eZeeNet™ Software 1.7
Nota de Aplicación

Creando, construyendo y depurando aplicaciones eZeeNet en AVR Studio

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 del documento

Este documento ofrece una breve introducción al proceso de crear, construir y depurar proyectos basados en eZeeNet, utilizando el IDE de Atmel, AVR Studio [1], compilador WinAVR[2], [3] y hardware JTAGICE mkII [4] en un PC bajo Microsoft® Windows™ 2000/XP.

En este tutorial se enseña la forma de crear un pequeño proyecto que controle uno de los LEDs en la placa desarrollo MeshBean de MeshNetic y depurarlo por medio de AVR Studio y el hardware JTAGICE mkII.

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)
```

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

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

[3] Using the GNU Compiler Collection/ By Richard M. Stallman and the GCC Developer Community

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

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

Prerrequisitos

Asegúrese de que dispone de las últimas versiones de AVR Studio/Service Pack (http://atmel.com/dyn/products/tools_card.asp?family_id=607&family_name=AVR+8%2DBit+RISC+&tool_id=2725) y WinAVR (http://winavr.sourceforge.net) instaladas en su PC.

También necesitará una tarjeta MeshBean y un dispositivo Atmel JTAGICE.

Creando un nuevo proyecto en AVR Studio

1. Inicie AVR Studio. Aparecerá en pantalla el asistente de proyectos (ver Figura 1).

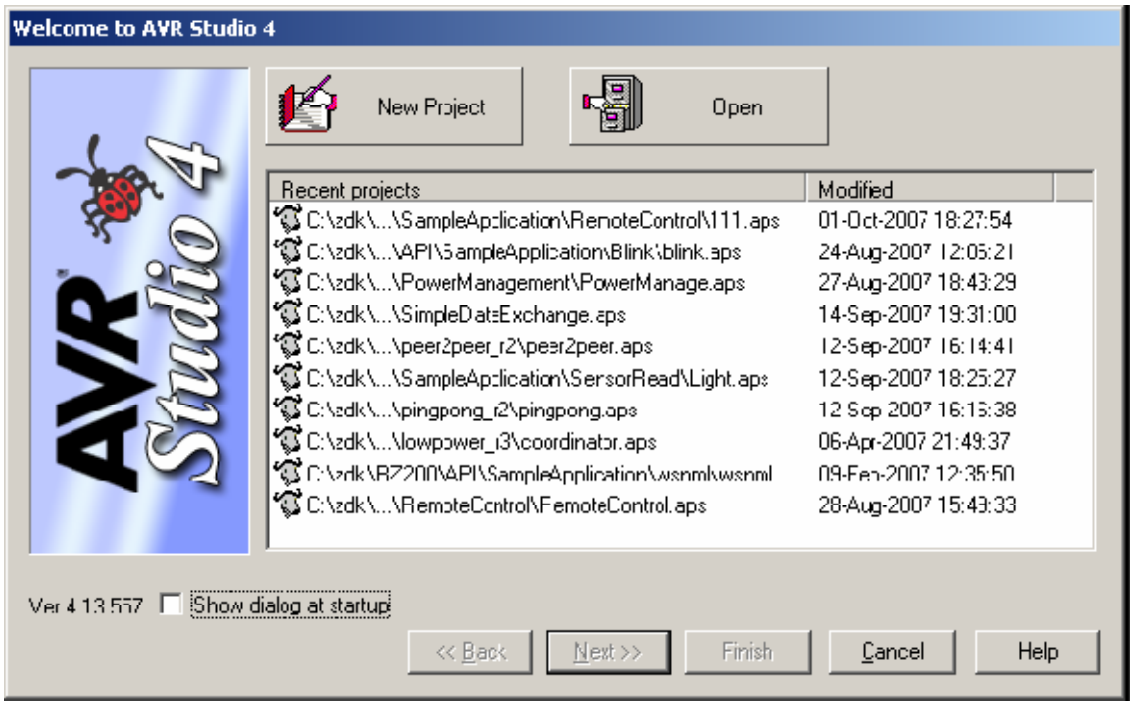

Figura 1: Ventana del Asistente de Proyectos

2. Presione el botón `New Project`. Si no aparece el asistente, seleccione `Project -> New project` del menú principal de AVR Studio.

