Using Open Source Tools for AT91SAM7S Cross Development

Revision C

Author:

James P. Lynch
Grand Island, New York, USA

May 15, 2007

TABLE OF CONTENTS

Introduction 4
ARM Software Cross Development System 4
Target Hardware 12
Open Source Tools Required 13
Check for JAVA Support 13
Downloading YAGARTO 18
Downloading the Segger J-Link GDB Server 24
Downloading the Atmel SAM-BA Boot Assistant 28
Install All Tools 29
Install All Tools 30
  Install OpenOCD 30
  Install YAGARTO Tool Chain 32
  Install Eclipse IDE 35
  Install YAGARTO Tools 37
  Install the Segger J-Link GDB Server 39
  Install the Wiggler Parallel Port Driver 42
  Install the Amontec JTAGkey USB Drivers 45
  Install the Olimex ARM-USB-OCD USB Drivers 50
  Install the Atmel SAM-BA Flash Programming Utility 54
Download the Tutorial Sample Projects 56
Move the OpenOCD Configuration Files 60
Running Eclipse for the First Time 61
Set Up Eclipse External Tools 64
  Set Up OpenOCD as an Eclipse External Tool (wiggler) 64
  Set Up OpenOCD as an Eclipse External Tool (ARM-USB-OCD) 66
  Set Up OpenOCD as an Eclipse External Tool (JTAGkey) 68
  Set Up J-Link GDB Server as an Eclipse External Tool (SAM-ICE) 70
  Set Up SAM-BA as an Eclipse External Tool 72
  Adding Your JTAG Tools into the “Favorites” List 74
Create an Eclipse Project 76
Using the Eclipse Editor 83
  Creating a New Source File 83
  Undo / Redo 83
  Cut, Copy and Paste Operations 84
  Saving Your Code 84
  Brace Checking 85
  Searching 85
Discussion of the Source Files – FLASH Version 89
  AT91SAM7S256.H 89
  BLINKER.C 91
  CRT.S 92
  ISRSUPPORT.C 96
  LOWLEVELINIT.C 97
  MAIN.C 99
  TIMERISR.C 101
  TIMERSETUP.C 102
  DEMO_AT91SAM7_BLINK_FLASH.CMD 105
  MAKEFILE 108
  OpenOCD Programming Script File 110
Adjusting the Optimization Level 111
Including Libraries 111
  Adding Libraries to the Link 111
  Where are the Libraries 112
  Display the Modules in a Library 113
  The Bad News about Libraries 114
Building the FLASH Application 115
Using OpenOCD to Program the FLASH memory 116
  OpenOCD Configuration File for Wiggler (FLASH programming version) 116

OpenOCD Configuration File for JTAGKey (FLASH programming version)..117
OpenOCD Configuration File for ARMUSBOCD (FLASH programming version)................................117
Using SAM-ICE and SAM-BA to Program the FLASH memory 124
Debugging the FLASH Application 129
Create a Debug Launch Configuration...129
Add the Debug Launch Configuration to the List of Favorites...137
Open the Eclipse Debug Prespective...138
Starting OpenOCD 140
Starting J-Link GDB Server 142
Start the Eclipse Debugger...143
Components of the DEBUG Perspective 146
Debug Control 147
Run and Stop with the Right-Click Menu...148
Setting a Breakpoint 149
Single Stepping 153
Inspecting and Modifying Variables...156
Watch Expressions...159
Assembly Language Debugging 160
Inspecting Registers..161
Inspecting Memory 163
Create an Eclipse Project to Run in RAM 167
DEMO_AT91SAM7_BLINK_RAM.CMD...169
MAKEFILE.MAK 171
Build the RAM Project 172
Debugging the RAM Application 173
Create an Embedded Debug Launch Configuration for RAM 173
Set up the hardware 180
Open the Eclipse “Debug” Perspective 181
Start OpenOCD 182
Start J-Link GDB Server 183
Start the Eclipse Debugger...184
Setting Software Breakpoints 186
Compiling from the Debug Perspective...188
Conclusions..191
About the Author 191
Appendix 1. Olimex AT91SAM7- P64 Board 192
Appendix 2. SOFTWARE COMPONENTS...198

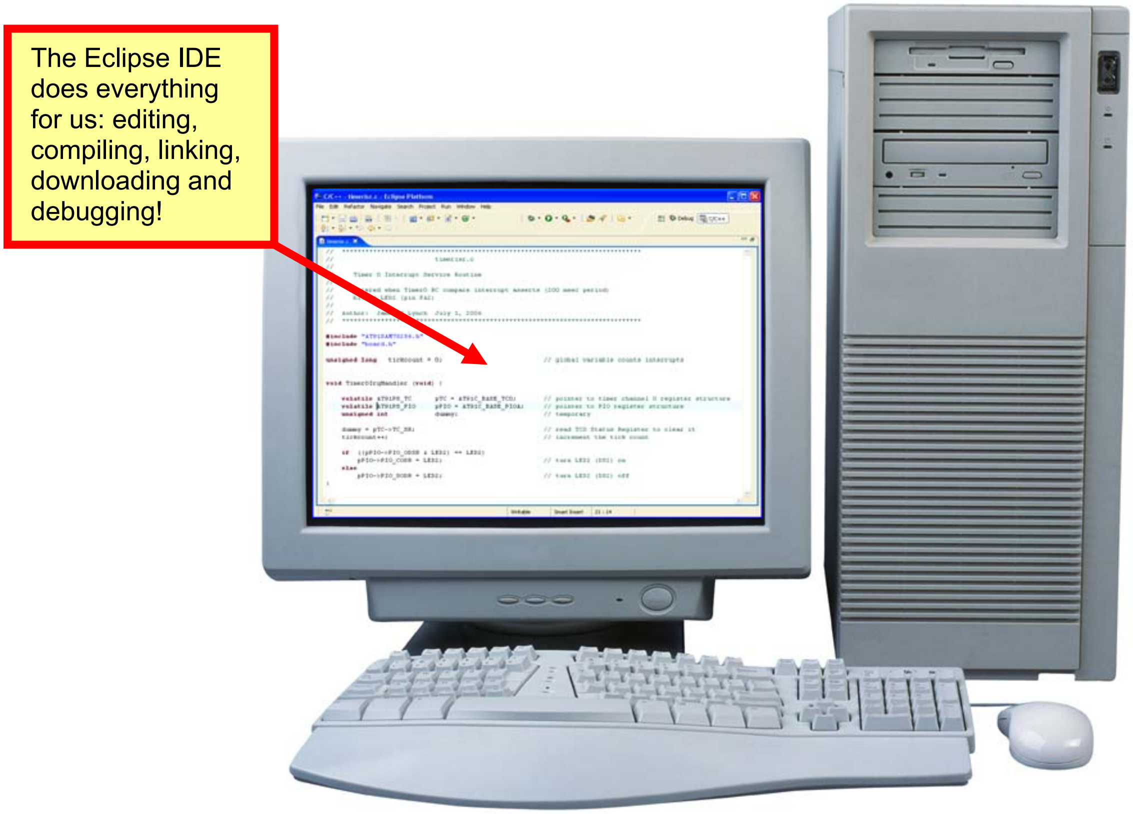
Introduction

For those on a limited budget, use of open source tools to develop embedded software for the Atmel AT91SAM7S family of microcontrollers may be a very attractive approach. Professional software development packages from Keil, IAR, Rowley Associates, etc are convenient, easy to install, well-supported and fairly efficient. The problem is their price ($900 US and up) which is a roadblock for the student, hobbyist, or engineer with limited funding.

Using free open source tools currently available on the web, a very acceptable cross development package can be assembled in an hour's work. It does require a high-speed internet connection and a bit of patience.

ARM Software Cross Development System

While there are a few diehards out there who still write their C programs with Windows Notepad and use a command prompt window to enter the GNU compile and link commands by hand, this is utter foolishness when complete Integrated Development Environments such as Eclipse are available. Eclipse allows entering and modifying your C programs using a modern software editor. It also provides single-click build and debug operations.

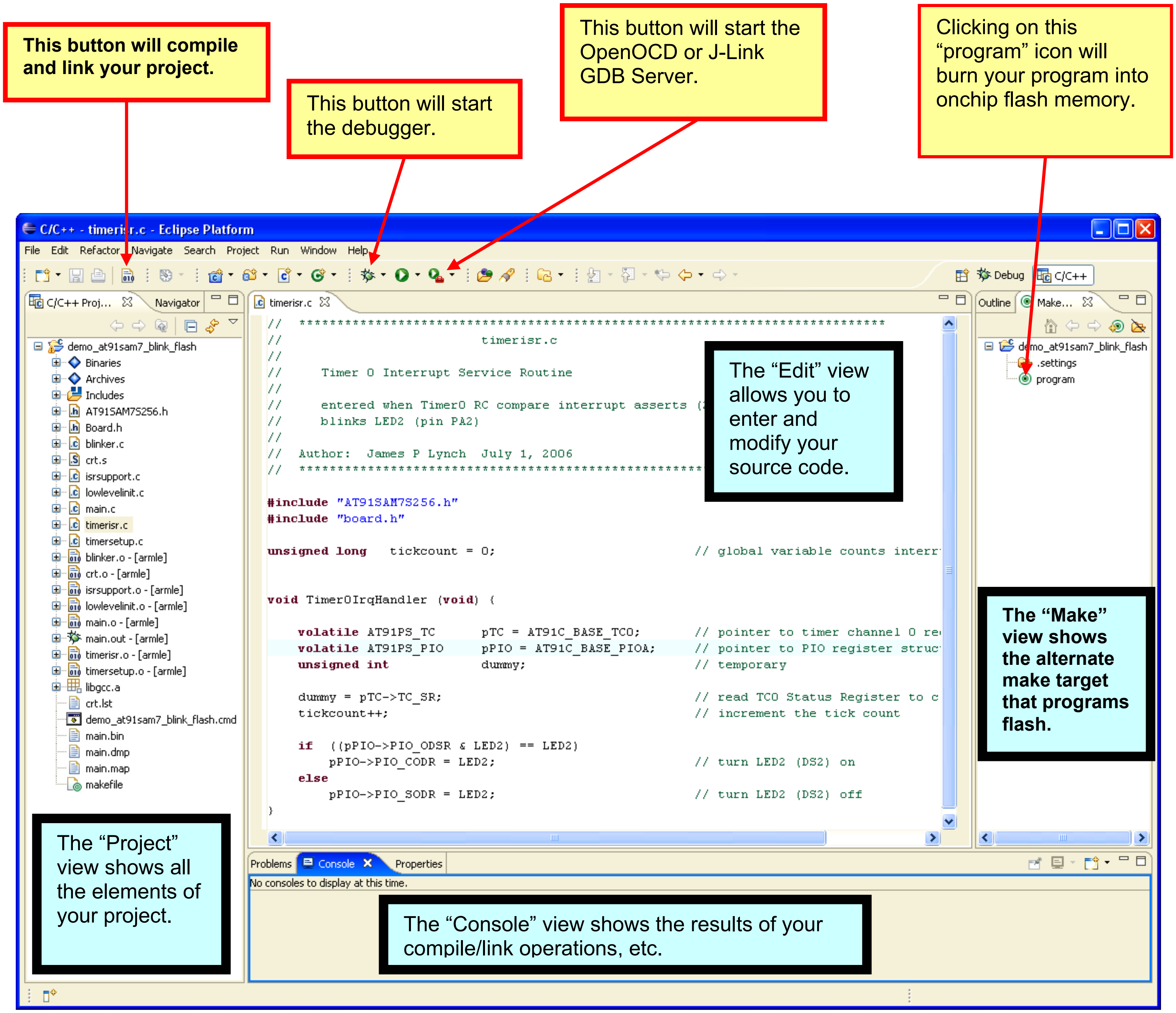


The Eclipse IDE does everything for us: editing, compiling, linking, downloading and debugging!

