eZeeNet™ Software 1.7
Nota de Aplicación

Creando, construyendo y depurando aplicaciones eZeeNet en Eclipse

c/ Doce de Octubre 38 - 28009 Madrid • Telf.: 91 504 02 01 • Fax: 91 504 00 69
e-mail: info@nextfor.com • http://www.nextfor.com

Resumen

Este documento proporciona una breve introducción al proceso de crea, construir y depurar proyectos basados en eZeeNet, utilizando el software IDE libre Eclipse, el compilador WinAVR y el hardware mkII JTAGICE en un PC con Microsoft® Windows™2000/XP.

Este tutorial explica cómo crear un pequeño proyecto que controla uno de los LEDs en la placa de desarrollo MeshBean de MeshNetics y depurarlo mediante Eclipse y el hardware mkII JTAGICE.

Convenciones utilizadas en el documento

Botones	Los nombres de los botones de diálogo son representados en Courier: `OK`, `Cancel`
Comandos de menú	Los ítems son representados en Courier y mostrados en el orden en que deben ser seleccionados: `File` -> `Open`
Atajos de teclado	Muchos atajos deben ser presionadas simultáneamente en el orden en que se muestran: F7, Ctrl-Shift-F5
Código fuente	Los fragmentos de código se muestran en texto coloreado:

```
/*********************************************
User's entry.
*********************************************/
void fw_userEntry(FW_ResetReason_t resetReason)
```

Nombres de archivos y directorios	Los nombres de archivos y directorios se notan como simples citas: ‘bin’, ‘avr-gdb.exe’

Intención del documento

Este documento está dirigido a los desarrolladores que quieran familiarizarse con la creación de aplicaciones mediante la pila MeshNetics eZeeNet ZigBee.

Documentos relacionados

[1] ZigBit™ Development Kit. User’s Guide. MeshNetics Doc. S-ZDK-451

[2] JTAGICE mkII Quick Start Guide http://www.atmel.com/dyn/resources/prod_documents/doc2562.pdf

[3] WinAVR User Manual / Ed. by Eric B. Weddington

[4] AVR Studio. User Guide. Available in HTML Help with the product.

Prerrequisitos

Asegúrese de que tiene instalado en su PC la última versión de JAVA Run-time Environment (JRE) (puede descargarlo de http://www.javasoft.com), Eclipse para desarrollos C++ (http://www.eclipse.org) y EinAVR (http://winavr.sourceforge.net).

También necesitará una placa MeshBean [1] y un dispositivo Atmel mkII JTAGICE[2].

Deberá instalar el controlador USB mkII JTAGICE disponible con el WinAVR [3].

Instalando el controlador mkII JTAGICE

PRECAUCIÓN

Si instala este controlador, no podrá utilizar JTAGICE con AVR Studio [4]. Si desea usar AVR Studio, deberá reinstalar el controlador (localizado en la carpeta 'usb' del directorio de instalación de AVR Studio, normalmente '\Archivos de Programa\Atmel\AVR tools\usb')

Las siguientes instrucciones son válidas para Windows XP

1. Conecte el mkII JTAGICE a su PC, enciéndalo y abra el Administrador de Dispositivos de Windows. Si no instaló el controlador proporcionado con AVR Studio, podrá ver un `USB Device` marca bajo la rama `Other devices` (ver Figura 1).

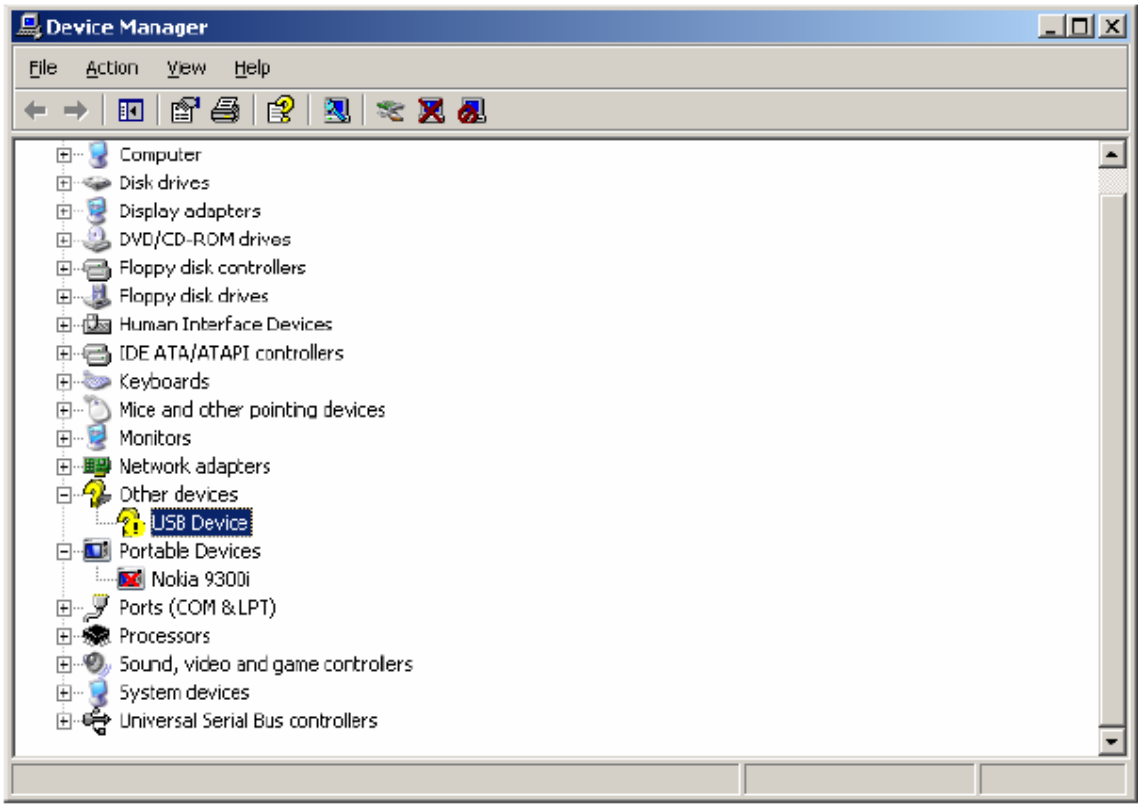

Figura 1: Pantalla del Administrador de Dispositivos

2. Si no instaló el controlador de Atmel, habrá un ítem `JTAGICE mkII` bajo la rama `Jungo`. Haga click en cualquiera de esos elementos y seleccione `Update driver` del menú. En la siguiente ventana (ver Figura 2), seleccione `Install from a list or specific location (Advanced):`

