 bits donde se cuentan los eventos del temporizador. TCCR1B permite configurar la fuente de reloj del temporizador entre otras cosas. Finalmente, el registro TIMSK que permite configurar que tipo de interrupciones están habilitadas para el temporizador 1. En la página 139 de la ficha técnica del ATmega8a se puede verificar los valores que se usaron para TCCR1B. Finalmente en la rutina de interrupción se puede ver que se hace la recarga del temporizador y luego se hace una articulación del bit correspondiente a PB1 con el operador ^ (XOR).

## Uso de Convertidores Analógico Digital (ADC)

Para hacer uso del convertidor analógico digital del microcontrolador basta con establecer los valores apropiados en los registros del ADC. Para esto es muy importante tener en cuenta los fundamentos básicos del proceso de conversión Analógico digital. Para este caso vamos a tomar de ejemplo el microcontrolador atmega328, el cual ha sido muy popular gracias al Arduino 1. Antes de empezar con los registros es necesario realizar una conexión básica en el hardware.

Después de las conexiones básicas de alimentación y resistencia de pull-up para el pin de reset, es necesario que el ADC disponga de una tensión de referencia.

## Uso de Convertidores Digital a Analógico (DAC)

Para el caso de los convertidores Digital Analógico, el manejo se limita a seguir lo especificado en la documentación del microcontrolador. Sin embargo, si el microcontrolador no tiene disponible un DAC, es posible usar una señal del tipo PWM en combinación con un filtro pasa-bajo para lograr el mismo efecto. En la Ilustración 33 se muestra el concepto. La señal azul corresponde a una señal de pwm hipotética que se puede generar con un microcontrolador. Si se pasa la señal cuadrada por un filtro pasa bajo, como podría ser el caso de un circuito RC, la señal cuadrada se transformará en la señal de color rojo. En la ilustración se puede ver que la señal de color rojo sigue una forma exponencial, donde el valor de estabilización depende de la duración del ciclo útil positivo. La amplitud de las variaciones de la señal roja se muestra exageradas en la ilustración para mostrar como el capacitor se carga y se descarga durante los ciclos positivos y negativos de la señal cuadrada. Sin embargo, en la practica bastara con usar una frecuencia de pwm suficientemente alta y un filtro pasa bajo adecuado para lograr una señal limpia. Mediante este método la transformación de digital a analógico se logra variando el ciclo útil del pwm desde el microcontrolador. La ventaja de este método es que permite implementar el DAC usando un único pin digital del microcontrolador.

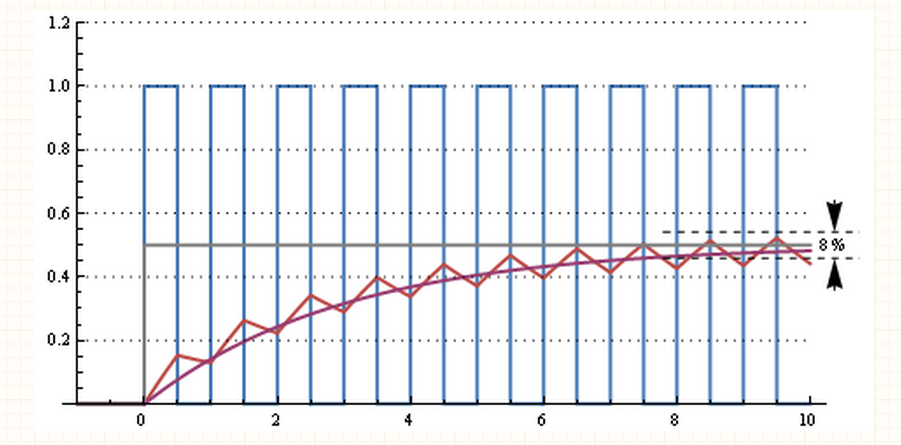


Ilustración . Filtrado de una señal PWM. (imagen tomada de https://www.cnblogs.com/shangdawei/p/4821764.html)

Comunicación serial (datos int y float)

## Comunicación Serial (UART)

Para realizar comunicación serial es necesario configurar unos registros del microcontrolador para establecer los parámetros de la comunicación. Entre los parámetros más importantes están los baudios, los cuales fijan la velocidad de la comunicación. Sin embargo, es importante tener en cuenta los fundamentos de la comunicación UART antes de hablar sobre la comunicación de los registros.