C/C++ - timerisr.c - Eclipse Platform
File Edit Refactor Navigate Search Project Run Window Help
Debug
C/C++
timerisr.c
Writable
Smart Insert
21 : 14

```
//  *************************************************************************
//                              timerisr.c
//
//      Timer 0 Interrupt Service Routine
//
//      entered when Timer0 RC compare interrupt asserts (200 msec period)
//      blinks LED1 (pin PA1)
//
//  Author:  James P Lynch  July 1, 2006
//  *************************************************************************

#include "AT91SAM7S256.h"
#include "board.h"

unsigned long  tickcount = 0;                      // global variable counts interrupts

void Timer0IrqHandler (void) {

    volatile AT91PS_TC     pTC = AT91C_BASE_TC0;     // pointer to timer channel 0 register structure
    volatile AT91PS_PIO    pPIO = AT91C_BASE_PIOA;   // pointer to PIO register structure
    unsigned int           dummy;                    // temporary

    dummy = pTC->TC_SR;                              // read TC0 Status Register to clear it
    tickcount++;                                     // increment the tick count

    if  ((pPIO->PIO_ODSR & LED1) == LED1)
        pPIO->PIO_CODR = LED1;                       // turn LED1 (DS1) on
    else
        pPIO->PIO_SODR = LED1;                       // turn LED1 (DS1) off
}
```

In the Eclipse screen image below, the C source file "**timerisr.c**" is being edited. There are toolbar buttons to compile and link your project, program it into onchip flash memory and start the integrated debugger. This tutorial is devoted to how you can set all this up.



This button will compile and link your project.

This button will start the debugger.

This button will start the OpenOCD or J-Link GDB Server.

Clicking on this "program" icon will burn your program into onchip flash memory.

The "Edit" view allows you to enter and modify your source code.

The "Make" view shows the alternate make target that programs flash.

The "Project" view shows all the elements of your project.

The "Console" view shows the results of your compile/link operations, etc.

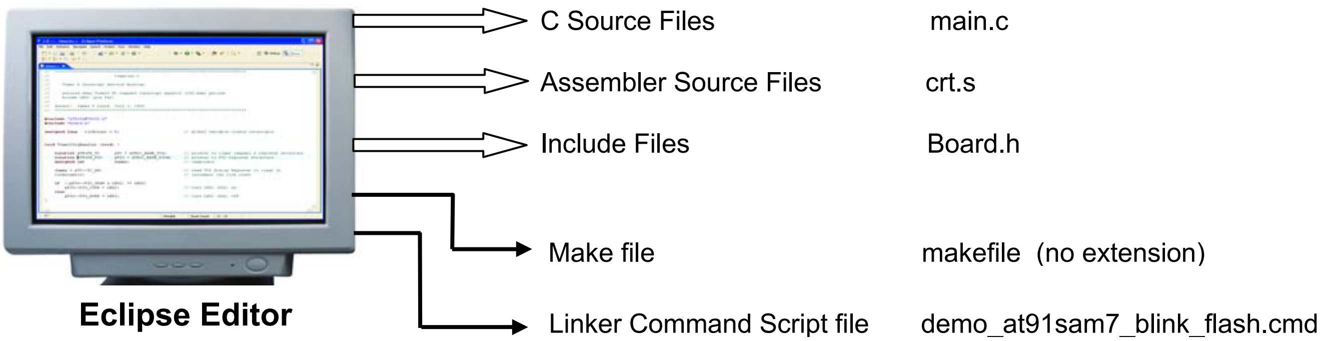
```
//  **************************************************************************
//                            timerisr.c
//
//    Timer 0 Interrupt Service Routine
//
//    entered when Timer0 RC compare interrupt asserts (
//    blinks LED2 (pin PA2)
//
//  Author:  James P Lynch  July 1, 2006
//  *****************************************************

#include "AT91SAM7S256.h"
#include "board.h"

unsigned long   tickcount = 0;                       // global variable counts interr


void Timer0IrqHandler (void) {

    volatile AT91PS_TC      pTC = AT91C_BASE_TC0;        // pointer to timer channel 0 re
    volatile AT91PS_PIO     pPIO = AT91C_BASE_PIOA;      // pointer to PIO register struc
    unsigned int            dummy;                       // temporary

    dummy = pTC->TC_SR;                                  // read TC0 Status Register to c
    tickcount++;                                         // increment the tick count

    if  ((pPIO->PIO_ODSR & LED2) == LED2)
        pPIO->PIO_CODR = LED2;                           // turn LED2 (DS2) on
    else
        pPIO->PIO_SODR = LED2;                           // turn LED2 (DS2) off
}
```

No consoles to display at this time.

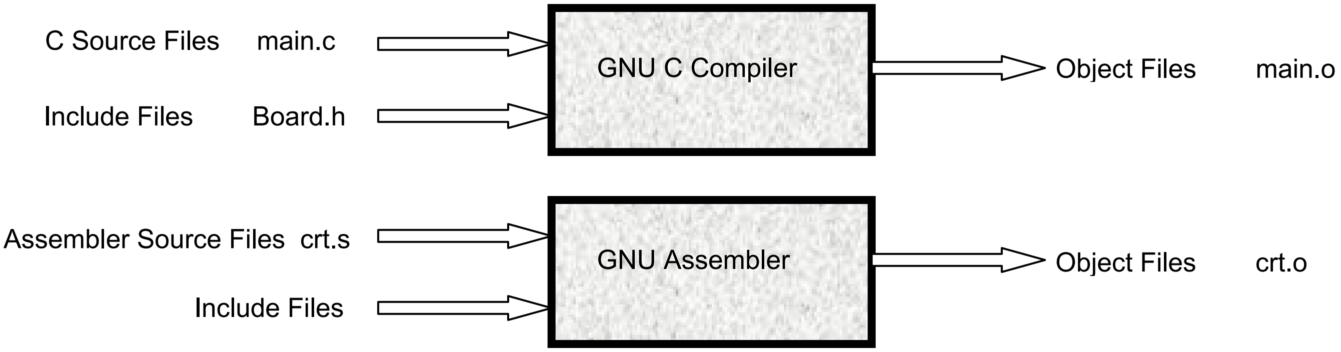
The C compiler and linker used in this tutorial is the Free Software Foundation GNU tool chain for ARM. The GNU C Compiler was first developed by Richard Stallman in 1987 and has been maintained and updated since by a cadre of independent software engineers worldwide. The GNU C compiler is noted for its completeness and wide range of code generators targeting most of the popular microcomputer architectures. In addition to the compiler, the GNU tool chain includes an assembler, linker, make utility debugger, libraries and various other utilities.

This GNU open source C compiler provides a speed and code density performance very close to the best professional compilers from ARM, Keil, Hitex, IAR and others. The GNU Make utility is used by Eclipse to manage your builds and call the proper build tools in the correct sequence. The GNU GDB debugger is fully integrated with the Eclipse IDE to give animated debugging with breakpoints, single stepping and sophisticated inspection of variables and data structures.

Let’s describe how all these software tools are used to develop ARM applications. The Eclipse software editor is used to create C and assembler source files and include files. It is also used to create the make file and linker script file.

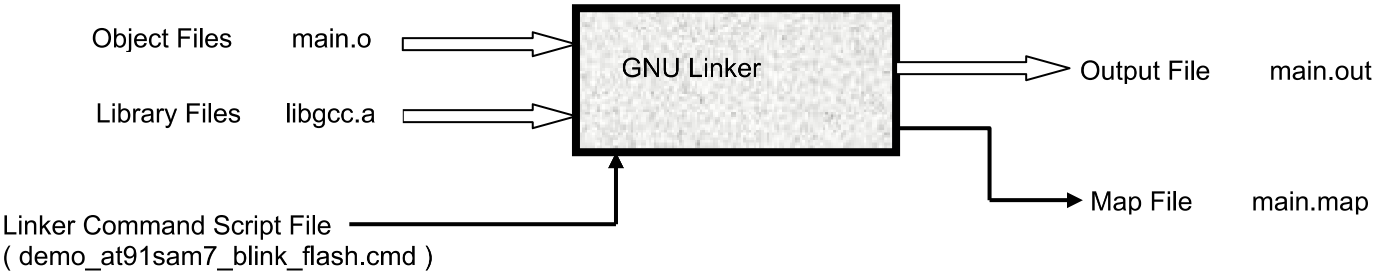


The GNU C Compiler (ARM version) and the GNU assembler (ARM version) are used to compile and assemble the source files. The outputs of the compiler and assembler are object files. Object files are fairly close to the final machine language instructions executed by the ARM chip, but addresses are not filled in. These addresses are resolved and filled in later by the linker (giving the user the ability to load the program anywhere in memory).

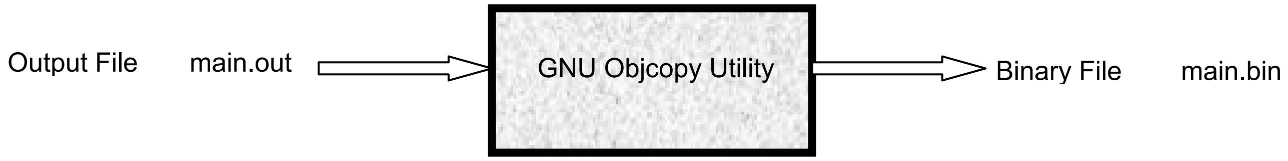


The GNU Linker is used to collect the object files you have created, plus any object modules you need from libraries, resolve all addresses, and combine them into a downloadable output file with the “**.out**” extension. A linker command script file with the extension “**.cmd**” is used to specify the order and target memory location your object modules.

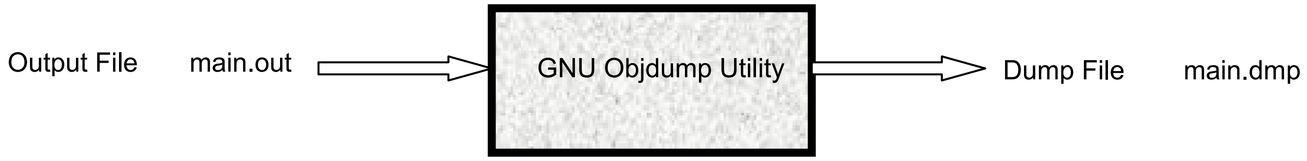
The “**.out**” output file is complex; it includes both machine language executable instructions and debugging information. Normally, this file is used to download into RAM memory for execution exclusively within RAM or to simply assist the Eclipse/GDB debugger in identifying symbols and their memory addresses, etc. The Linker also produces a “**.map**” file which is helpful in determining the lengths of modules, their placement in memory, etc.



If you wish to burn your application into onchip FLASH memory, then a pure binary file is needed by the OpenOCD JTAG debugger or the Atmel SAM-BA flash programming utility. This is created by running the "**.out**" file through the GNU ObjCopy utility to create a "**.bin**" binary file.



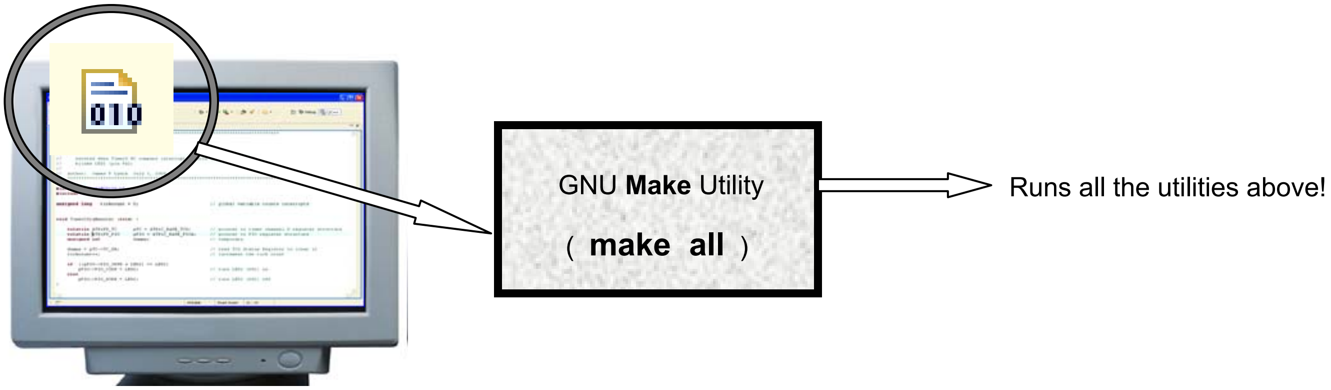
As a completely optional step, the GNU Objdump utility can be used to create a "**.dmp**" dump file which is an embellishment to the map file. If this is of no interest to you, just remove it from the make file.