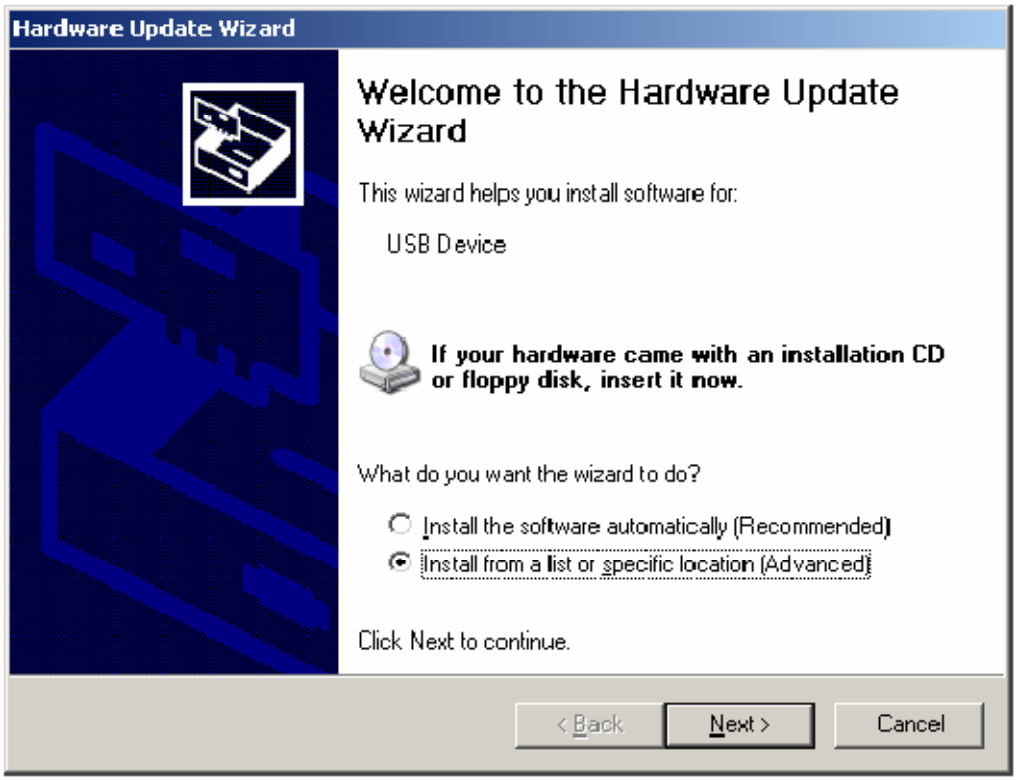

Figura 2: Asistente de actualización del hardware, pantalla de comienzo

3. Haga click en `Next >` para continuar. En la siguiente ventana (ver Figura 3), seleccione `Don't search. I will choose the driver to install` y haga click en `Next >` para continuar.

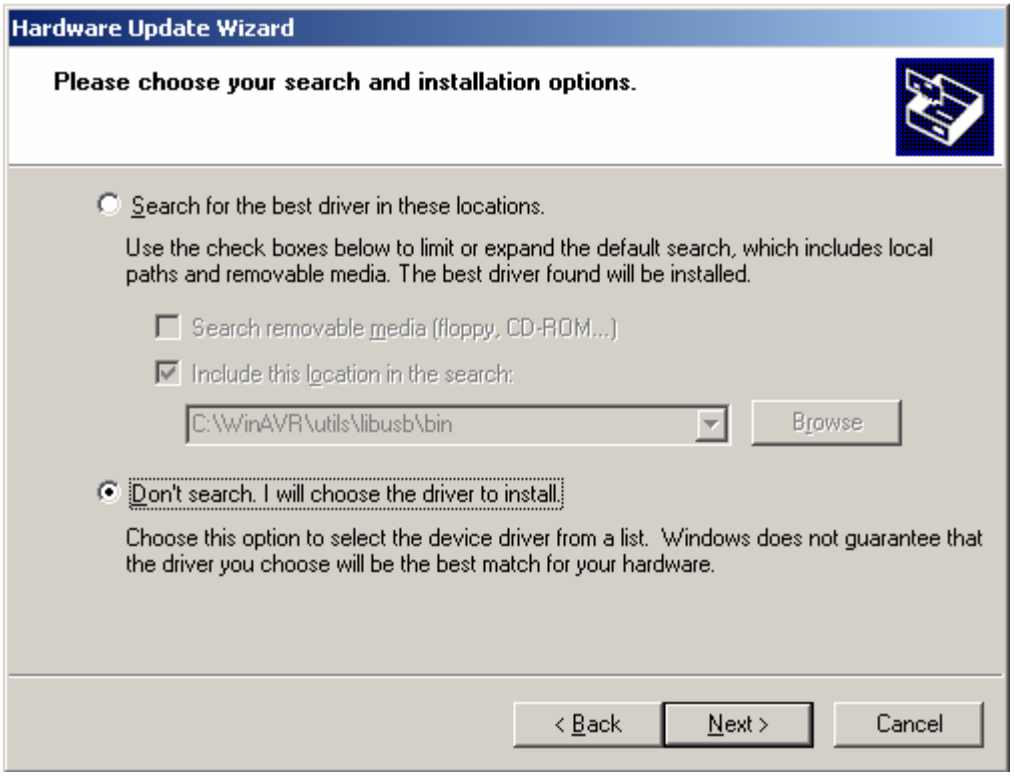

Figura 3: Asistente de actualización del hardware, opción de buscar

4. Haga click en `Next >` de nuevo y, al aparecer la nueva ventana, click en el botón `Have Disk…` (ver Figura 4):

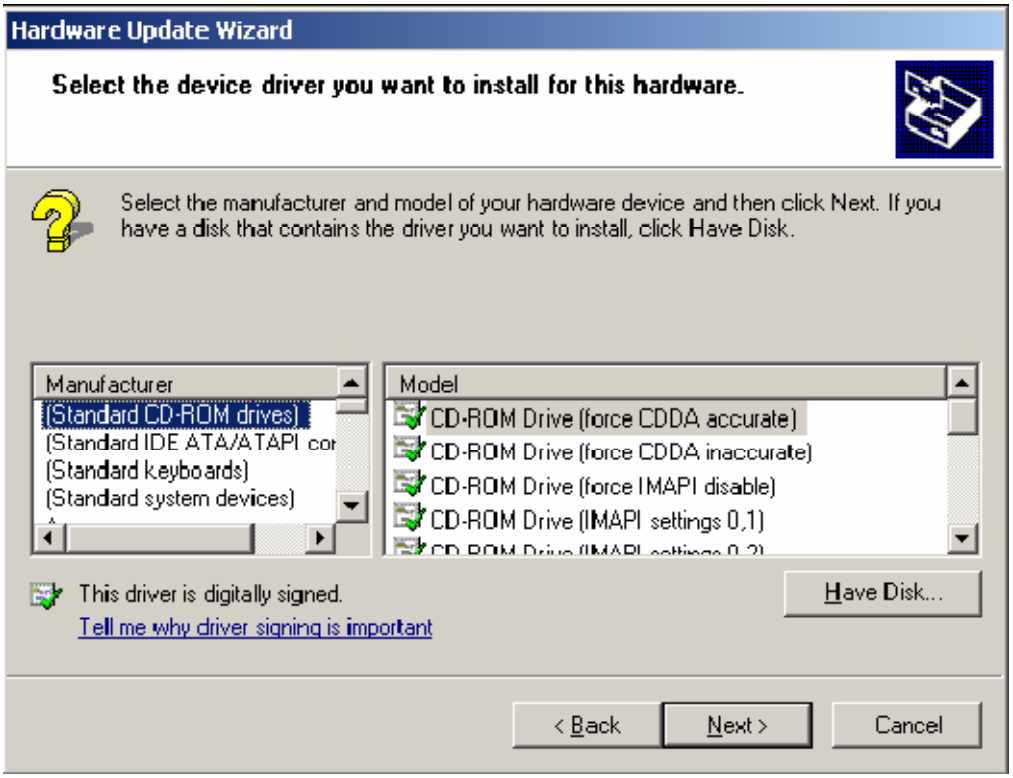

Figura 4: Asistente de actualización del hardware, pantalla de dispositivos

5. Haga click en `Browse…`, navegue a la carpeta 'Utils\Libusb\Bin' del directorio de instalación de WinAVR, seleccione'jtagice2.inf' y haga click en el botón `Open`, entonces pulse `OK`

Creando un nuevo proyecto en Eclipse

1. Inicie el IDE Eclipse ejecutando 'eclipse.exe' . Si es la primera vez que utiliza Eclipse, aparecerá una pantalla similar a la de la Figura 5.