En esta sección solo se abordará la comunicación serial asíncrona (UART). En la Ilustración 34 se muestra el diagrama de conexión necesario para llevar a cabo este tipo de comunicación. Los dispositivos que van a participar deben de tener disponibles dos puertos, uno que usualmente se designa como Tx, responsable del envío de datos y otro que se designa como Rx, responsable de recibir los datos. Como se puede ver en la ilustración, la conexión se debe hacer cruzada, es decir, que el Tx de un dispositivo se debe conectar con el Rx del otro dispositivo. También se puede ver que ambos dispositivos deben compartir el nodo de tierra.

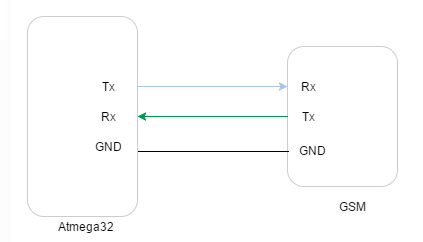


Ilustración . Diagrama de conexión para comunicación serial entre un dispositivo AVR y un Módulo GSM. (imagen tomada de http://www.exploreembedded.com/wiki/Serial\_UART\_Interface\_with\_AVR)

Se dice que la comunicación es síncrona cuando existe una señal de reloj común a ambos dispositivos, la cual les permite sincronizarse y saber cuándo tiempo debe durar la transmisión de un 1 o un 0. Es decir que a las conexiones de la Ilustración 34 habría que agregar una tercera línea con una señal de reloj si se tratara de comunicación síncrona. En este caso, al tratarse de una comunicación asíncrona, la duración de transmisión de un 0 o un 1 es incierta. Por este motivo es necesario que ambos dispositivos acuerden de antemano dicha duración. A esto se le llama baud rate o baudios, que indica número de símbolos por unidad de tiempo. Es decir que un numero alto de baudios implica una corta duración de la transmisión de un bit y por tanto una alta frecuencia de comunicación. Si ambos dispositivos no manejan los mismos baudios, como consecuencia ambos dispositivos realizaran una lectura errada de los bits que se transmitan.

En teoría se puede usar cualquier cantidad de baudios que se desee, sin embargo, debido a que los baudios se configuran mediante registros de 8 bits, para los AVR que se cubren en esta guía, no cualquier baud rate es realizable. Los valores realizables dependerán inicialmente de la frecuencia de reloj y de las características que posee el AVR que se desee emplear.

En la Ilustración 35 se muestra un ejemplo de una comunicación UART. Cuando ambos dispositivos están en reposo exhiben un nivel lógico alto en sus puertos de transmisión. Cuando se desea realizar la transmisión se indica mediante un nivel lógico bajo como se puede ver en la ilustración. Inmediatamente después del primer cero lógico se inicia la transmisión de los datos. Finalmente, el fin de la transmisión se indica nuevamente mediante un nivel lógico alto hasta que se desee enviar nuevamente información. Un motivo por el que se mantiene un nivel lógico alto para indicar estado inactivo es debido a que esto permite saber si los dispositivos que participan en la comunicación poseen alimentación eléctrica. Si fuese el caso contrario seria mas difícil distinguir un estado inactivo de un estado sin alimentación eléctrica.

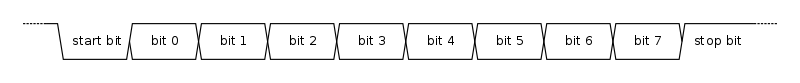


Ilustración . Ejemplo de un envio de datos mediante UART. (imagen tomada de <https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter>).

# Solución de Problemas y Depuración

Cuando se presentan errores que no corresponden problemas de sintaxis, estos no son señalados por el IDE. En muchas ocasiones esta clase de errores corresponden a errores en la lógica del diseñador y de acuerdo a la envergadura del proyecto puede ser muy difícil detectarlos en el código mediante simple inspección.