You literally could do all the above operations by entering commands into a Windows command prompt. However, Eclipse uses the GNU Make utility to automate all this for you. Make scans a makefile that you prepare and executes the above utilities automatically in the proper order. Ever better, using file "dependencies" that you supply, the Make utility only compiles those source files that need it (the ones you just changed). In a large project, this is a real time saver.

When you click on the Eclipse "**Build All**" toolbar button shown below, Eclipse will run the GNU Make utility which will compile and link your project. In this case, Eclipse effectively runs the command "**make all**".

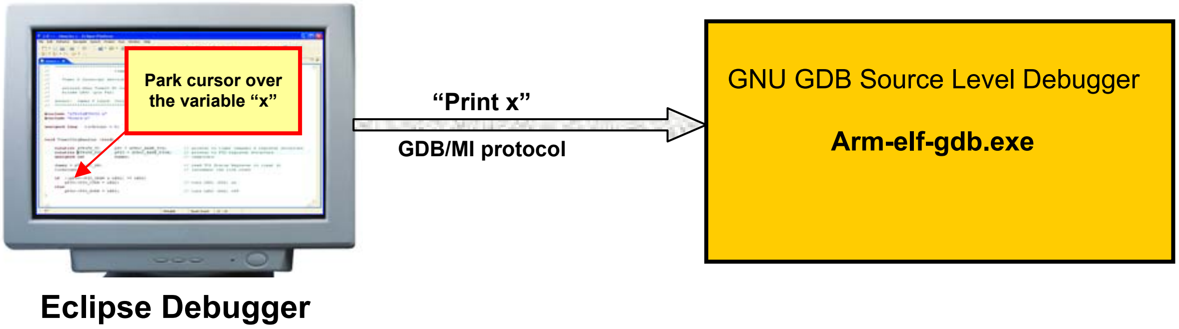
Another toolbar button in Eclipse will run the make utility with the "**program**" target (effectively the command "**make program**" – this will burn your application into onchip flash via the JTAG connection.



Debugging is a bit complicated since the application's execution platform is a circuit board separate from your PC. Several PC programs and a special hardware interface are required to accomplish "remote debugging".

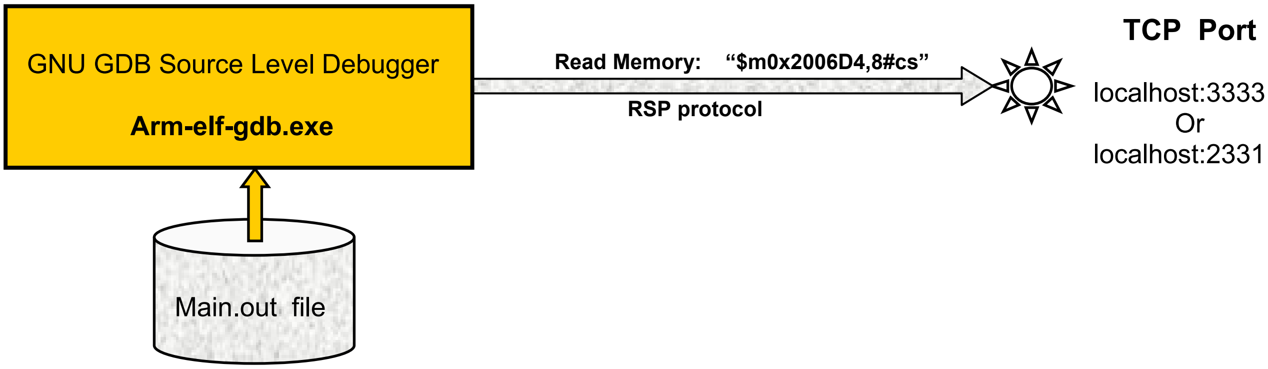
When you start the Eclipse Debugger, Eclipse will automatically start an auxiliary program, the GNU GDB Source Level Debugger (arm-elf-gdb.exe). Eclipse communicates to this program using the GDB/MI protocol which is similar to the command line interface (CLI) that people have been using for years to operate GDB in batch mode.

For example, Eclipse may send the command "print x" to GDB when you park the cursor over the variable "x".


Park cursor over the variable "x"
"Print x"
GDB/MI protocol
GNU GDB Source Level Debugger
Arm-elf-gdb.exe
Eclipse Debugger

GDB has access to your main.out file which has both instructions and symbol information. Using the symbol information, it determines that the variable "x" is a long integer at memory address 0x2006D4. GDB now emits a "read memory" debugging command in a serial protocol called RSP (Remote Serial Protocol).

For example, it may generate a text packet like this: "$m0x2006D4,8#cs" which means read 8 bytes from memory address 0x2006D4. This RSP packet is sent to a TCP port.

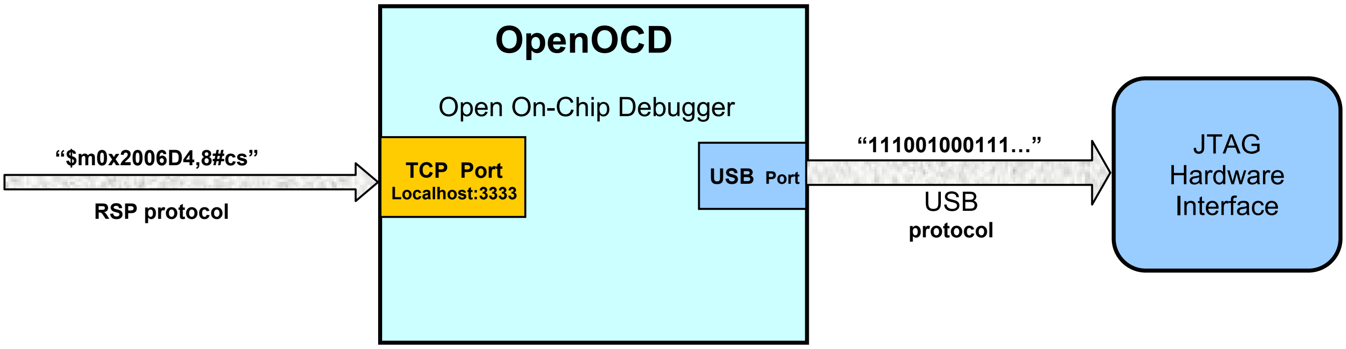


A special daemon server program (a program that operates surreptitiously in the background waiting for commands) is required to accept the RSP protocol debugging commands from GDB and convert them into ARM JTAG protocol commands which will go to the ARM chip's Embedded ICE module. The ARM JTAG protocol is complex; without going into too much detail, it involves clocking bits in and out a 38 bit register using a send line, a receive line and a clock line.

This daemon program will either be **OpenOCD** or the **J-Link GDB Server**; which one depends on the type of hardware JTAG interface you have purchased. The daemon operates in a client-server arrangement. The GNU GDB Source Code Debugger is the client (it makes debugging requests) while the daemon (such as OpenOCD) is the server (it interrogates the ARM chip via the JTAG port and returns the result).

This requires that the daemon (OpenOCD or J-Link GDB Server) must be running before GDB is started.

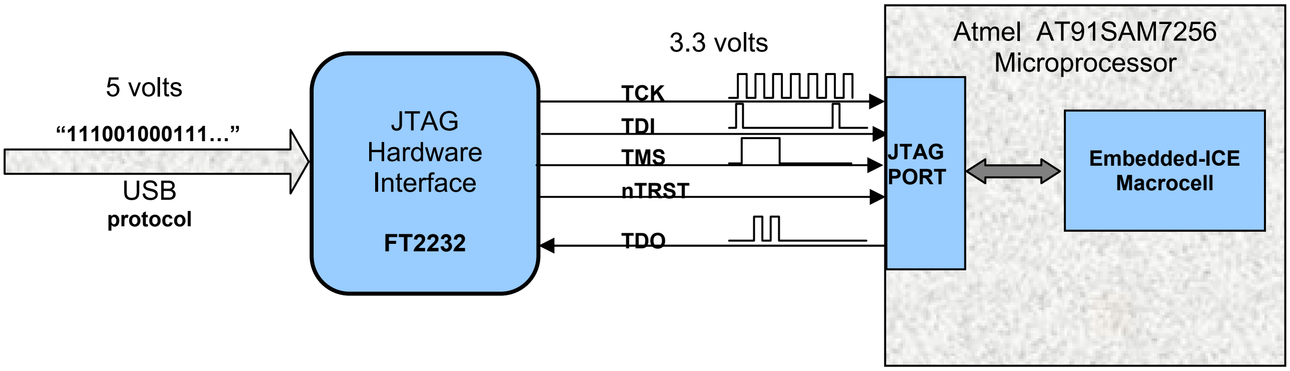
The connection from GDB to the OpenOCD program is via a TCP port named "localhost:3333". Alternately, the connection from GDB to the J-Link GDB Server program is via a TCP port named "localhost:2331". The OpenOCD or J-Link GDB Server then uses the PC's USB port to communicate to the JTAG hardware interface. Note that the OpenOCD daemon can also use the PC's parallel printer port to operate the JTAG lines if you have the inexpensive "wiggler" JTAG device.



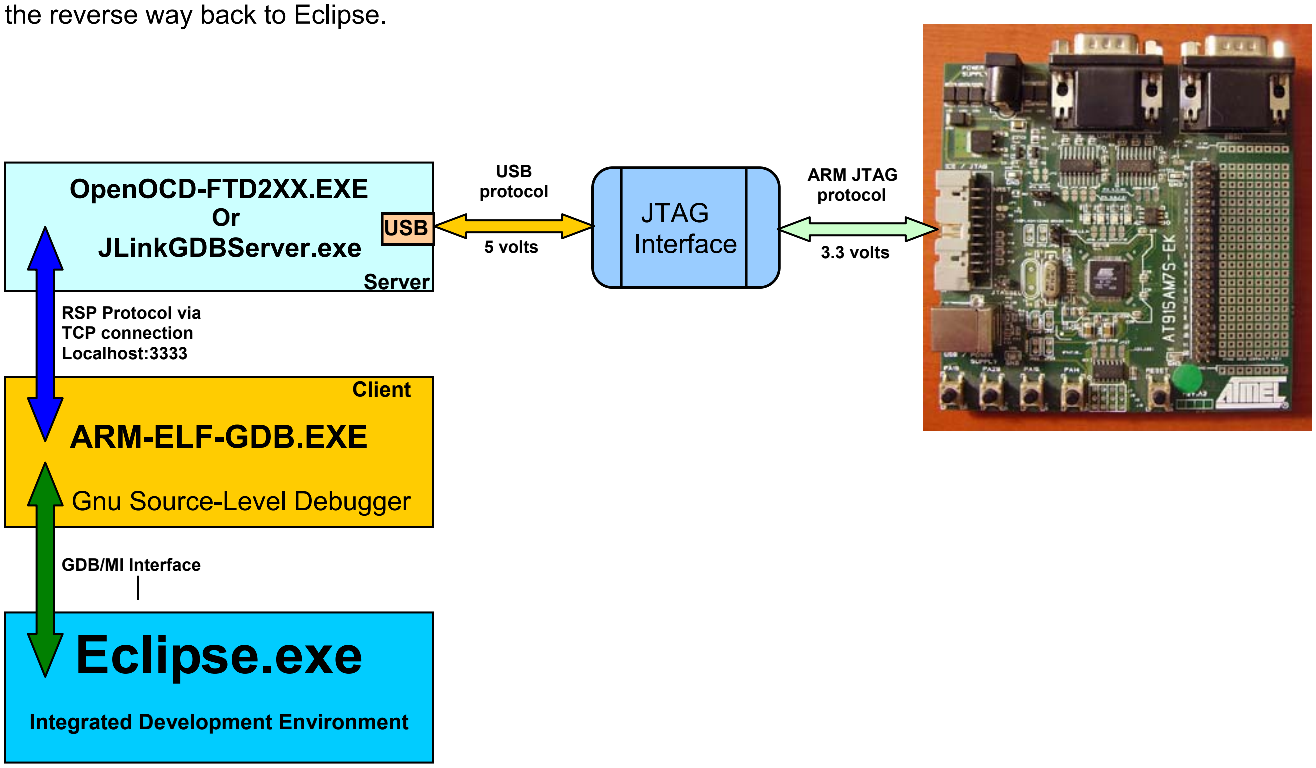
Now we have one final element in our road to debugging, the JTAG hardware interface. The USB port is a high speed serial interface and we have five JTAG lines to manipulate. The JTAG hardware interface converts the USB serial signal to the JTAG clock/data format. Most JTAG/USB hardware debugger manufacturers use the FTD2232 chip that has a "bit-bang" design wherein the incoming USB serial byte is output on 8 bidirectional port pins. These pins are then connected to the JTAG lines of the ARM chip. The FTD2232 circuit also translates the 5 volt USB signal to the 3.3 volt level required to drive the JTAG pins.