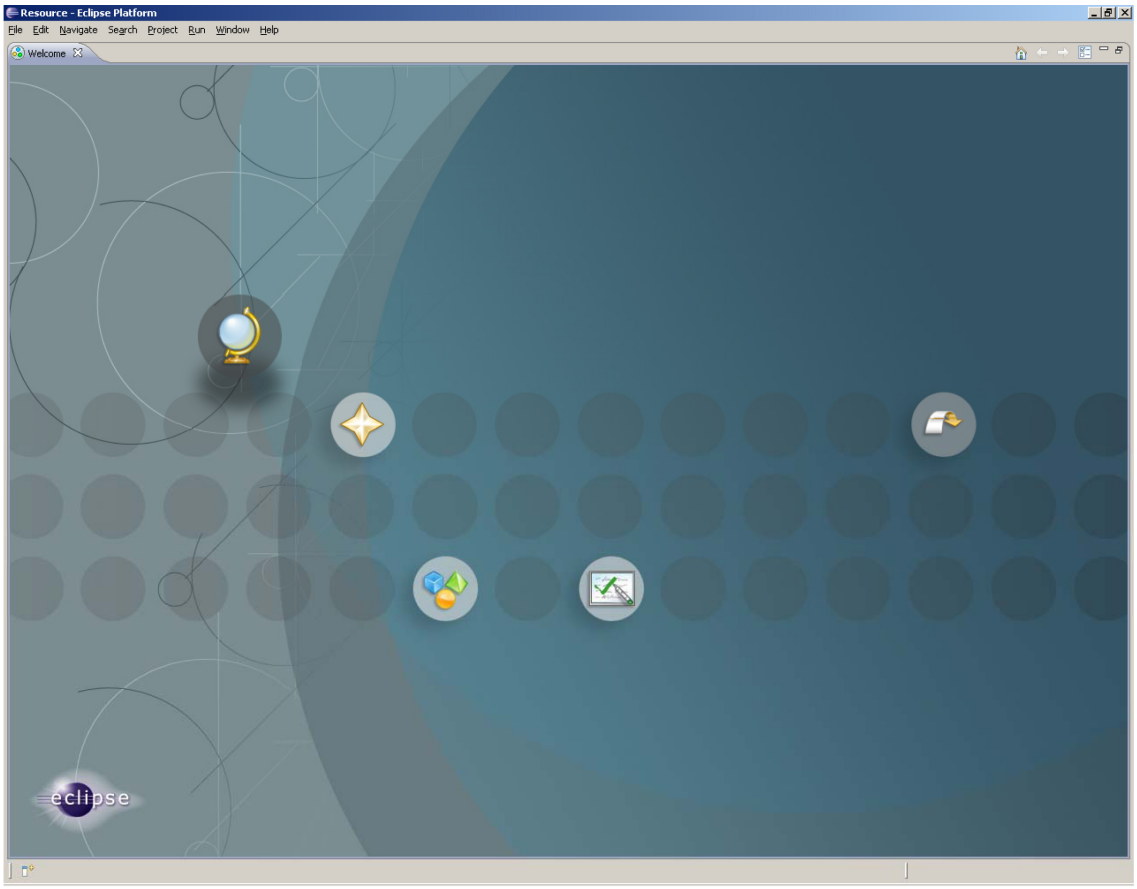

Figura 5: Pantalla de inicio Eclipse

2. Haga click en el botón más a la derecha (`Go to Workbench`) o en el botón `X` junto a `Welcome` en la parte superior de la ventana. Como muchos IDEs modernos, Eclipse tiene un concepto de *workspace* – Un espacio común para varios proyectos, es decir, puede tener un *workspace* para desarrollos embebidos y otro para aplicaciones Windows. Un proyecto puede pertenecer a diferentes *workspace*. Por defecto, Eclipse crea un nuevo *workspace* en la carpeta de usuario Windows (C:\Documents and Settings\... o como se llame en su ordenador). Este no es el mejor lugar para guardar sus proyectos, porque a las herramientas de WinAVR no les gusta que los nombres de directorios contengan espaciox. Se recomienda crear un directorio separad para los proyectos eZeeNet, por ejemplo C:\eZeeNet. Desde ahora se usará este directorio en el tutorial.
3. Seleccione `File` -> `Switch workspace` -> `Other` del menú de aplicación de Eclipse e introduzca "C:\eZeeNet" en el cuadro de texto (El directorio será creado si no existe). Eclipse se reiniciará y abrirá el nuevo *workspace*.
4. Seleccione `File` -> `New` -> `Project…` del menú, expanda la sección `C` de la lista que aparece, seleccione `C Project` y haga click en el botón `Next >` para continuar. Introduzca el nombre del proyecto ("DebugTest") en el cuadro de texto `Project name:` y haga click en `Finish`. Algunas veces se abrirá la siguiente ventana:

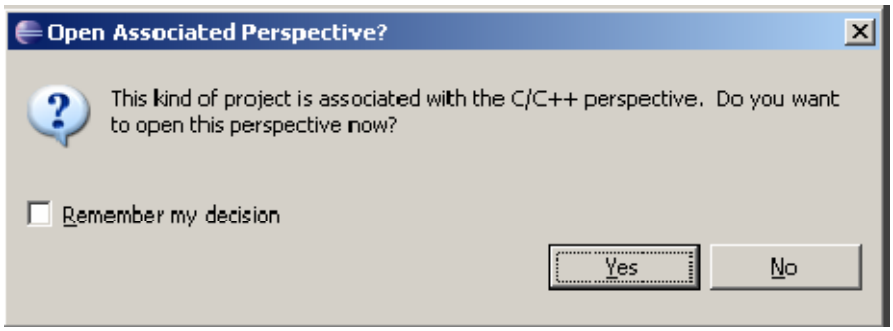

Figura 6: Cuadro de diálogo Abrir Perspectiva Asociada

5. Haga click en `Yes` para continuar

Configurando las herramientas externas

Para usar las prestaciones de programación/depuración de mkII en Eclipse, tendremos que configurar las dos herramientas externas que hay que ejecutar cada vez que se necesite programar el dispositivo con una nueva imagen y antes de comenzar la sesión de depuración. Como WinAVR es un juego de herramientas en línea de comandos, la programación del módulo con la imagen construida se realiza mediante la herramienta en línea de comando 'avarice.exe'. Tan sólo hay que ajustar un "wrapper" para esta herramienta, para evitar escribir nada cada vez que se necesita ejecutar.

1. Seleccione `Run` -> `External Tools` -> `Open External Tools Dialog…` del menú (ver Figura 5).
2. Haga click en el botón `New launch configuration` de la barra de herramientas en la esquina superior izquierda del cuadro de diálogo `External Tools` (ver Figura 7).
3. Introduzca `Burn image` en el cuadro de texto `Name:`. Haga click en el botón `Browse File System…` bajo el cuadro de texto `Location:`, diríjase al directorio de instalación de WinAVR y seleccione el archivo 'avarice.exe' (normalmente situado en el subdirectorio 'bin' de WinAVR).
4. Pulse el botón `Browse Workspace…` bajo el cuadro de texto `Working Directory:`, seleccione su proyecto actual (DebugTest) y pulse OK. Note que el cuadro de texto cambia a "${workspace_loc:/DebugTest}", que no es una ruta estándar, sino una variable de entorno (Se discutirá más adelante).
5. Escriba "2 –repf DebugTest.elf –j usb" en el cuadro de texto `Arguments.`
6. Diríjase a la pestaña `Common` (ver Figura 8) y marque la casilla `External tools` en la lista `Display in favourites.`
7. Pulse el botón `Apply` para guardar la nueva configuración.