Por este motivo es muy común llevar a cabo “Debugging” que en español se traduce como depuración. En el caso de programas que se ejecutan en un computador de escritorio, la depuración consiste principalmente en un modo especial de ejecutar un programa bajo desarrollo. En este modo de ejecución, cada instrucción se ejecuta una por una según el programador lo ordena mediante la interfaz gráfica de usuario. Al mismo tiempo, se puede ver mediante con ayuda del IDE los valores que toman las variables a medida que se van ejecutando las instrucciones. De esta forma un programador puede hallar de forma más fácil los errores en su programa.

En el caso de desarrollo con microcontroladores es muy similar, pero hay unos nuevos conceptos adicionales que hay que tener en cuenta. En primer lugar, esta clase de programas no corren en la misma máquina de desarrollo (Equipo en el que se está escribiendo el programa), sino que corren en el microcontrolador, el cual es ajeno al sistema operativo en el que se está desarrollado y como consecuencia el IDE por sí mismo no puede hacer depuración. Lo anterior significa que es necesario disponer de un hardware adicional que le permita a la máquina de desarrollo controlar la ejecución de las instrucciones en el microcontrolador y además poder ver valores de variables y registros. A este hardware se le llama Debugger en inglés, y por todo lo anterior se le suele dar el nombre de on-chip Debugging cuando se trata de depuración con microcontroladores.

En algunas ocasiones los fabricantes integran la función de depuración al programador, pero también es común que se requieran productos separados para cada labor, un producto para programación y otro para depuración. Lo anterior ocurre normalmente cuando las funcionalidades de depuración son muy avanzadas y por consiguiente el hardware para implementarlas se hace complejo.

Ejemplos de programadores y Debuggers son: JTAG ICE3, PicKit4, ST-LINK V2, Cold Fire, entre otros.

Para realizar la depuración normalmente se hace uso de un protocolo de comunicación entre el microcontrolador y el Debugger. Es muy común el uso de protocolos como JTAG, el cual, a su vez, se usa también para programar el microcontrolador.

En esta sección se mostrará un ejemplo de depuración usando el microcontrolador ATmega162 y el protocolo JTAG. Este se basa en el proyecto que hace parpadear un led mediante interrupción de un temporizador. Mas abajo se muestra el código empleado, se puede encontrar en el repositorio bajo el nombre “DebuggingExample1.cpp”.

#include <avr/io.h>

#include <avr/interrupt.h> // libreria de AVR que facilita uso de interrupciones

#define *F\_CPU* 4000000UL //(1Mhz) macro para definir la frecuencia de trabajo del microcontrolador. se usa en la implementacion del comando delay.

// #define \_BV( bit ) ( 1<<(bit) ) // macro para facilitar la escritura de los comandos para leer y escribir sobre registros del microcontrolador.

#define COUNTER\_INIT 61630 // valor a partir del cual debe empezar a contar el temporizador.

int main(void)

{

DDRC |= \_BV(0); // se establece el pin PC0 como salida digital

PORTC |= \_BV(0);

sei(); // funcion que configura registros para habilitar interrupciones.

TCNT1 = COUNTER\_INIT; // se inicializa el temporizador en 616630

TIMSK |= \_BV(7); // se habilita la interrupcion de overflow en temporizador 1

TCCR1B |= \_BV(2) | \_BV(0); // se establece fuente de reloj con preescalamiento de 1024

int a = 0;

while (1)

{

// este incremento de la variable "a" no tiene ninguna funcionalidad

// solo se agrega para mostrar la observacion de variables durante el

// proceso de debugging

a++;

if(a == 10000){

a=0;

}

}

}

// rutina de interrupcion cuando se desborda el temporizador 1

ISR(TIMER1\_OVF\_vect){

TCNT1 = COUNTER\_INIT; // se recarga nuevamente el temporizador en 61630

PORTC ^= \_BV(0); // se articula el valor del bit 0 en el puerto C.