3. En la ventana que aparece a continuación (ver Figura 2), seleccione `AVR GCC` en `Project type`, e introduzca el nombre del proyecto en `Project name`. Para crear automáticamente un archivo fuente inicial, seleccione la casilla `Create initial file` e introduzca un nombre para el archivo (sin extensión) en `Initial file`. Para crear una carpeta con el nombre del proyecto seleccione la casilla `Create folder`. Finalmente, presione el botón `...` junto a `Location` y seleccione un directorio para su proyecto. Presione `Next >>` para continuar a la siguiente pantalla.

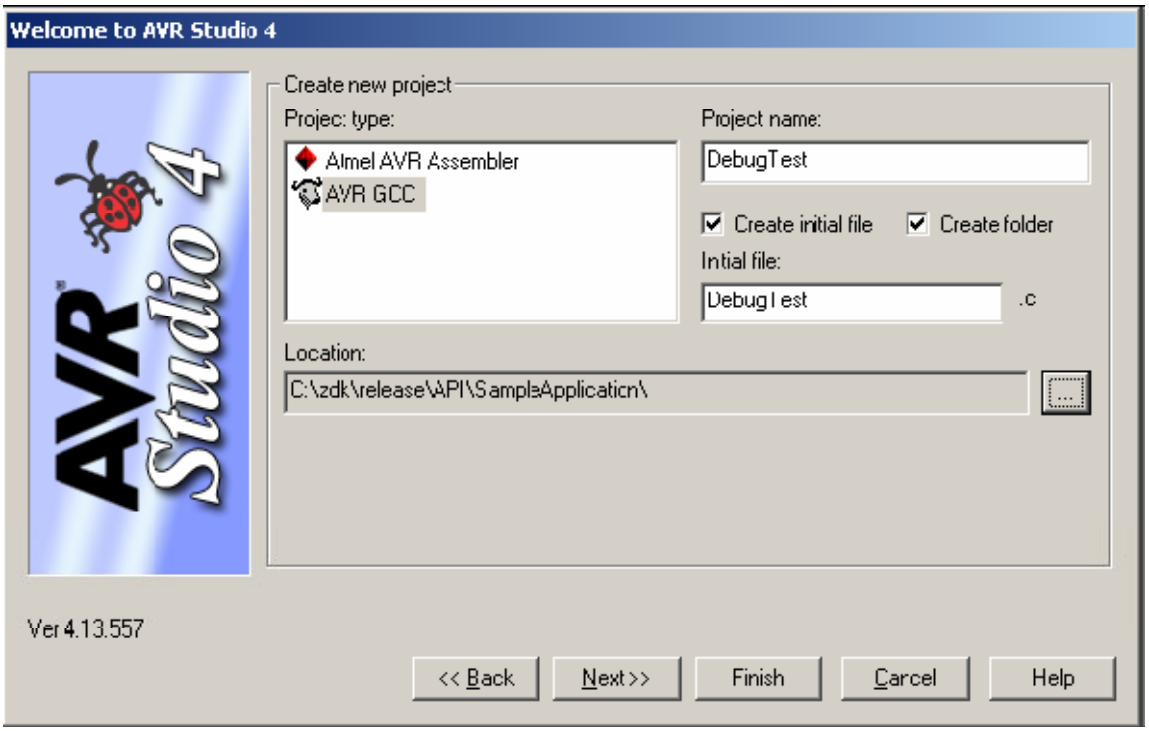

Figura 2: Creando un nuevo proyecto

4. En la siguiente pantalla que aparece (ver Figura 3), seleccione `JTAGICE mkII` de `Debug platform` y `ATmega1281` de `Device`. Pulse `Finish` para cerrar el asistente.