If you're using the inexpensive "wiggler" device, the PC printer port lines are simply level-shifted to 3.3 volts and applied directly to the ARM JTAG pins. This works but is notoriously slow and susceptible to ground loop problems.

ARM7 and ARM9 microcontrollers have an Embedded-ICE macrocell. This is a hardware circuit that implements most of the popular debugger functions on-chip. It has two hardware breakpoint/watchpoint circuits that can monitor and then stop instruction flow if a designated address/data combination is encountered (without degrading performance in any way). This means that you can set two breakpoints in applications running in FLASH memory, single step the program; read and write memory and ARM registers; program the onchip flash, and so forth. Not many years ago, this would require a special "break-out" version of the microprocessor or an "in-circuit debugger" or a resident debugging software monitor - all costly solutions.



The diagram below shows the command flow from Eclipse through GDB and OpenOCD/J-Link on its way to your target board's JTAG hardware pins. Results, such as the value of a requested memory read, flow the reverse way back to Eclipse.



The result of all this software cooperation is a nifty graphical debugging environment. If, for example, you park the cursor over a variable name in the source file, Eclipse will ask the GDB Source Level Debugger for it. Using the symbol information in your main.out file, GDB will perform a memory read request on the appropriate memory address. The OpenOCD daemon will convert that request into the complex serial shift register protocol required by the ARM chip's JTAG/Embedded ICE unit. The ARM hardware will read the symbol's value from that address (the processor must be halted to do this) and pass it back to OpenOCD which passes it back to the GDB Source Level Debugger which returns it to Eclipse for display.

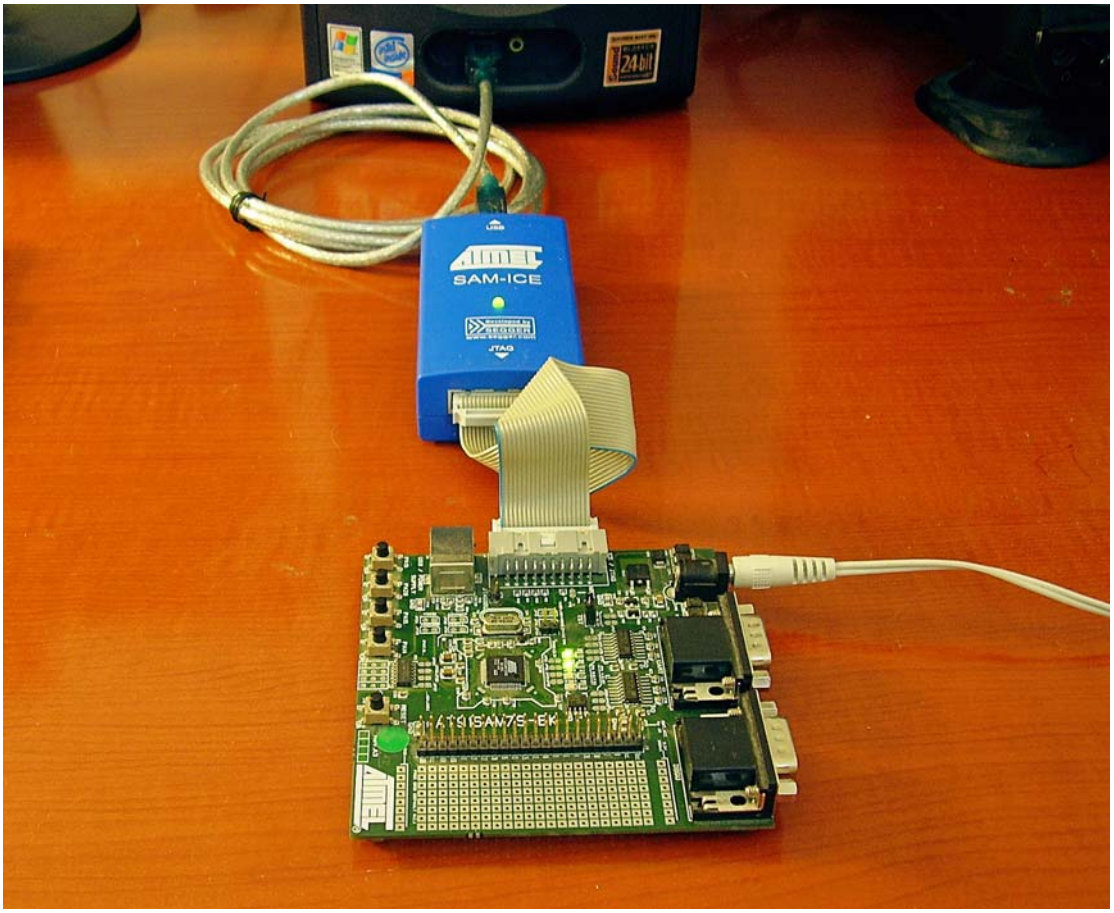
The JTAG hardware choice is usually one of cost. Here are some popular JTAG hardware interfaces available today.

Vendor	Price	Com Port	Software Needed	Comments
ATMEL SAM-ICE	$129.00 (US)	USB	J-Link GDB Server	Branded version of the Segger J-Link
Olimex ARM JTAG	$19.95 (US)	Printer Port	OpenOCD	Called the "wiggler", slow download speed
Olimex ARM-USB-OCD	$69.95 (US)	USB	OpenOCD	extra serial port and 5 volt power for target
Olimex ARM-USB-Tiny	$49.95 (US)	USB	OpenOCD	Hobbyist/Student version
Amontec JTAGKey	$131.78 (US)	USB	OpenOCD	Has extra ESD protection
Amontec JTAGKey-Tiny	$38.60 (US)	USB	OpenOCD	Hobbyist/Student version
Segger J-Link-ARM	$330.11 (US)	USB	J-Link GDB Server	Has extensive software available

The author has tried most of these JTAG interfaces and they all work very well, except for the "wiggler" which can be very temperamental. In any case, it would behoove you to purchase a USB-based hardware interface if you can afford it as parallel ports on PC platforms are rapidly falling out of favor.

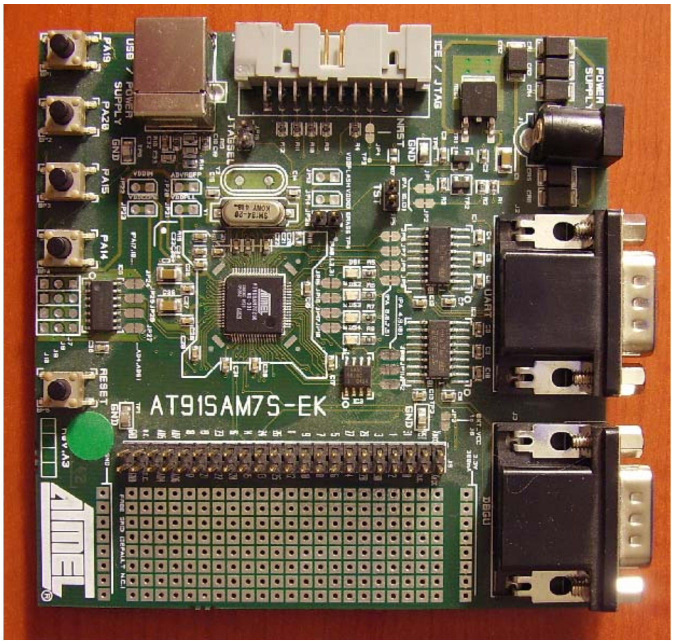
The OpenOCD software daemon which connects the Eclipse/GDB debugger to the Olimex and Amontec JTAG devices is open source and free. Purchasers of the Atmel SAM-ICE also have a free, unlimited license to the Segger J-Link GDB Server.

When you have chosen your JTAG hardware, your setup will look like the one shown below. Here a SAM-ICE JTAG interface is attached to the PC's USB port and the target board's 20-pin JTAG connector. A simple wall-wart 9 volt DC power supply also powers the board.



Target Hardware

As a hardware platform to exercise our ARM cross development tool chain, we will be using the Atmel AT91SAM7S-EK evaluation board, shown directly below.



This board includes two serial ports, a USB port, an Atmel Crypto memory, JTAG connector, four buffered analog inputs, four pushbuttons, four LEDs and a prototyping area.

The Atmel AT91SAM7S256 ARM microcontroller includes 256 Kbytes of on chip FLASH memory and 64 Kbytes of on chip RAM.

The board may be powered from either the USB channel or an external DC power supply (7v to 12v).

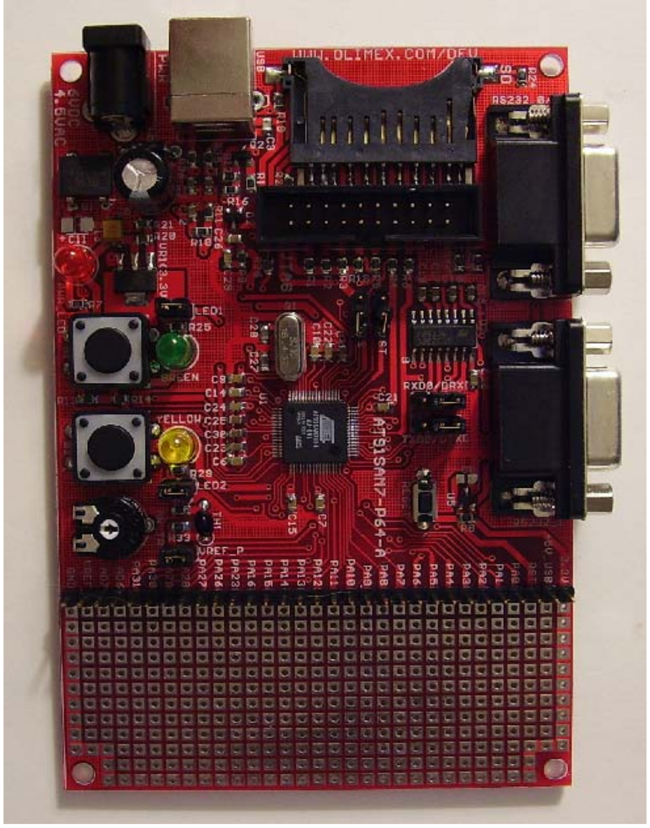
This board is available from Digikey and retails for $149.00 (US) www.digikey.com

There are numerous third party AT91SAM7 boards available. Notable is the Olimex SAM7-P256 shown on the right (Olimex SAM7-P64 board shown, SAM7-P256 board is very similar). This board includes two serial ports, a USB port, expansion SD memory port, two pushbuttons, two LEDs, one analog input with potentiometer and a prototyping area.

The Atmel AT91SAM7S256 ARM microcontroller includes 256 Kbytes of on chip FLASH memory and 64 Kbytes of on chip RAM. The board may be powered from either the USB channel or an external DC power supply (7v to 12v).

This board is available from Olimex, Spark Fun Electronics and Microcontrollershop; it retails for $69.95 (US)

www.olimex.com
www.sparkfun.com
www.microcontrollershop.com



For the rest of this tutorial, we will concentrate on the Atmel AT91SAM7S-EK evaluation board.

The Olimex board can be substituted but the reader must then make minor adjustments since the Olimex board uses different I/O ports for the LEDs.
See Appendix 1 for additional instructions.