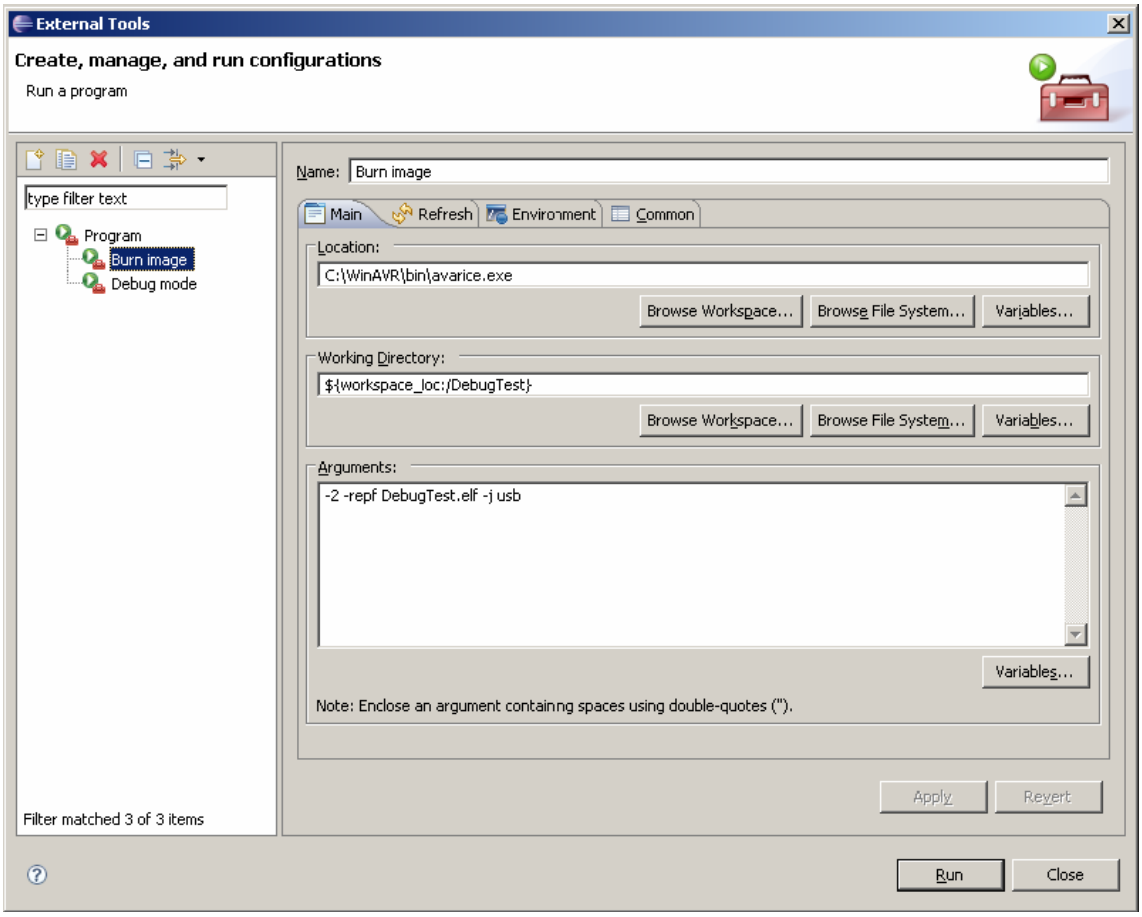

Figura 7: Cuadro de diálogo *External Tools,* Pestaña *main*

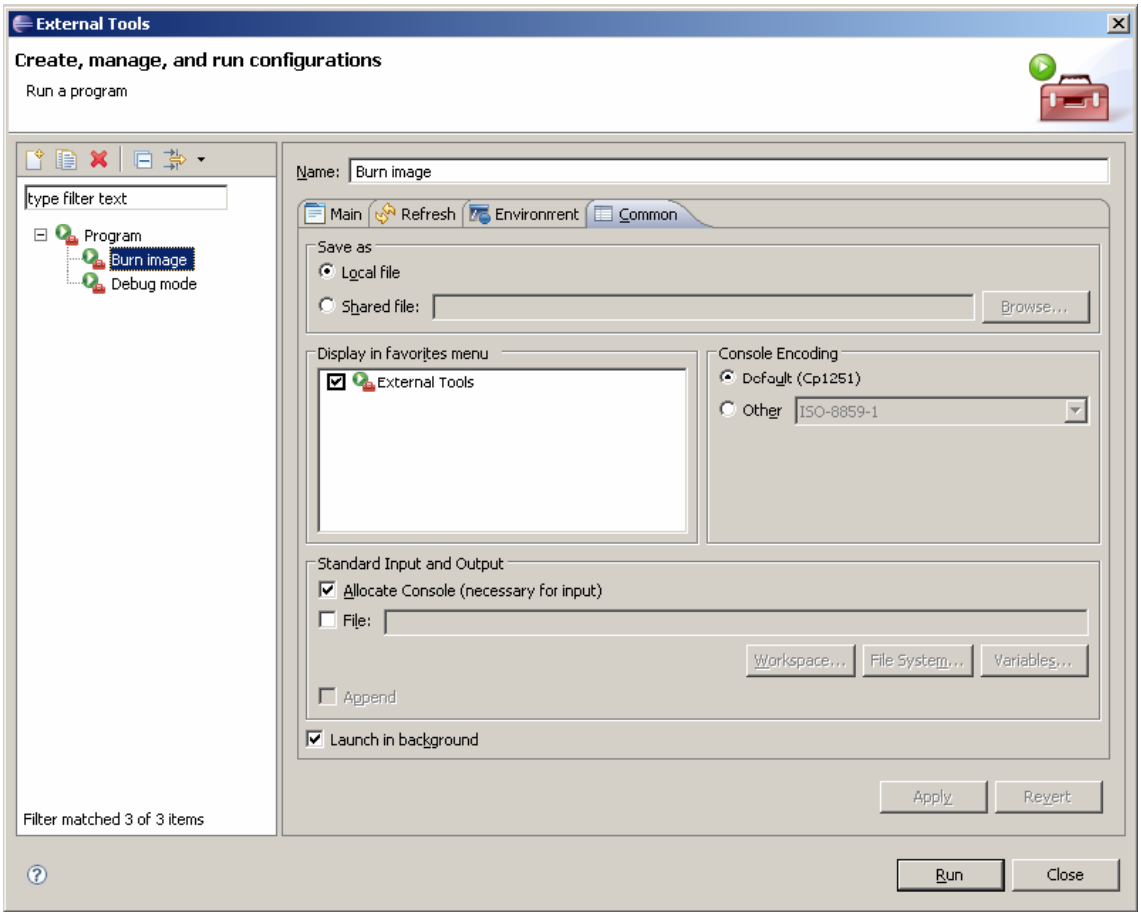

Figura 8: Cuadro de diálogo External Tools, Pestaña *Common*

El programa 'avarice.exe' también necesita ser ejecutado cada vez que quiera empezar una sesión de depuración, así que hay que configurar otra entrada "external tool", llamada `Debug mode` (ver Figura 7). Haga click en el botón `New launch configuration` de nuevo e introduzca `Debug mode` en el cuadro de texto `Name`. Una vez más, introduzca una nueva ruta para 'avarice.exe' en el cuadro de texto `Location` y la ruta del proyecto actual en el cuadro de texto `Working Directory`. En el cuadro de texto `Arguments`, introduzca "-2 -j usb :2525" (Fíjese en el espacio entre 'usb' y ':'), y haga click en `Apply`. Pulse `Close` para cerrar el cuadro de diálogo.