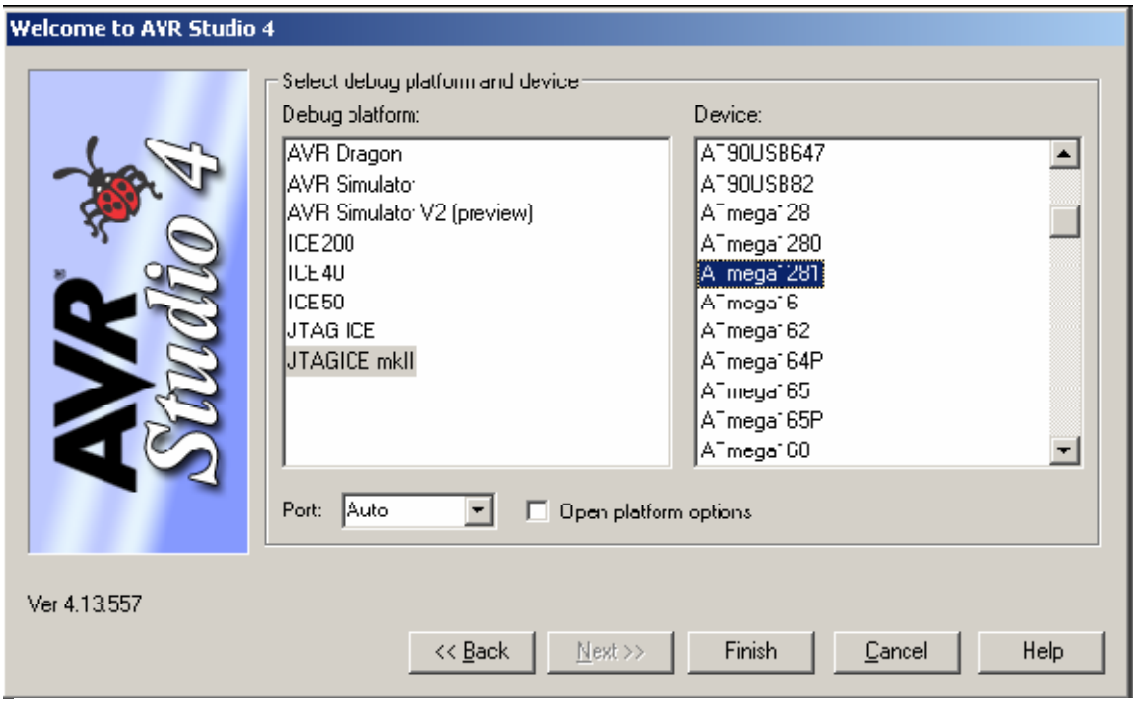

Figura 3: Seleccionando plataforma de depuración y dispositivo

Ajustando las opciones del proyecto

La colección de herramientas WinAVR requiere el uso del llamado *makefile*, un archivo de texto plano que contiene todas las opciones de proyecto, nombres de librerías/fuentes/objetivos y todo lo necesario para construir la imagen objetivo. Esto resulta un tanto más difícil que el uso de GUI, pero ofrece un mayor control sobre el proceso de construcción.

Para usar *makefile* en AVR Studio, vaya a `Project -> Configuration` en el menú y seleccione la casilla `Use external makefile`. Pulse el botón `...` junto al cuadro de texto y seleccione su *makefile* (normalmente localizado en la misma carpeta que el proyecto AVR Studio). Pulse `OK` para cerrar la ventana.

Aquí tiene un ejemplo de *makefile* que puede utilizar como referencia en proyectos futuros:

```
#############################################################
# 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 = $(COMMON)
LDFLAGS += -Wl,-Map=DebugTest.map
## Intel Hex file production flags
#HEX_FLASH_FLAGS = -R .eeprom
HEX_EEPROM_FLAGS = -j .eeprom
HEX_EEPROM_FLAGS += --set-section-flags=.eeprom="alloc,load"
HEX_EEPROM_FLAGS += --change-section-lma .eeprom=0
## Path to Stack, StackSupport, HAL, TOSLib.
SUPPORT_DIR = ../..
#SUPPORT_DIR=$(wildcard ../../../WSN)$(wildcard
../../../trunk/src/WSN)
## Modules directories paths.
APP_DIR = .
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: ../../StackSupport/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
## Other dependencies
-include $(shell mkdir dep 2>/dev/null) $(wildcard dep/*)
```

Escribiendo el código fuente

Puede escribir su código fuente en AVR Studio o mediante cualquier editor de texto apropiado. La estructura general del archivo es la misma que la que usaría en cualquier otro proyecto basado en C (cabeceras, definiciones, código). A continuación se muestra el código que se va a usar en este tutorial:

```
/*****************************************************
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
```

Una vez haya terminado de escribir su código, puede construir la imagen ejecutable seleccionando `Build -> Build` del menú o presionando F7. Pude hacer uso también del comando `Rebuild all` en proyectos largos para asegurar que se compilan todos los cambios del código. La ventana Build (en la parte inferior de la pantalla) mostrará la salida del compilador *avr-gcc* (ver Figura 4).