}

En esta ocasión no se entrará en los detalles del programa ya que esto se aborda en el ejemplo de interrupciones. Para iniciar el proceso de depuración lo primero es realizar la compilación del programa en modo debug como se muestra en la Ilustración 33. Se puede ver en pestaña desplegable que se encuentra dentro de la elipse roja que hay varias opciones. Dependiendo de la fase de desarrollo del proyecto se debe seleccionar debug o reléase. Cuando se compila en modo debug, el IDE agrega unas instrucciones adicionales al programa compilado para que el dispositivo debugger pueda hacer paradas entre líneas, leer variables y registros cuando el depurador lo requiera. Para entrar en modo depuración lo primero que se debe hacer es seleccionar una herramienta. Se puede ver a la derecha de la pestaña desplegable el símbolo de un martillo con la descripción “JTAG on JTAGICE3…”. Al hacer click en ese botón se puede hacer la configuración de la herramienta que se desea usar. Para hacer una simulación debe seleccionarse simulador. Una vez se ha seleccionado la herramienta con la que se desea trabajar se debe hacer click en triangulo azul con barras verticales que se encuentra a la izquierda de la pestaña desplegable de modo de compilación (elipse roja).

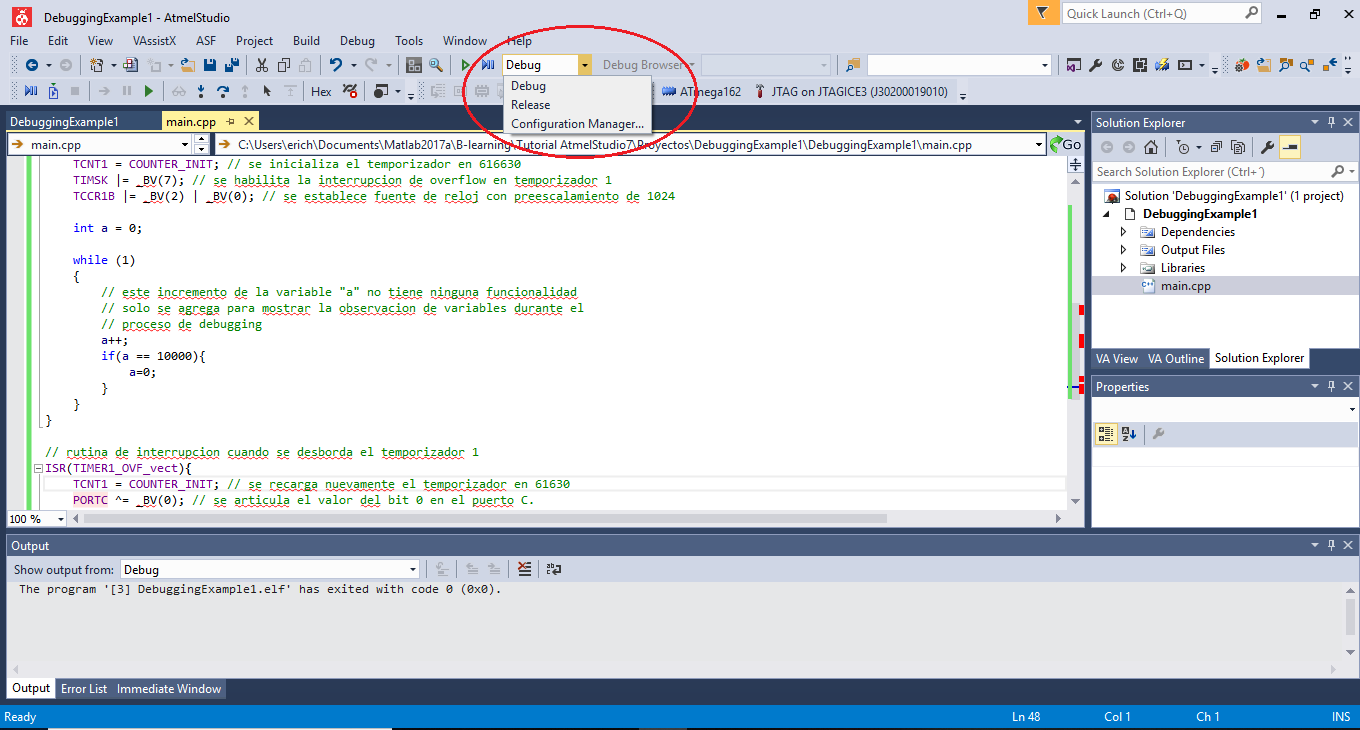


Ilustración 36. Configuración del tipo de compilación.