Creando un *makefile*

Las herramientas de WinAVR requieren de la utilización del llamad *makefile* –un archivo de texto plano que contiene todas las configuraciones (opciones del compilador, rutas, librerías, etc.) requeridas para construir la imagen objetivo. Dispone de un *makefile* de ejemplo que podrá utilizar como referencia en futuros proyectos.

Para añadir un *makefile* a su proyecto, seleccione `File` -> `New` -> `File` (ver Figura 5) o haga click con el botón derecho en la ventana del explorador de proyectos y seleccione `New` -> `File` del menú emergente

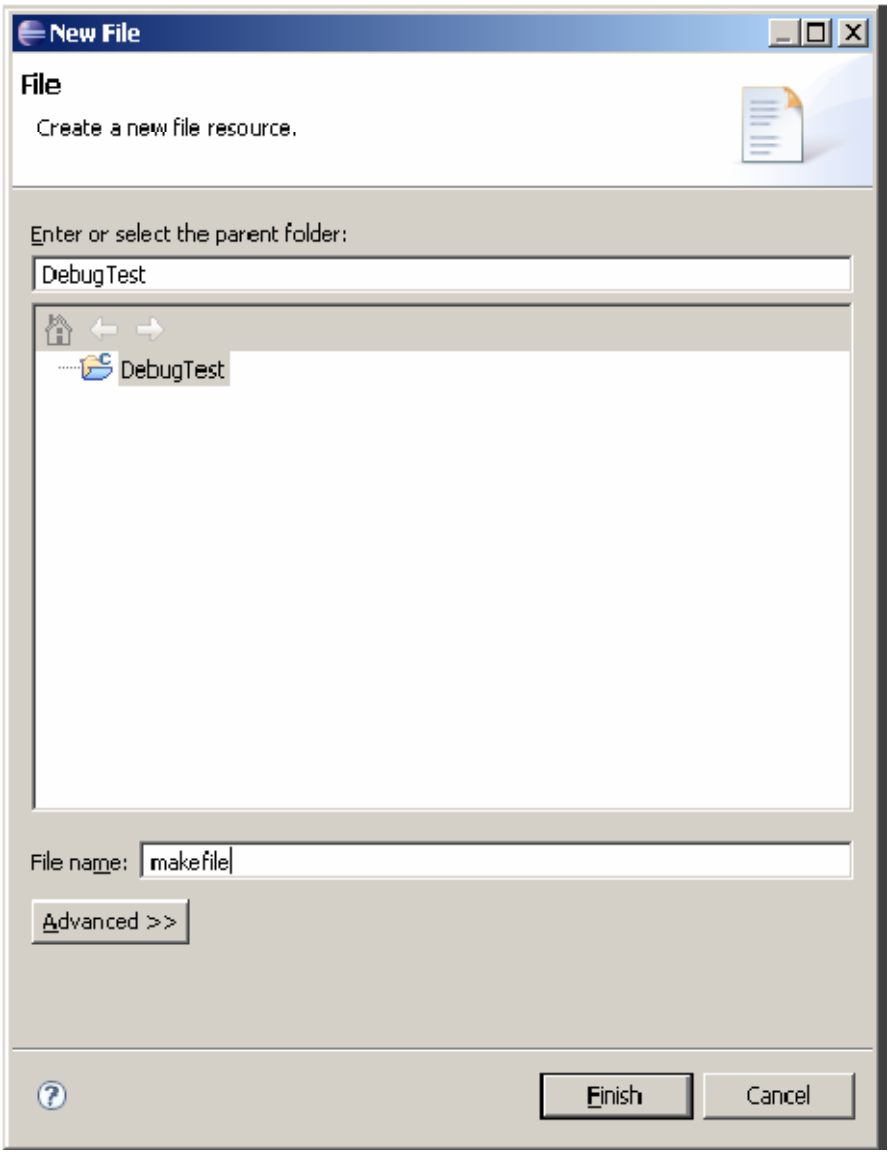

Figura 9: Cuadro de diálogo *New File*

Introduzca *makefile* en el cuadro de texto `File name` y haga click en `Finish` para cerrar el cuadro de diálogo (ver figura 9). Se creará un *makefile* nuevo y se abrirá el editor de código. Pegue el contenido del *makefile* de ejemplo en el editor

```
#############################################################
# Makefile for the project DebugTest
#############################################################
## General Flags
PROJECT = DebugTest
MCU = atmega1281
TARGET = DebugTest.elf
CC = avr-gcc.exe

## Options common to compile, link and assembly rules
COMMON = -mmcu=$(MCU)

## Compile options common for all C compilation units.
CFLAGS = $(COMMON)
CFLAGS += -Wall -gdwarf-2 -Os -std=gnu99 -fsigned-char -fpack-
struct
CFLAGS += -MD -MP -MT $(*F).o -MF dep/$(@F).d

## Assembly specific flags
ASMFLAGS = $(COMMON)
ASMFLAGS += $(CFLAGS)
ASMFLAGS += -x assembler-with-cpp -Wa,-gdwarf2

## Linker flags
LDFLAGS = $(COMMON)
LDFLAGS += -Wl,-Map=DebugTest.map

## Intel Hex file production flags
#HEX_FLASH_FLAGS = -R .eeprom

HEX_EEPROM_FLAGS = -j .eeprom
HEX_EEPROM_FLAGS += --set-section-flags=.eeprom="alloc,load"
HEX_EEPROM_FLAGS += --change-section-lma .eeprom=0

## Path to Stack, StackSupport, HAL, TOSLib.
SUPPORT_DIR = $(AEZN_HOME)

## Modules directories paths.
APP_DIR = $(PRJ_HOME)
STACK_DIR = $(SUPPORT_DIR)/Stack
STACK_SUPPORT_DIR = $(SUPPORT_DIR)/StackSupport
TOSLIB_DIR = $(SUPPORT_DIR)/TOSLib
HAL_DIR = $(SUPPORT_DIR)/HAL/HAL_R6
FRAMEWORK_DIR = $(SUPPORT_DIR)/Framework
```

```
## Include Directories.
INCLUDES = -I"$(STACK_DIR)/include" \
           -I"$(TOSLIB_DIR)/include" \
           -I"$(HAL_DIR)/base/include" \
           -I"$(HAL_DIR)/eZeeNet/include" \
           -I"$(HAL_DIR)/meshBean2/include" \
           -I"$(STACK_SUPPORT_DIR)/include" \
           -I"$(STACK_SUPPORT_DIR)/include/stack" \
           -I"$(FRAMEWORK_DIR)/include"

## Library Directories
LIBDIRS = -L"$(HAL_DIR)/lib" \
          -L"$(TOSLIB_DIR)/lib" \
          -L"$(FRAMEWORK_DIR)/lib" \
          -L"$(STACK_SUPPORT_DIR)/lib"
## Libraries
LIBS = -ltos \
       -lmeshBean2 \
       -lFW \
       -lZigBitInt \
       -lstackSupport \
       -lc

SRC = $(APP_DIR)/DebugTest.c \
      $(STACK_SUPPORT_DIR)/src/ConfigServer.c \

## Objects explicitly added by the user
LINKONLYOBJECTS = $(STACK_DIR)/lib/NWKMACLibA.o \
                  $(STACK_DIR)/lib/APLLibA.o \
                  $(HAL_DIR)/lib/wdtinit.o

## Build
all: clean $(TARGET) DebugTest.hex DebugTest.eep size

## Compile
DebugTest.o: DebugTest.c
  $(CC) $(INCLUDES) $(CFLAGS) -c $<

ConfigServer.o: $(STACK_SUPPORT_DIR)/src/ConfigServer.c
  $(CC) $(INCLUDES) $(CFLAGS) -c $<

## Build
all: clean $(TARGET) $(PROJECT).srec $(PROJECT).hex
$(PROJECT).eep size
```

```
##Link
$(TARGET):
  $(CC) $(CFLAGS) $(INCLUDES) $(SRC) $(LINKONLYOBJECTS)
-o $(TARGET) $(LIBDIRS) $(LIBS)
%.srec: $(TARGET)
  avr-objcopy -O srec $(HEX_FLASH_FLAGS) $< $@

%.hex: $(TARGET)
  avr-objcopy -O ihex $(HEX_FLASH_FLAGS) $< $@

%.eep: $(TARGET)
  avr-objcopy $(HEX_EEPROM_FLAGS) -O ihex $< $@

%.lss: $(TARGET)
  avr-objdump -h -S $< > $@

size: ${TARGET}
  @echo
  @avr-size -C --mcu=${MCU} ${TARGET}

## Clean target
.PHONY: clean
clean:
  -rm -rf $(OBJECTS) DebugTest.elf dep/* DebugTest.hex
DebugTest.eep DebugTest.srec DebugTest.map

## Other dependencies
-include $(shell mkdir dep 2>/dev/null) $(wildcard dep/*)

## End of makefile
```