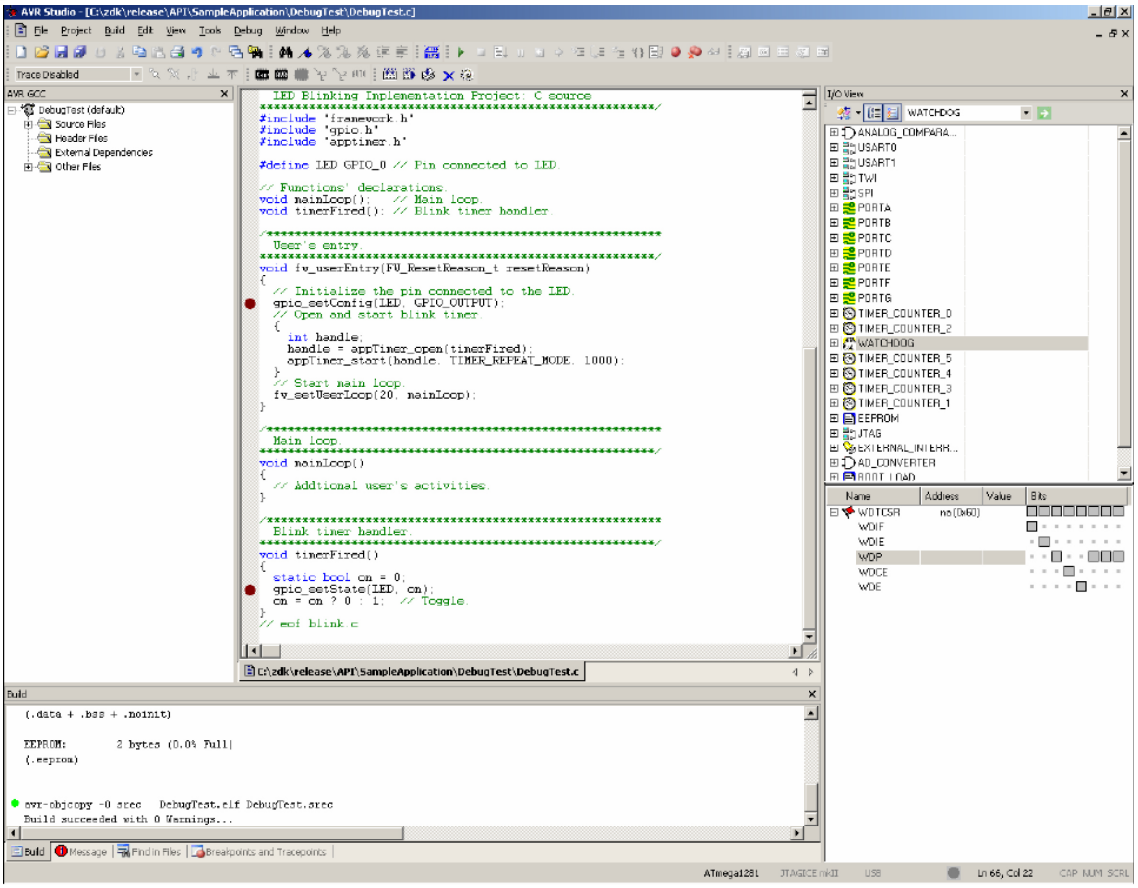

Figura 4: Construyendo la imagen

Si el código no contiene errores, se generará un archivo *DebugTest.elf* en la misma carpeta en la que se encuentra los archivos del proyecto (a menos que especifique un directorio de salida diferente en el *makefile*). Esta es la imagen ejecutable que utilizará el depurador.

Depurando la imagen

Puede ahora comenzar la sesión de depuración para la imagen que acaba de construir. Primero, conecte su MeshBean al dispositivo JTAGICE y alimente los dispositivos (pueden ser alimentados por USB).

Para probar las características de depuración de AVR Studio, sitúe un *breakpoint* (pulse F9) en cualquier línea de código de la que esté seguro que va a ser ejecutada. En este caso, se hará en la función *timerFired()*, que va a ser llamada periódicamente para el parpadeo de los LEDs de la placa MeshBeans.

Ahora, seleccione `Debug -> Start debugging` de la aplicación AVR Studio o pulse Ctrl-Alt-Shift-F5. Este comando se encuentra disponible únicamente después de ejecutar el comando `Build`. Si reinicia AVR Studio, deberá construir de nuevo la imagen. AVR Studio comenzará a programar el dispositivo con la imagen construida, indicando el progreso con la barra de progreso en la parte inferior de la ventana. Después de que la programación esté completa, saltará la siguiente ventana (ver Figura 5).