Open Source Tools Required

To build this ARM cross development tool chain, we need the following components:

- **Eclipse IDE version 3.2**
- **Eclipse CDT 3.1 Plug-in for C++/C Development (Zylin custom version)**
- **Native GNU C++/C Compiler suite for ARM Targets**
- **OpenOCD version 141 or later for JTAGKey or ARM-USB-OCD JTAG debugging**
- **Segger J-Link GDB Server version 3.70b for SAM-ICE JTAG debugging**
- **Atmel SAM-BA version 2.5 flash programming utility**

The first four components (Eclipse, CDT, GNU Toolchain and OpenOCD) can be downloaded from a single source. The YAGARTO ARM Cross Development Package was assembled by Michael Fischer of Lohfelden, Germany. It includes the latest Eclipse release 3.2 and the Zylin-modified CDT (C/C++ Development Toolkit). The ARM compiler tool chain runs as a Windows native application with no Cygwin DLL required. Michael has also modified the GDB debugger to improve its performance in an embedded debug environment. Rounding out the package is the latest version of OpenOCD (the JTAG debugger). YAGARTO is packaged as four downloads with a fool-proof installer for each. Michael's YAGARTO web site is non-commercial with no affiliation with any manufacturer.

Yagarto may be downloaded from here: http://www.yagarto.de/

The Segger J-Link GDB Server can be downloaded from the Segger web site: http://www.segger.de/

The Atmel SAM-BA flash programming utility can be downloaded from the Atmel web site:

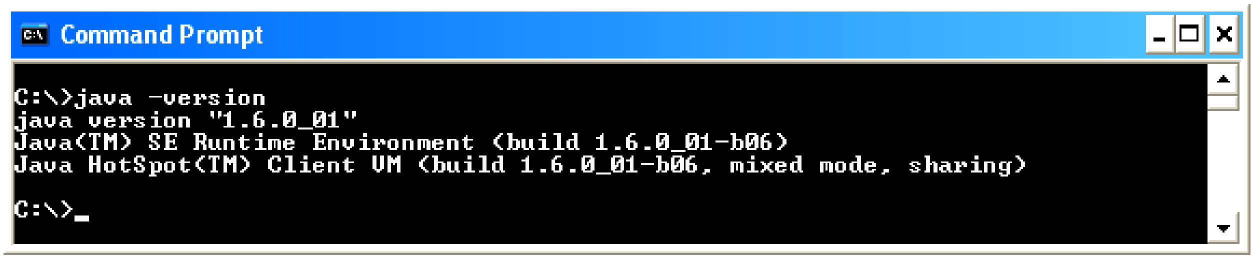
http://www.atmel.com/dyn/products/product_card.asp?part_id=3524

Note: The Eclipse/CDT does NOT run on Windows 98 or Windows ME

Check for JAVA Support

Since the Eclipse Integrated Development Environment (IDE) is written partially in JAVA, we must have JAVA support on our computer to run it. With the recent peace treaty between Microsoft and Sun Microsystems, most recent desktop PCs running Windows 2000 or Windows XP already have JAVA runtime support installed.

To check this, open a command prompt window (click on "**Start – All Programs – Accessories – Command Prompt**") and type the command **c:\>java –version** (thanks to Michael Fischer for this trick).



```
C:\>java -version
java version "1.6.0_01"
Java(TM) SE Runtime Environment (build 1.6.0_01-b06)
Java HotSpot(TM) Client VM (build 1.6.0_01-b06, mixed mode, sharing)

C:\>_
```

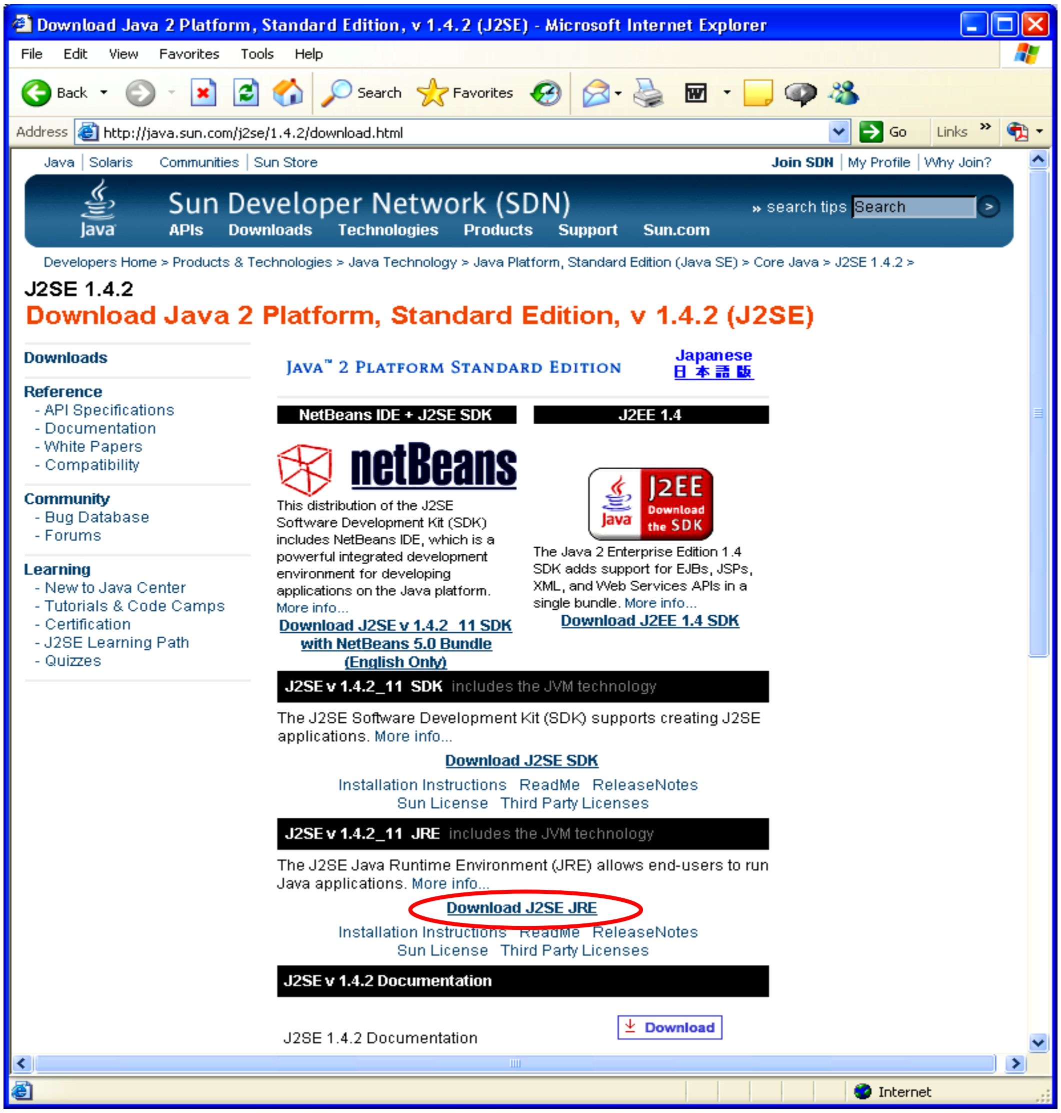
If the command prompt indicates no such program as java.exe, or if the Java version is not **1.6.0_01** or higher, you will need to download and install the JAVA runtime environment as outlined in the instructions below. The author recommends that you **always** have the latest and greatest JAVA runtime installed on your computer. Otherwise, skip to the section “Downloading YAGARTO”.

To install the JAVA Runtime Environment, go to the SUN web site and download it.

http://java.sun.com/j2se/1.4.2/download.html

The Sun JAVA web site is very dynamic so don’t be surprised if the JAVA run time download screens differ slightly from this tutorial.

To support Eclipse, we just need the Sun JAVA Runtime Environment (JRE). Click on “**Download J2SE JRE**” as shown below.



Download Java 2 Platform, Standard Edition, v 1.4.2 (J2SE) - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Address http://java.sun.com/j2se/1.4.2/download.html
Go
Links
Java | Solaris Communities | Sun Store
Join SDN | My Profile | Why Join?
Sun Developer Network (SDN)
APIs Downloads Technologies Products Support Sun.com
» search tips Search
Internet

Developers Home > Products & Technologies > Java Technology > Java Platform, Standard Edition (Java SE) > Core Java > J2SE 1.4.2 >

J2SE 1.4.2

Download Java 2 Platform, Standard Edition, v 1.4.2 (J2SE)

Downloads

Reference
- API Specifications
- Documentation
- White Papers
- Compatibility

Community
- Bug Database
- Forums

Learning
- New to Java Center
- Tutorials & Code Camps
- Certification
- J2SE Learning Path
- Quizzes

JAVA™ 2 PLATFORM STANDARD EDITION

Japanese 日本語版

NetBeans IDE + J2SE SDK

netBeans

This distribution of the J2SE Software Development Kit (SDK) includes NetBeans IDE, which is a powerful integrated development environment for developing applications on the Java platform. More info...

Download J2SE v 1.4.2_11 SDK with NetBeans 5.0 Bundle (English Only)

J2EE 1.4

J2EE Download the SDK

The Java 2 Enterprise Edition 1.4 SDK adds support for EJBs, JSPs, XML, and Web Services APIs in a single bundle. More info...

Download J2EE 1.4 SDK

J2SE v 1.4.2_11 SDK includes the JVM technology

The J2SE Software Development Kit (SDK) supports creating J2SE applications. More info...

Download J2SE SDK

Installation Instructions ReadMe ReleaseNotes
Sun License Third Party Licenses

J2SE v 1.4.2_11 JRE includes the JVM technology

The J2SE Java Runtime Environment (JRE) allows end-users to run Java applications. More info...

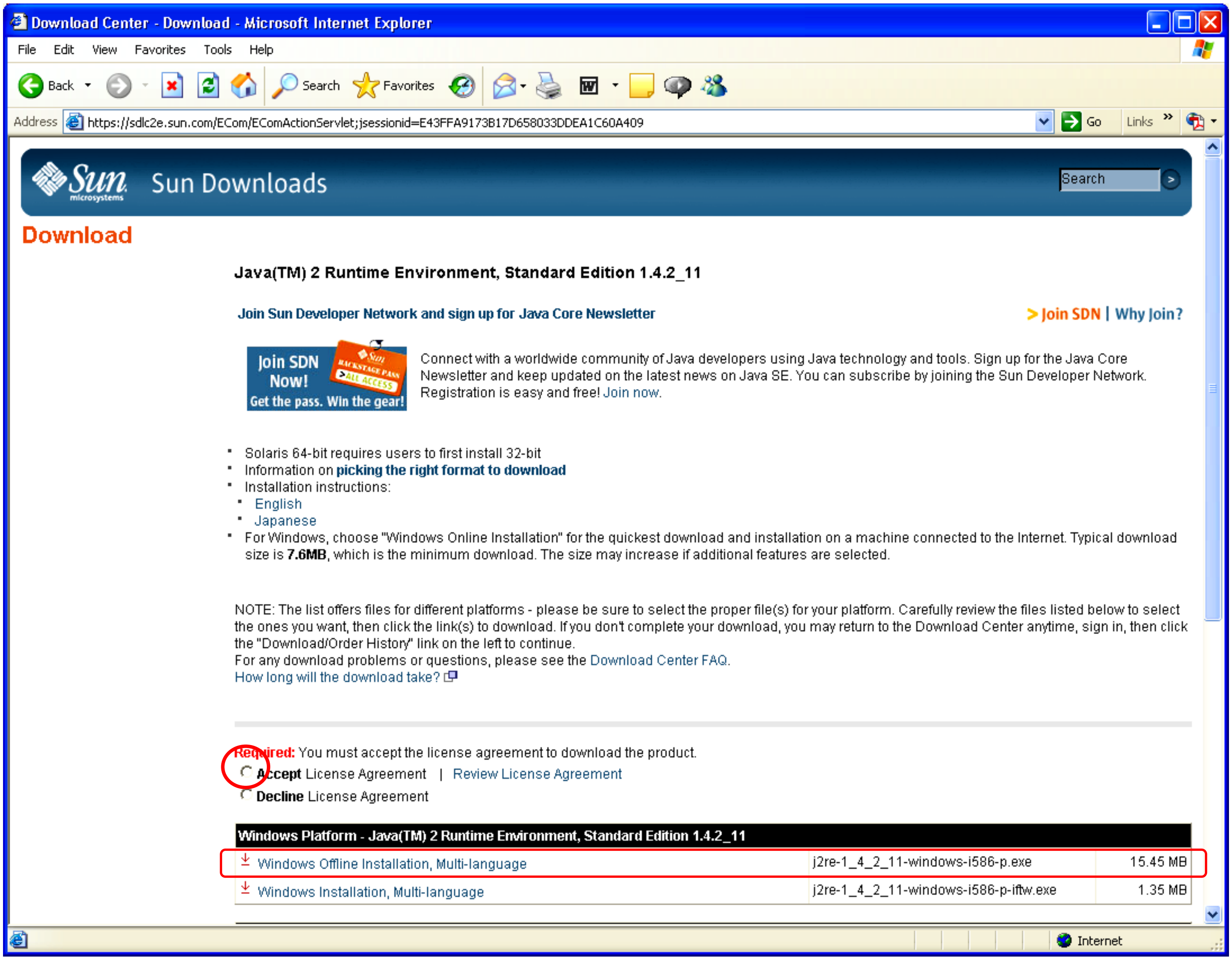
Download J2SE JRE

Installation Instructions ReadMe ReleaseNotes
Sun License Third Party Licenses

J2SE v 1.4.2 Documentation

J2SE 1.4.2 Documentation Download

In the next download screen, shown below, click the radio button “**Accept License Agreement**” and then click on “**Windows Offline Installation, Multi-language**”.