Añadiendo código al proyecto

Puede añadir al proyecto un archivo fuente (.C) que contenga el código con el que se va a construir el proyecto.

Seleccione `File` -> `New` -> `Source file` del menú de aplicación (ver Figura 5) o haga click con el botón derecho en la ventana del explorador de proyectos y seleccione `New` -> `Source file` del menú emergente. Introduzca "DebugTest.c" en el cuadro de texto `Source File` y haga click `Finish` en el botón para continuar. Tendrá que especificar la extensión '.c' del archivo. A continuación se muestra el código de ejemplo utilizado en este ejemplo:

```
/*****************************************************
LED Blinking Implementation Project: C source
*****************************************************/
#include "framework.h"
#include "gpio.h"
#include "apptimer.h"

#define LED GPIO_0 // Pin connected to LED.

// Functions' declarations.
void mainLoop(); // Main loop.
void timerFired(); // Blink timer handler.
/*****************************************************
User's entry.
*****************************************************/
void fw_userEntry(FW_ResetReason_t resetReason)
{
  // Initialize the pin connected to the LED.
  gpio_setConfig(LED, GPIO_OUTPUT);
  // Open and start blink timer.
  {
    int handle;
    handle = appTimer_open(timerFired);
    appTimer_start(handle, TIMER_REPEAT_MODE, 1000);
  }
  // Start main loop.
  fw_setUserLoop(20, mainLoop);
}
/*****************************************************
Main loop.
*****************************************************/
void mainLoop()
{
// Addtional user's activities.
}
/*****************************************************
Blink timer handler.
*****************************************************/
void timerFired()
{
  static bool on = 0;
  gpio_setState(LED, on);
  on = on ? 0 : 1; // Toggle.
}
// eof DebugTest.c
```

Utilizando variables de entorno en el *makefile*

Fíjese en las siguientes líneas del texto *makefile:*

```
## Path to Stack, StackSupport, HAL, TOSLib.
SUPPORT_DIR = $(AEZN_HOME)
## Modules directories paths.
APP_DIR = $(PRJ_HOME)
```

`AEZN_HOME` y `PRJ_HOME` son variables de entorno que, respectivamente, apuntan a la carpeta API de la instalación eZeeNet y el directorio que contiene los archivos del proyecto. Usando las variables, puede evitar el tener que especificar las rutas directamente en el *makefile* y, así, la necesidad de modificar el *makefile* para cada proyecto que crea. Puede utilizar las variables en Eclipse para especificar las rutas.

1. Seleccione `Project` -> `Properties` del menú de aplicación o haga click derecho en el nombre del explorador del proyecto y seleccione `Properties` del menú emergente.

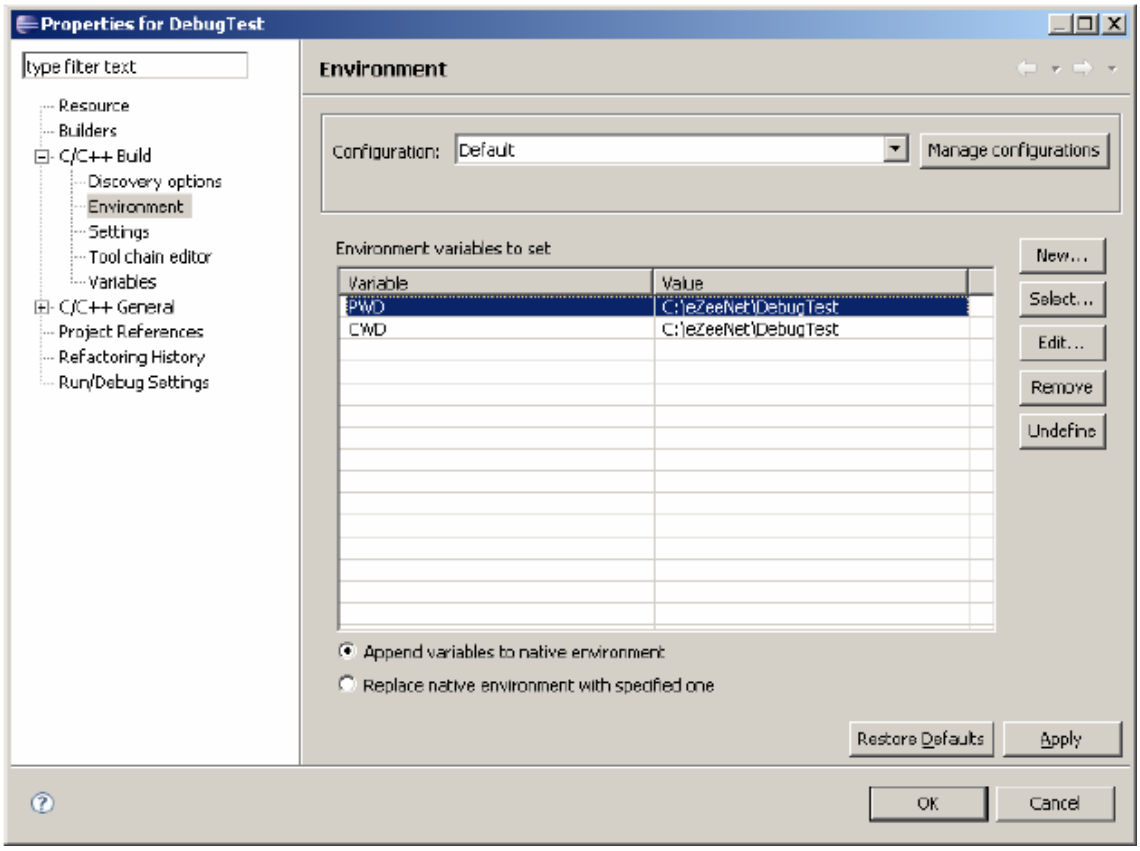

Figura 10: Cuadro de diálogo Propiedades

2. Haga click en el botón `New…` del cuadro de diálogo `Properties` (ver Figura 10). Introduzca "AEZN_HOME" en el cuadro de texto `Name` y la ruta completa del directorio API de la instalación ZDK en el cuadro de texto `Value`, como se muestra en la Figura 11.