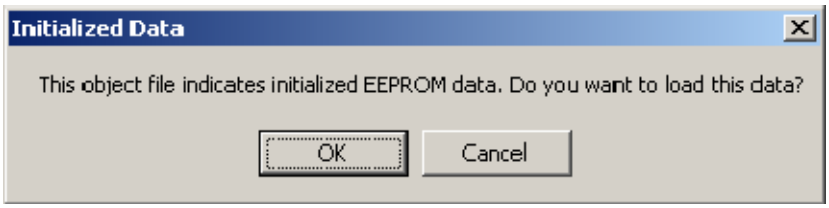

Figura 5: Confirmación de carga de datos a la EEPROM

eZeeNet utiliza una porción de la EEPROM para almacenar sus parámetros. Pulse en el botón `OK` para continuar.

En algunas ocasiones, AVR Studio interrumpirá su ejecución y mostrará la ventana de desensamblado con contenidos inválidos (ver Figura 6).

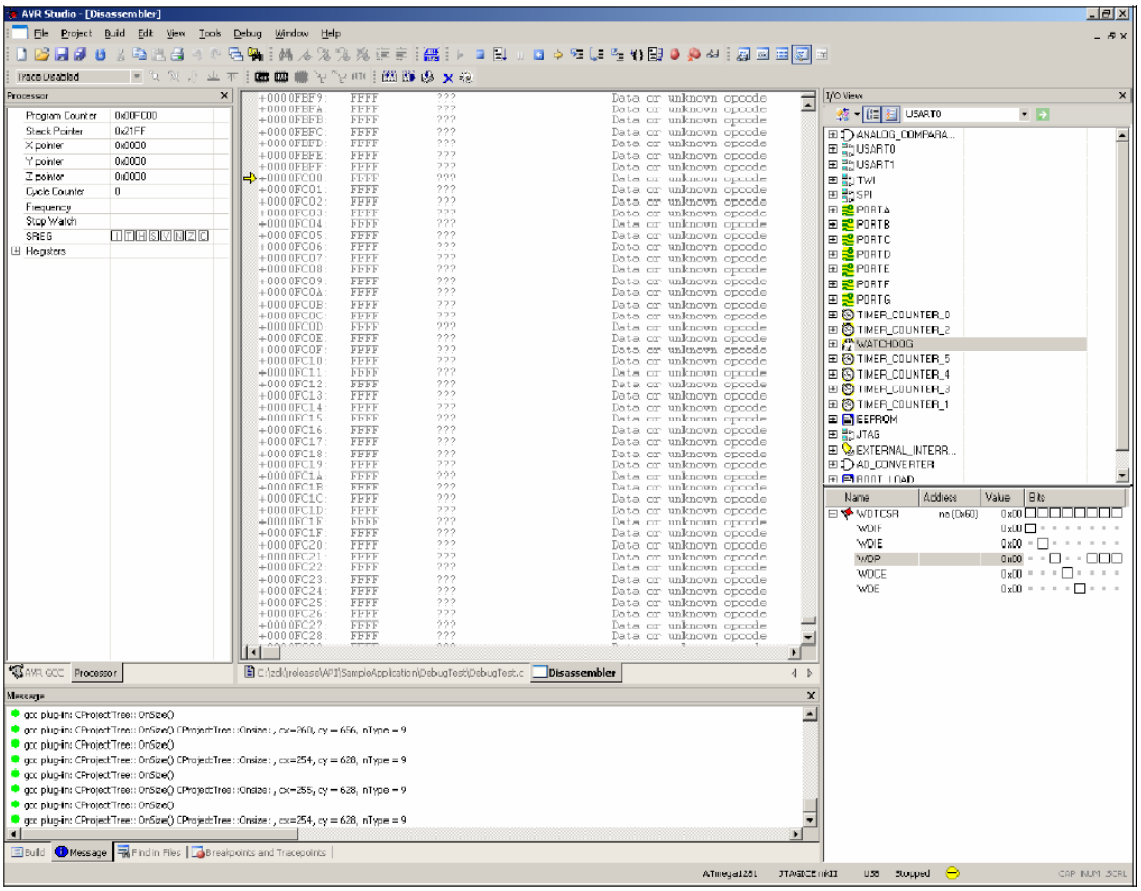

Figura 6: Ventana de desensamblado

Cierre esta ventana y presione F5 para continuar la ejecución. AVR Studio pausará la ejecución en el primer *breakpoint* del código (ver Figura 7).