Download Center - Download - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Go
Links
Internet
Sun microsystems
Sun Downloads
Search
Join SDN Now! Get the pass. Win the gear!
BACKSTAGE PASS
ALL ACCESS

Address: https://sdlc2e.sun.com/ECom/EComActionServlet;jsessionid=E43FFA9173B17D658033DDEA1C60A409

Download

Java(TM) 2 Runtime Environment, Standard Edition 1.4.2_11

Join Sun Developer Network and sign up for Java Core Newsletter > **Join SDN** | **Why Join?**

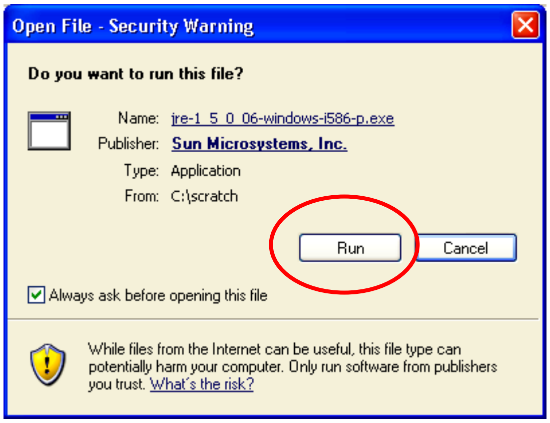
Connect with a worldwide community of Java developers using Java technology and tools. Sign up for the Java Core Newsletter and keep updated on the latest news on Java SE. You can subscribe by joining the Sun Developer Network. Registration is easy and free! Join now.

- Solaris 64-bit requires users to first install 32-bit
- Information on **picking the right format to download**
- Installation instructions:
  - English
  - Japanese
- For Windows, choose "Windows Online Installation" for the quickest download and installation on a machine connected to the Internet. Typical download size is **7.6MB**, which is the minimum download. The size may increase if additional features are selected.

NOTE: The list offers files for different platforms - please be sure to select the proper file(s) for your platform. Carefully review the files listed below to select the ones you want, then click the link(s) to download. If you don't complete your download, you may return to the Download Center anytime, sign in, then click the "Download/Order History" link on the left to continue.
For any download problems or questions, please see the Download Center FAQ.
How long will the download take?

Required: You must accept the license agreement to download the product.
○ **Accept** License Agreement | Review License Agreement
○ **Decline** License Agreement

Windows Platform - Java(TM) 2 Runtime Environment, Standard Edition 1.4.2_11		
Windows Offline Installation, Multi-language	j2re-1_4_2_11-windows-i586-p.exe	15.45 MB
Windows Installation, Multi-language	j2re-1_4_2_11-windows-i586-p-iftw.exe	1.35 MB



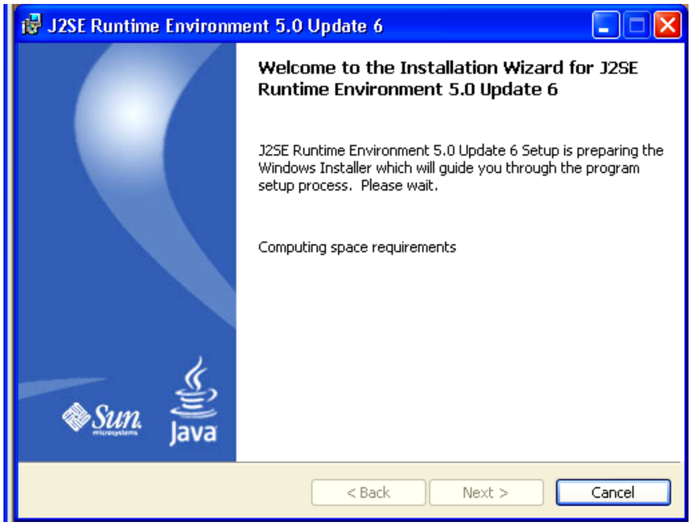
Do you want to run this file?

Name: jre-1_5_0_06-windows-i586-p.exe
Publisher: **Sun Microsystems, Inc.**
Type: Application
From: C:\scratch

☑ Always ask before opening this file

While files from the Internet can be useful, this file type can potentially harm your computer. Only run software from publishers you trust. What's the risk?

Now the Sun JAVA runtime installation engine will start. Click “**Run**” to start the installer.



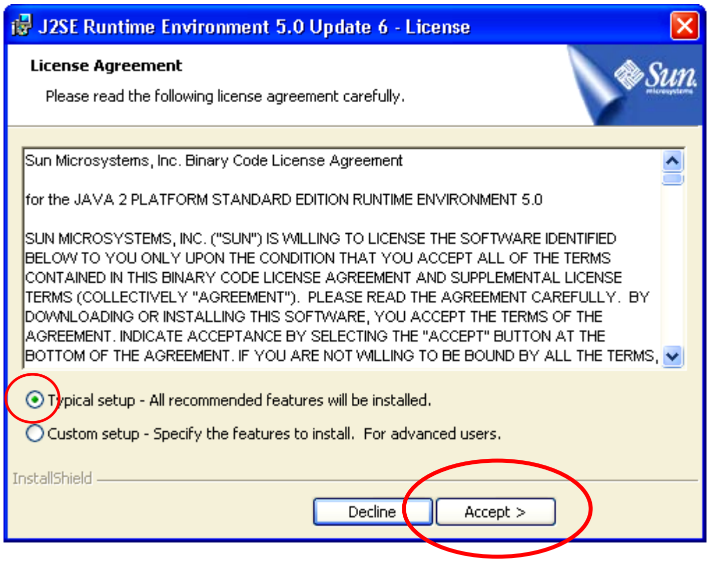
J2SE Runtime Environment 5.0 Update 6

Welcome to the Installation Wizard for J2SE Runtime Environment 5.0 Update 6

J2SE Runtime Environment 5.0 Update 6 Setup is preparing the Windows Installer which will guide you through the program setup process. Please wait.

Computing space requirements

Click on the "**Typical Setup**" radio button and then accept the license terms. JAVA is free; Sun has recently converted JAVA into an Open Source product.



J2SE Runtime Environment 5.0 Update 6 - License

License Agreement

Please read the following license agreement carefully.

Sun Microsystems, Inc. Binary Code License Agreement

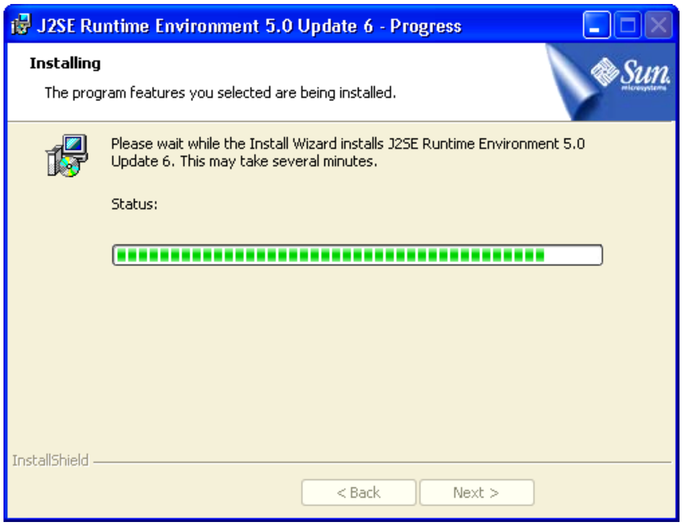
for the JAVA 2 PLATFORM STANDARD EDITION RUNTIME ENVIRONMENT 5.0

SUN MICROSYSTEMS, INC. ("SUN") IS WILLING TO LICENSE THE SOFTWARE IDENTIFIED BELOW TO YOU ONLY UPON THE CONDITION THAT YOU ACCEPT ALL OF THE TERMS CONTAINED IN THIS BINARY CODE LICENSE AGREEMENT AND SUPPLEMENTAL LICENSE TERMS (COLLECTIVELY "AGREEMENT"). PLEASE READ THE AGREEMENT CAREFULLY. BY DOWNLOADING OR INSTALLING THIS SOFTWARE, YOU ACCEPT THE TERMS OF THE AGREEMENT. INDICATE ACCEPTANCE BY SELECTING THE "ACCEPT" BUTTON AT THE BOTTOM OF THE AGREEMENT. IF YOU ARE NOT WILLING TO BE BOUND BY ALL THE TERMS,

● Typical setup - All recommended features will be installed.

○ Custom setup - Specify the features to install. For advanced users.

A series of installation progress screens will appear. Installation only takes a couple of minutes.



J2SE Runtime Environment 5.0 Update 6 - Progress

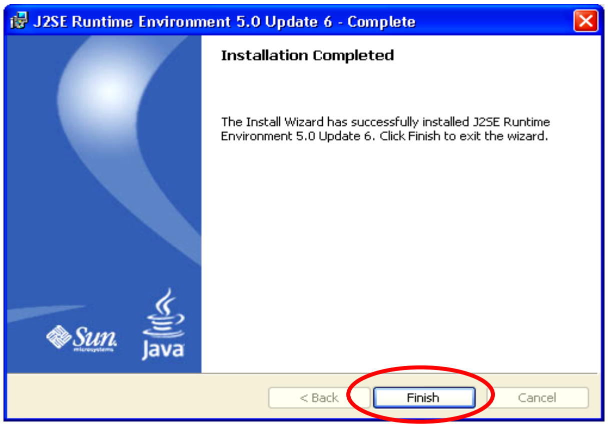
Installing

The program features you selected are being installed.

Please wait while the Install Wizard installs J2SE Runtime Environment 5.0 Update 6. This may take several minutes.

Status:

When the JAVA runtime installation completes, click on "**Finish**" to exit the installer.

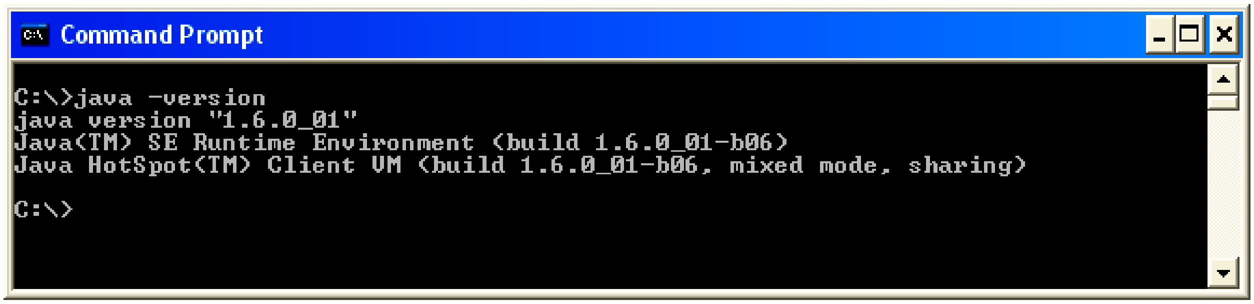


J2SE Runtime Environment 5.0 Update 6 - Complete

Installation Completed

The Install Wizard has successfully installed J2SE Runtime Environment 5.0 Update 6. Click Finish to exit the wizard.