En la Ilustración 34 se muestra como se ve el IDE después de iniciar el modo depuración. La línea resaltada en amarillo corresponde a la línea de código que esta ejecutando el procesador. Todos los comandos que se explicaron en la sección de simulador son igualmente válidos para el caso de depuración. Se puede ver que se configuro un break point en la rutina de interrupción y en el if dentro del ciclo while. Si se da click en el botón continue y se espera el tiempo suficiente se podrá observar que el en algún momento el IDE indicará como línea de ejecución la línea con break point dentro de la rutina de interrupción. También se puede ver que es posible alterar directamente los registros del microcontrolador por medio del depurador, sin usar ninguna instrucción. Todas estas funcionalidades deben usarse de manera creativa para poder encontrar errores y fallos cuando los halla. Esto suele ser muy útil cuando se trata de programas que son muy complejos.

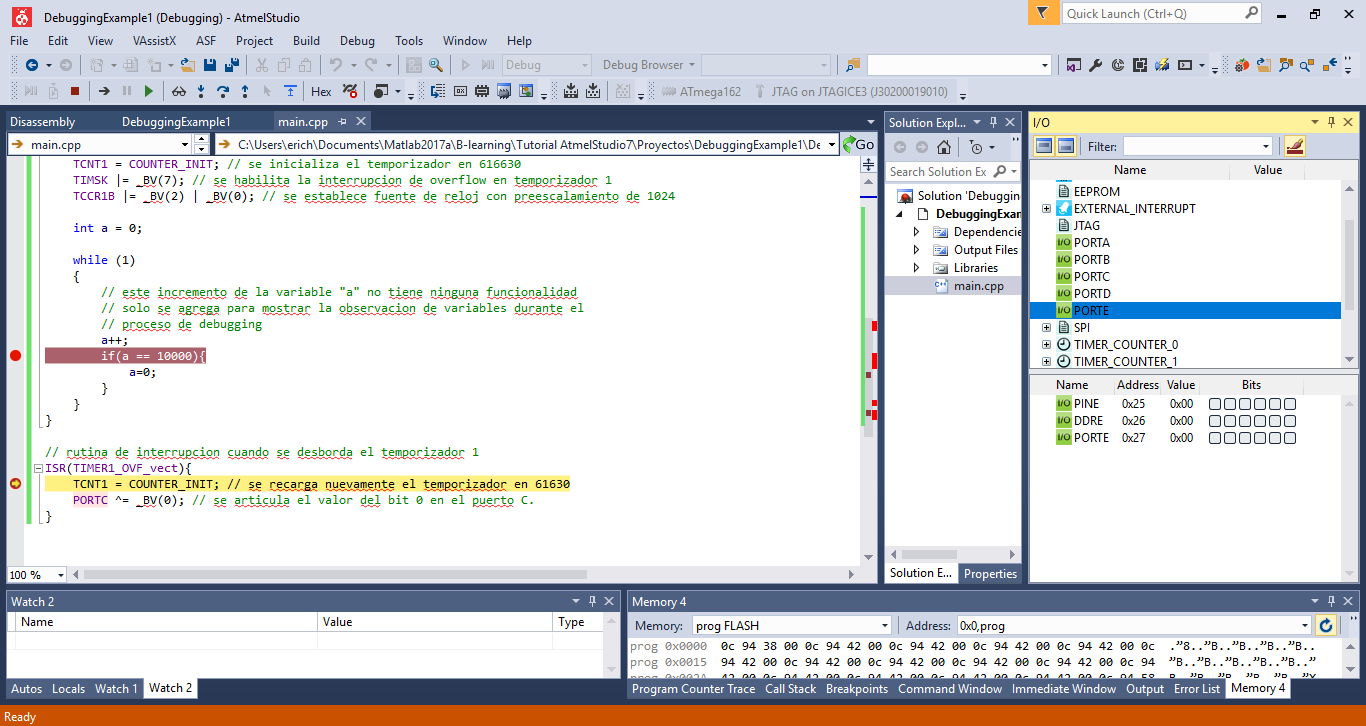


Ilustración 37. Demostración del modo depuración.

# Referencias

Datasheet jtag

[1] Datasheet attinny4313

Datasheet atmega328