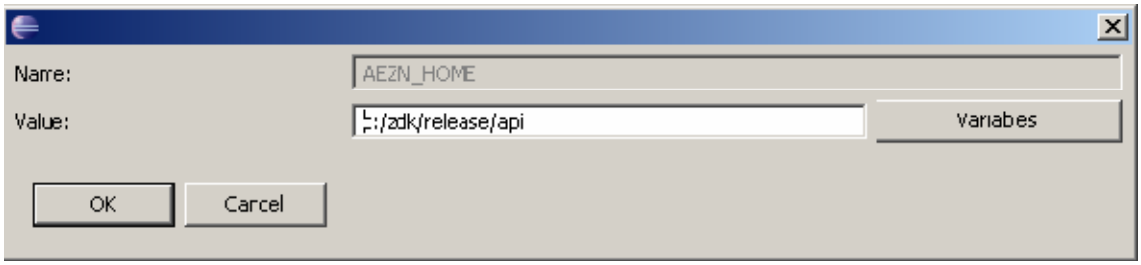

Figura 11: Cuadro de diálogo Definición de Variables

IMPORTANTE:

Tiene que usar barra oblicua hacia delante ('/') en vez de hacia atrás ('\') en los nombres de las rutas

3. Haga click en OK para cerrar la ventana. Se añadirá una nueva variable a la lista.
4. Repita los mismos pasos para crear la segunda variable, PRJ_HOME, con el valor ajustado al directorio de su proyecto actual ('c:/ezeenet/debugtest' o similar).
5. Haga click en Apply y, entonces, en OK para cerrar el cuadro de diálogo

Configurando el depurador

Seleccione Run -> Open Debug Dialog… del menú de aplicación. Pulse con el botón derecho del ratón en C/C++ Local application y seleccione New del menú emergente.

En el cuadro de diálogo que aparece, siga los siguientes pasos:

1. Introduzca Debug" en el cuadro de texto Name.
2. Click Browse en junto al cuadro de texto Project, seleccione su proyecto (Debug Test) de la lista y pulse en OK.
3. Introduzca "DebugTest.elf" en el cuadro de texto C/C++ Application.
4. Pulse Apply.
5. Seleccione la pestaña Debugger.
6. Seleccione gdbserver Debugger de la lista Debugger.
7. Desmarque la casilla.
8. Pulse Browse junto al cuadro de texto GDB Debugger y vaya al subdirectorio 'bin' del directorio de instalación de WinAVR. Seleccione el archivo 'avr-gdb.exe' y pulse Open (o introduzca la ruta completa a 'avr-gdb.exe', normalmente algo como 'C:\WinAVR\bin\avr-gdb.exe').
9. Limpie el cuadro de texto GDB command file.
10. Seleccione la pestaña Connection.
11. Cambie el tipo de conexión a TCP y el número de puerto a 2525.
12. Seleccione la pestaña Common.
13. Pulse el ítem Debug en la lista Display in favourites.
14. Pulse Apply y después Close.

Construyendo la imagen

Una vez haya configurado todo, es hora de construir la imagen que va a ser depurada. Seleccione `Project` -> `Build project` del menú de aplicación. Las herramientas WinAVR se muestran en la ventana consola en la parte inferior de la pantalla (ver Figura 12).

Programando el dispositivo con la nueva imagen construida

Una vez que la imagen se haya construido con éxito, se programa en el MeshBean a través de JTAG.

1. Conecte su dispositivo JTAG al MeshBean con el conector suministrado y al PC con el cable USB. Alimente ambos dispositivos.
2. Ahora, seleccione `Run` -> `External tools` -> `Burn image`. Si todas las conexiones y configuraciones son correctas, comenzará la programación del nodo con el indicador de progreso en la ventana consola.

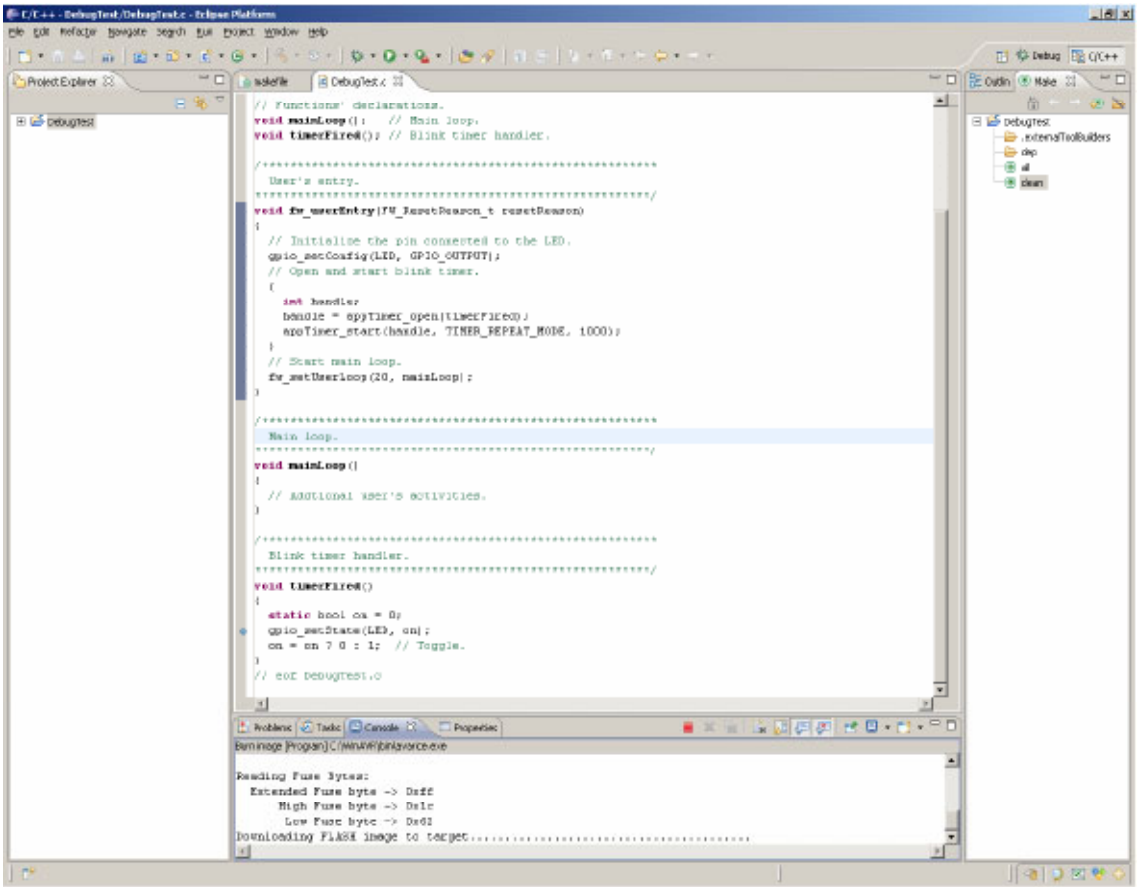

Figura 12: Pantalla de Depuración

Cuando haya finalizado, aparecerá un mensaje de error: "`USB bulk read error: usb_reap_async: error: A device attached to the system is not functioning.`".Tan sólo ignórelo.