To check that JAVA has been installed, create a command prompt (click on "**Start – All Programs – Accessories – Command Prompt**") and type the command **c:\>java –version**



```
C:\>java -version
java version "1.6.0_01"
Java(TM) SE Runtime Environment (build 1.6.0_01-b06)
Java HotSpot(TM) Client VM (build 1.6.0_01-b06, mixed mode, sharing)

C:\>
```

Note: The current revision of YAGARTO **requires** the latest JAVA runtime environment (version 1.6.0_01)

Downloading YAGARTO

Michael Fischer of Lohfelden, Germany has put together a native version of the GNU compiler tool chain for ARM targets based on MinGW (Minimalist GNU for Windows) and called it YAGARTO (**Y**ET **A**NOTHER **G**NU **AR**M **TO**OL CHAIN). The compiler suite does not require the Cygwin package and is therefore a bit more efficient running in a Windows environment.

Eclipse, a superior open-source Integrated Development Environment (IDE), coupled with the C Development Toolkit (CDT) plug-in provides an editor and source code debugger.

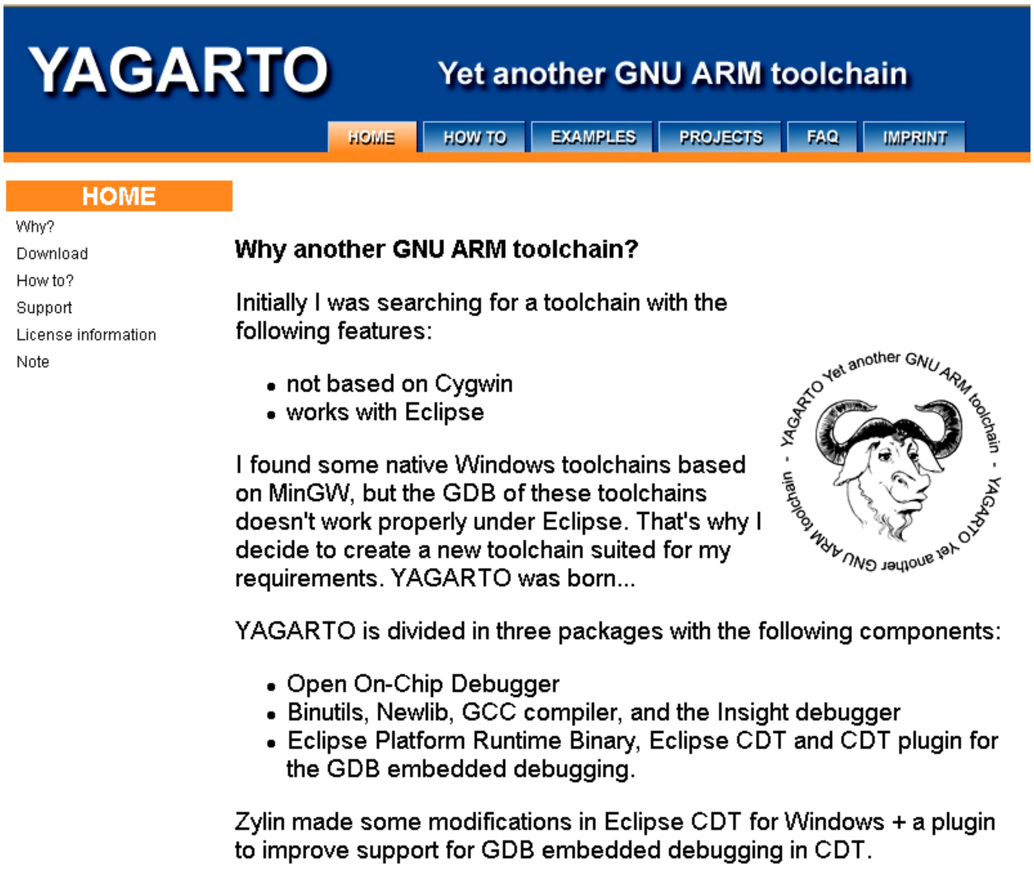
The OpenOCD JTAG debugger, developed by German student Dominic Rath, interfaces the Eclipse GDB source code debugger with the AT91SAM7S JTAG port. OpenOCD supports run/stop control, memory and register inspection, software and hardware breakpoints and can also be used to program the AT91SAM7S internal FLASH memory.

Each of these four components (compiler, Eclipse IDE, YAGARTO Tools and OpenOCD) are downloaded separately and each has its own automatic installer that is fool-proof and convenient.

Michael Fischer's YAGARTO web site, which is loaded with great software examples and tutorials, can be accessed at the following link.

http://www.yagarto.de

The YAGARTO web site should look something like this, shown below.


YAGARTO
Yet another GNU ARM toolchain
HOME
HOW TO
EXAMPLES
PROJECTS
FAQ
IMPRINT
HOME
Why?
Download
How to?
Support
License information
Note
Yet another GNU ARM toolchain - YAGARTO - Yet another GNU ARM toolchain - YAGARTO -

Why another GNU ARM toolchain?

Initially I was searching for a toolchain with the following features:

- not based on Cygwin
- works with Eclipse

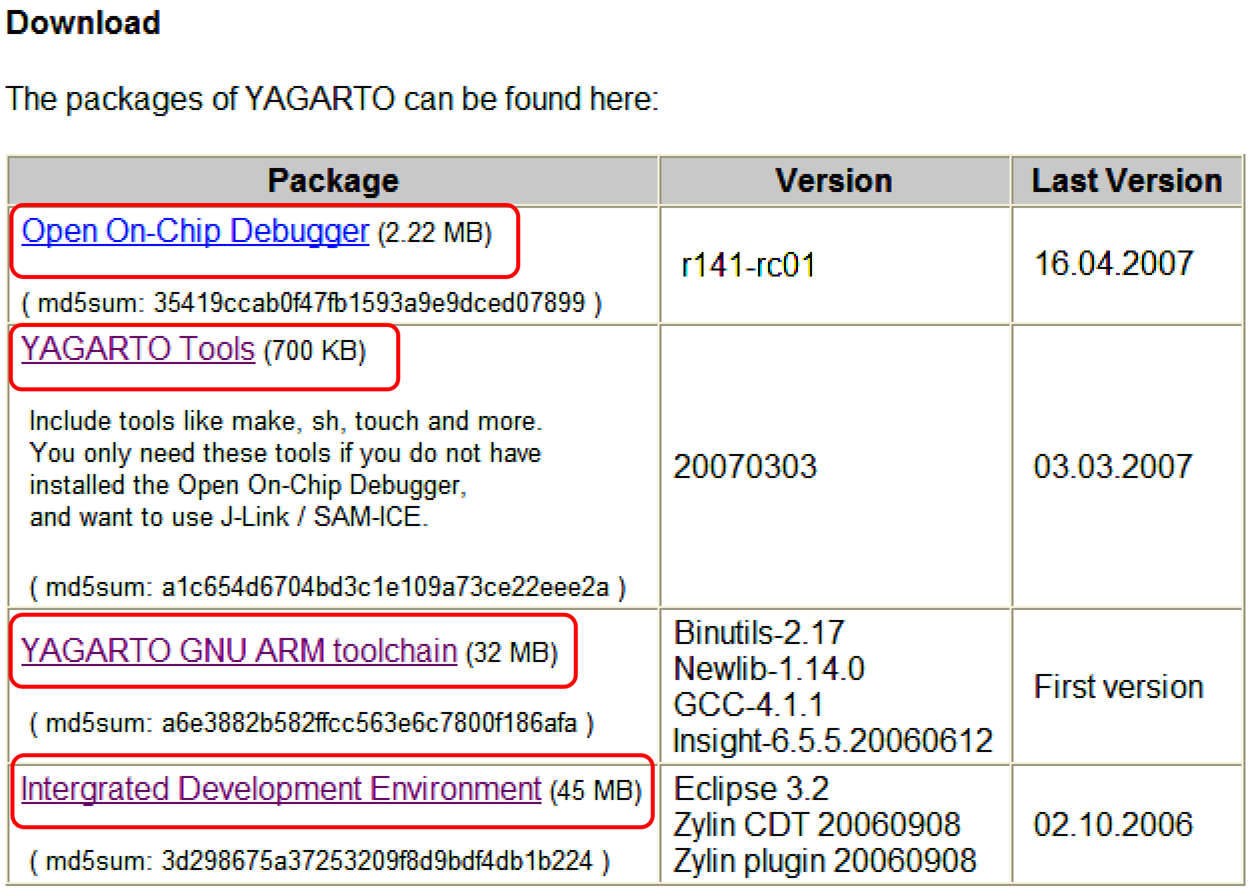
I found some native Windows toolchains based on MinGW, but the GDB of these toolchains doesn't work properly under Eclipse. That's why I decide to create a new toolchain suited for my requirements. YAGARTO was born...

YAGARTO is divided in three packages with the following components:

- Open On-Chip Debugger
- Binutils, Newlib, GCC compiler, and the Insight debugger
- Eclipse Platform Runtime Binary, Eclipse CDT and CDT plugin for the GDB embedded debugging.

Zylin made some modifications in Eclipse CDT for Windows + a plugin to improve support for GDB embedded debugging in CDT.

Scroll down the YAGARTO web site until you see the four download components displayed, as shown below.



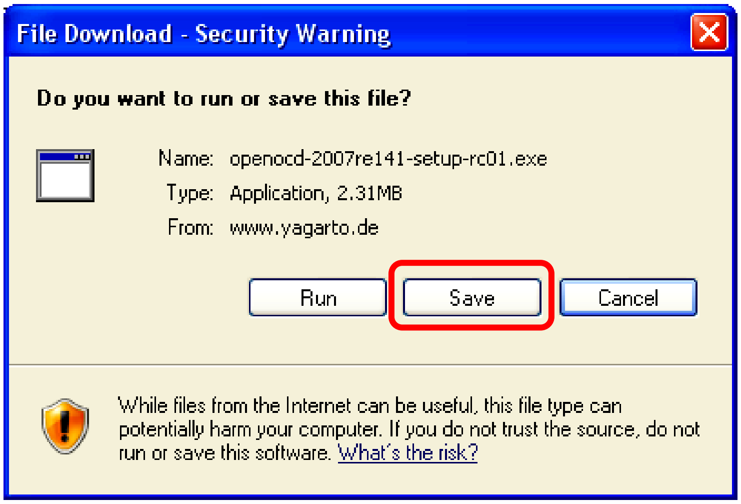
Download

The packages of YAGARTO can be found here:

Package	Version	Last Version
Open On-Chip Debugger (2.22 MB) (md5sum: 35419ccab0f47fb1593a9e9dced07899)	r141-rc01	16.04.2007
YAGARTO Tools (700 KB) Include tools like make, sh, touch and more. You only need these tools if you do not have installed the Open On-Chip Debugger, and want to use J-Link / SAM-ICE. (md5sum: a1c654d6704bd3c1e109a73ce22eee2a)	20070303	03.03.2007
YAGARTO GNU ARM toolchain (32 MB) (md5sum: a6e3882b582ffcc563e6c7800f186afa)	Binutils-2.17 Newlib-1.14.0 GCC-4.1.1 Insight-6.5.5.20060612	First version
Intergrated Development Environment (45 MB) (md5sum: 3d298675a37253209f8d9bdf4db1b224)	Eclipse 3.2 Zylin CDT 20060908 Zylin plugin 20060908	02.10.2006

Using Windows Explorer, create an empty folder called "c:\download" to hold the four downloaded YAGARTO installation packages. This will let us easily reinstall things if we make a mistake.

Click on the link for the "**Open On-Chip Debugger**" package as shown above. We're going to save these packages in the **"**c:\download**"** folder and run them later. Select "**Save**" as shown below on the left and then specify the download folder "c:\download" as shown on the right below. Click "**Save**" in the "**Save As"** screen below on the right to start the download process.