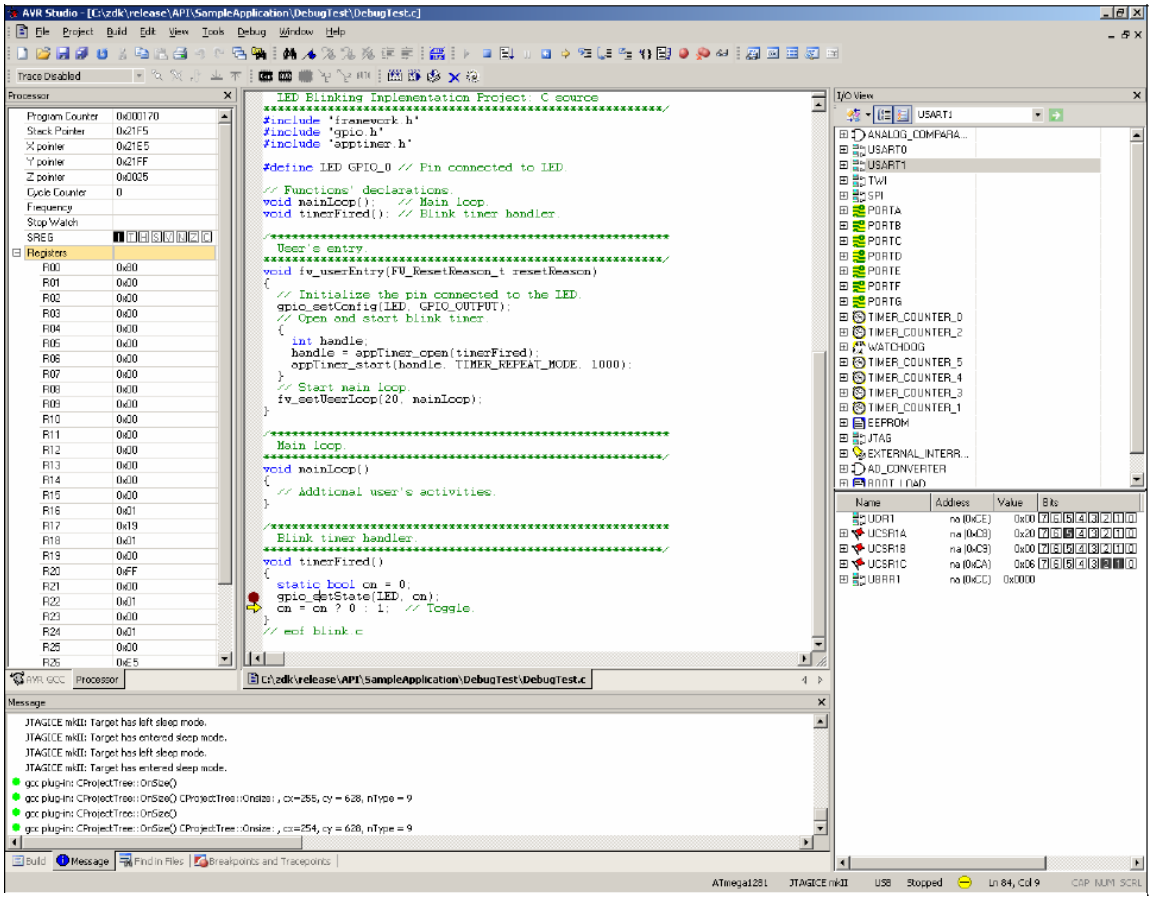

Figura 7: Depurando

En el menú `View`, puede seleccionar adicionales ventanas de depuración: procesador, E/S, desensamblador, vigilancia de variables, memoria, registros, etc. Pulsando con el botón derecho del ratón en el editor de código, podrá disponer también de unas pocas opciones de depuración, como añadir y eliminar *brakpoints* y expresiones de vigilancia.

Puede continuar con la ejecución en cualquier momento pulsando F5. Si pulsa Shift-F5, reiniciará la sesión de depuración.

Para detener el depurador, seleccione `Debug` -> `Stop debugging` del menú o pulse Ctrl-Shift-F5. Este comando está sólo disponible en modo *stop*, es decir, deberá pausar primero la ejecución y después detenerlo completamente.