No olvide reprogramar la MeshBean cada vez que haga cambios en el código y reconstruya la imagen.

Depurando la aplicación

Una vez que la MeshBean haya sido programada con la imagen ejecutable, se puede comenzar con la sesión de depuración.

1. Primero sitúe un *breakpoint* en una de las funciones `void timerFired()`, (ver Figura 12). Posicione el cursor en la segunda línea de esta función y seleccione `Run` -> `Toggle breakpoint` del menú o presione Ctrl-Shift-B. La línea de marcará con un círculo a su izquierda.
2. Para comenzar la depuración, seleccione primero `Run` -> `External tools` -> `Debug mode`. Esto provocará que su JTAGICE pase a modo depuración y permitirá un control adicional sobre el dispositivo objetivo. **Necesita ejecutar este comando cada vez que quiera comenzar un sesión de depuración**. La salida de la ventana consola debería mostrar esto:

```
AVaRICE version 2.6, May 15 2007 17:07:24
Defaulting JTAG bitrate to 1 MHz. Make sure that the target
frequency is at least 4 MHz or you will likely encounter
failures controlling the target.
```

3. Seleccione del menú de aplicación o pulse F11. Deberá aparecer la siguiente pantalla (ver Figura 13):

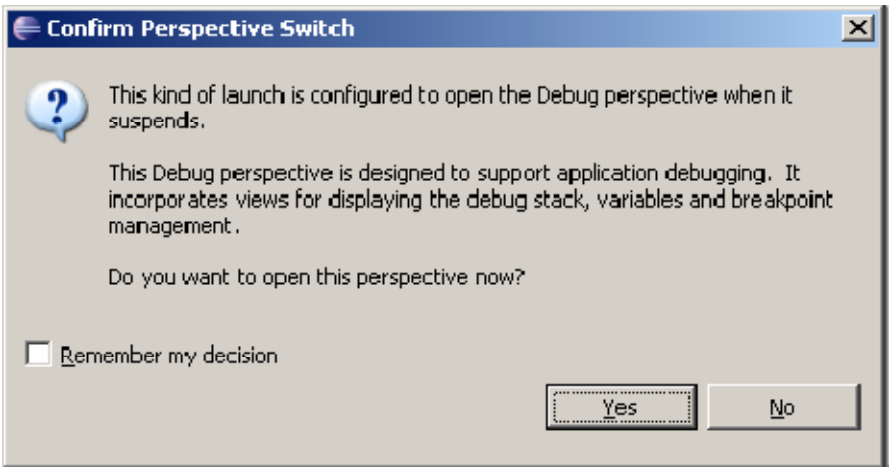

Figura 13: Confirmación de la Perspectiva

4. Pulse `Yes` para cambiar la vista (o, en terminología Eclipse, perspectiva) a modo depuración. EL diseño de la pantalla cambiará a algo como esto (ver Figura 14):

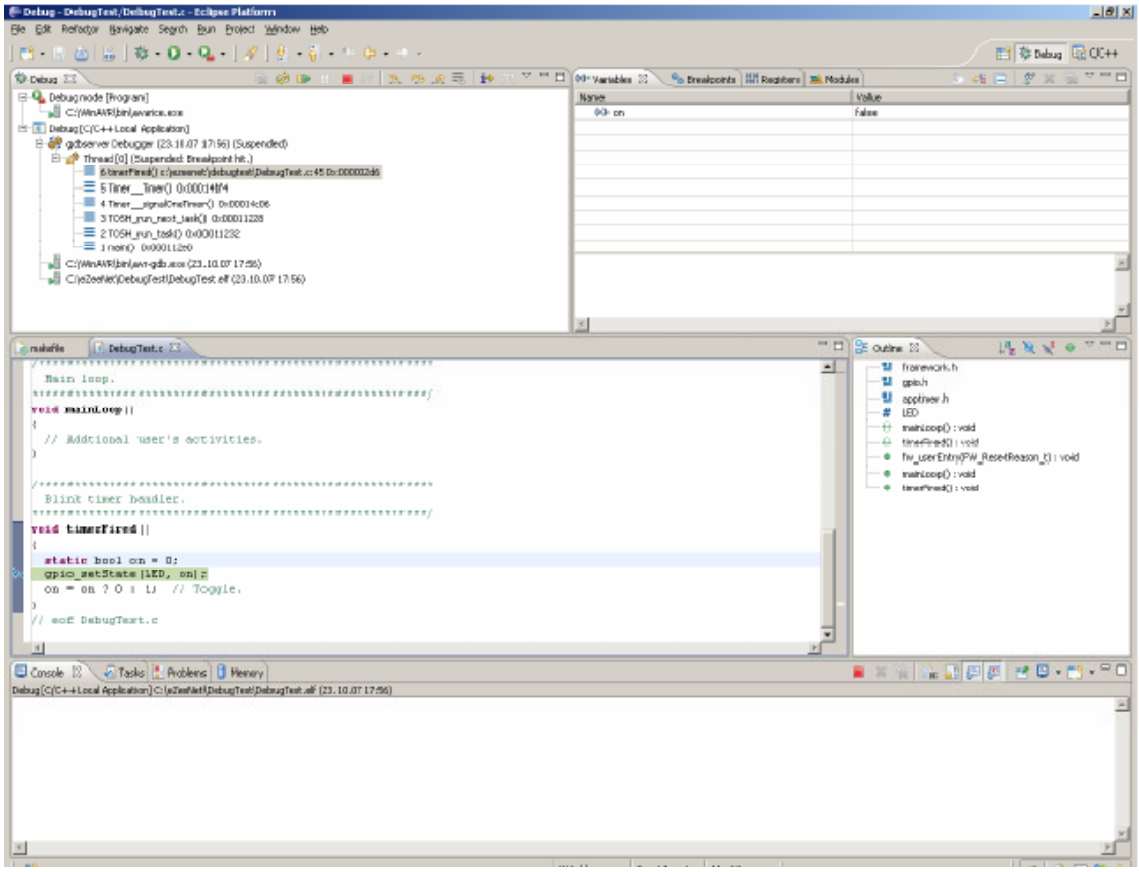

Figura 14: Pantalla de perspectiva Proyecto

Puede cambiar entre perspectivas Depuración y Proyecto utilizando los botones `Debug` y `C/C++` en la esquina superior derecha de la ventana Eclipse.

La ejecución debería parar ahora en la línea que tiene un *breakpoint*. Se pueden utilizar las muchas facilidades de depuración de Eclipse para ver el trazado de la pila, las variables de memoria, registros y otras muchas cosas. El menú `Window` -> `Show view` da accesos a cualquiera de las ventanas de depuración.

El menú `Run` contiene la mayoría de las prestaciones de depuración que necesita, como “step into”, “step over”, “run to line” y otros.

5. Presione F8 o seleccione `Run` -> `Resume` del menú para continuar con la ejecución
6. Seleccione `Run` -> `Terminate` para parar la depuración.