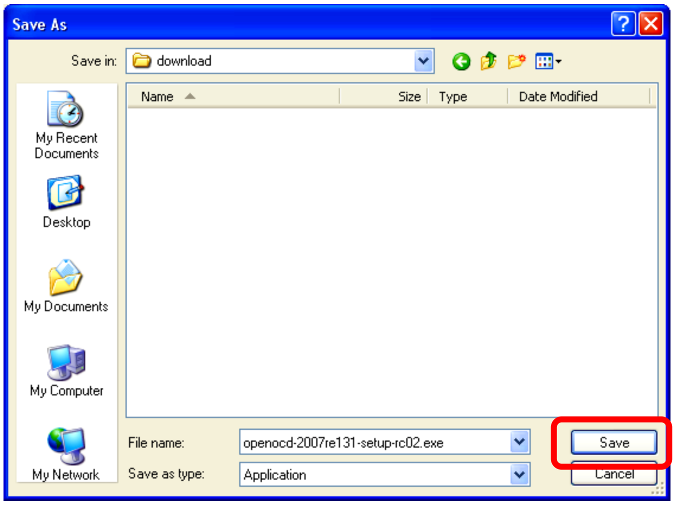


File Download - Security Warning

Do you want to run or save this file?

Name: openocd-2007re141-setup-rc01.exe
Type: Application, 2.31MB
From: www.yagarto.de

While files from the Internet can be useful, this file type can potentially harm your computer. If you do not trust the source, do not run or save this software. What's the risk?

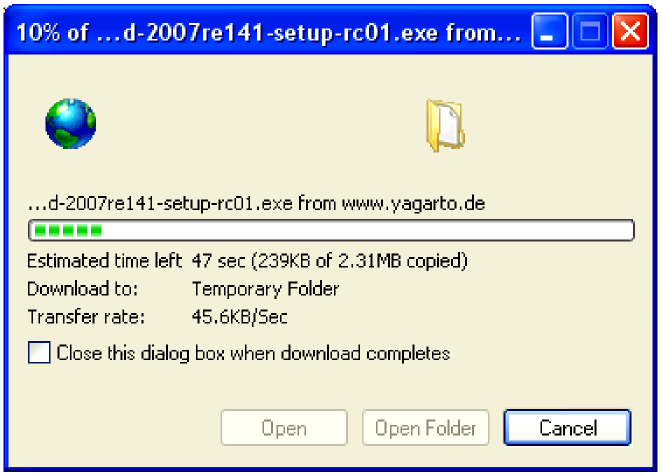


Save As

Save in: download

File name: openocd-2007re131-setup-rc02.exe
Save as type: Application

The OpenOCD (Open On-Chip Debugger) package downloads quickly since it is only 2.2 Mb. Click on "**Close**" as shown below on the right to finish the download.

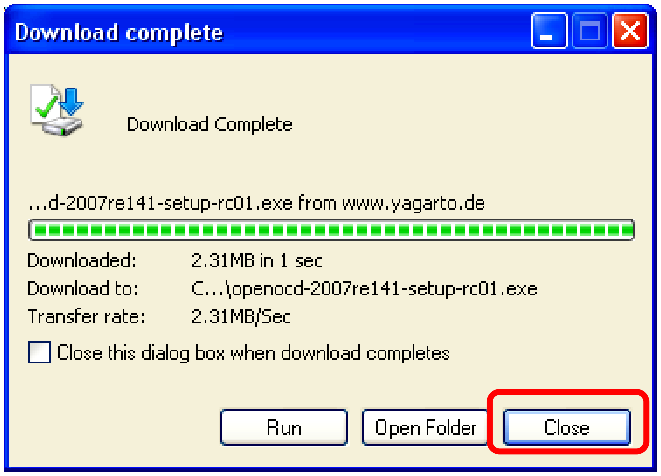


10% of ...d-2007re141-setup-rc01.exe from...

...d-2007re141-setup-rc01.exe from www.yagarto.de

Estimated time left 47 sec (239KB of 2.31MB copied)
Download to: Temporary Folder
Transfer rate: 45.6KB/Sec

□ Close this dialog box when download completes



Download complete

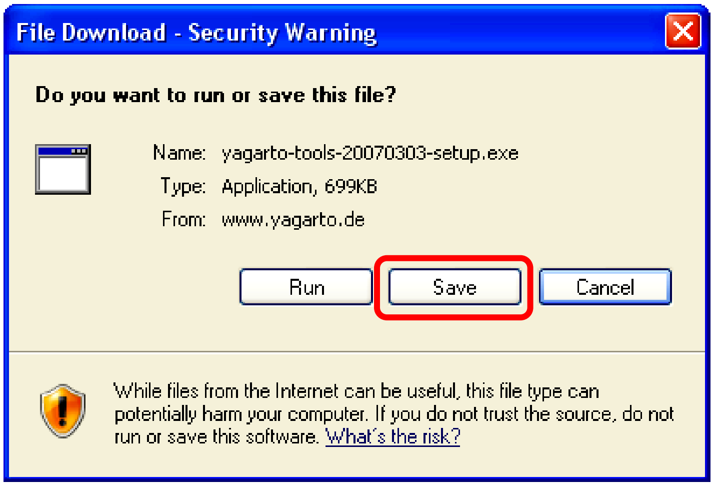
Download Complete

...d-2007re141-setup-rc01.exe from www.yagarto.de

Downloaded: 2.31MB in 1 sec
Download to: C...\openocd-2007re141-setup-rc01.exe
Transfer rate: 2.31MB/Sec

□ Close this dialog box when download completes

Click on the link for the "**Yagarto Tools**" package as shown in the Yagarto Download section above. Select "**Save**" as shown below on the left and then specify the download folder "**c:\download**" as shown on the right below. Click "**Save**" in the "**Save As"** screen below on the right to start the download process.

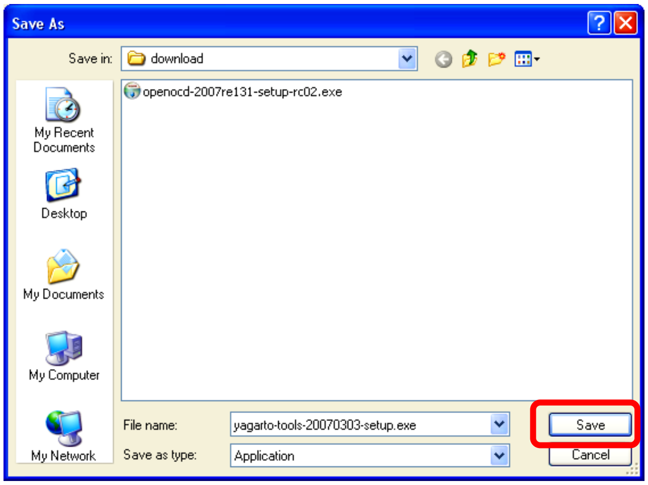


File Download - Security Warning

Do you want to run or save this file?

Name: yagarto-tools-20070303-setup.exe
Type: Application, 699KB
From: www.yagarto.de

While files from the Internet can be useful, this file type can potentially harm your computer. If you do not trust the source, do not run or save this software. What's the risk?



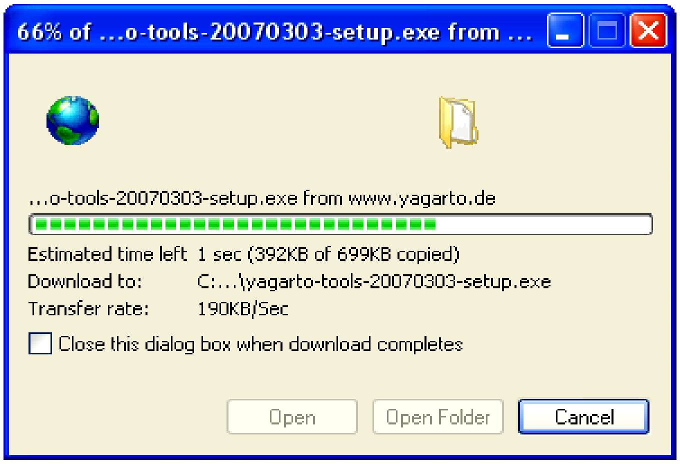
Save As

Save in: download

openocd-2007re131-setup-rc02.exe

File name: yagarto-tools-20070303-setup.exe
Save as type: Application

The "Yagarto Tools" package downloads quickly since it is only 700 Kb. Click on "**Close**" as shown below on the right to finish the download.

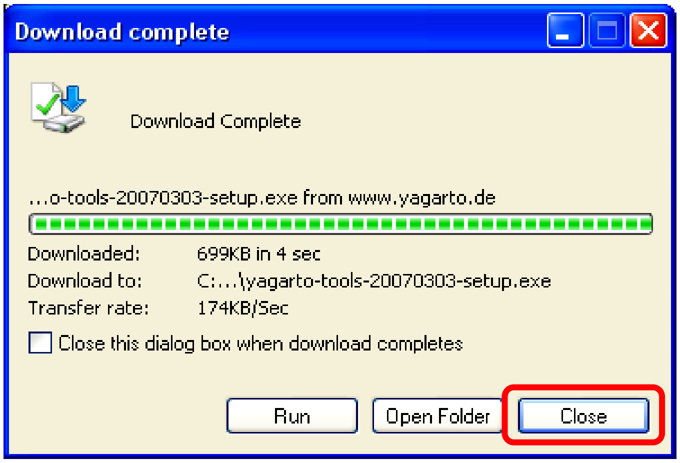


66% of ...o-tools-20070303-setup.exe from ...

...o-tools-20070303-setup.exe from www.yagarto.de

Estimated time left 1 sec (392KB of 699KB copied)
Download to: C:...\yagarto-tools-20070303-setup.exe
Transfer rate: 190KB/Sec

□ Close this dialog box when download completes



Download complete

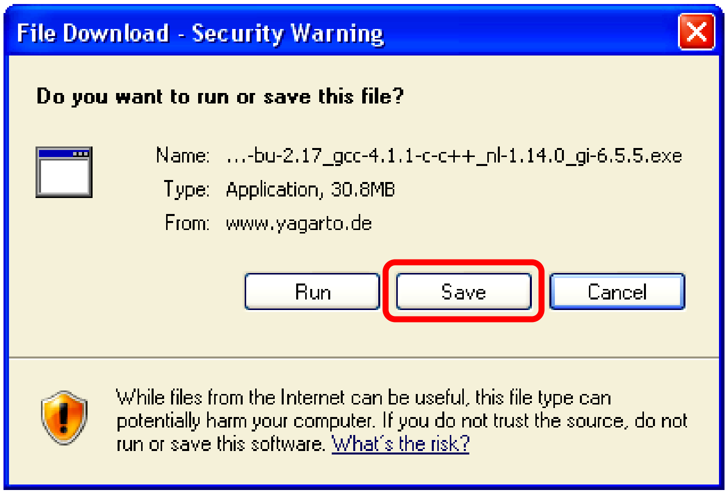
Download Complete

...o-tools-20070303-setup.exe from www.yagarto.de

Downloaded: 699KB in 4 sec
Download to: C:...\yagarto-tools-20070303-setup.exe
Transfer rate: 174KB/Sec

□ Close this dialog box when download completes

Now click on the link for the “**Yagarto GNU ARM toolchain**” package as shown in the Yagarto Download section above. Select “**Save**” as shown below on the left and then specify the download folder “**c:\download**” as shown on the right below. Click “**Save**” in the “**Save As”** screen below on the right to start the download process.



File Download - Security Warning

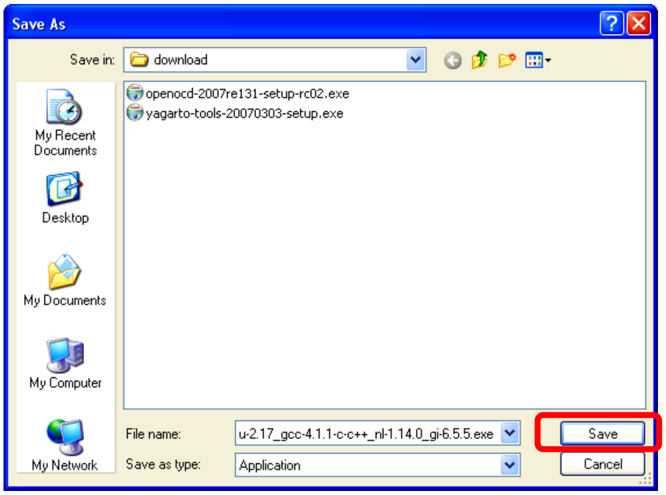
Do you want to run or save this file?

Name: …-bu-2.17_gcc-4.1.1-c-c++_nl-1.14.0_gi-6.5.5.exe

Type: Application, 30.8MB

From: www.yagarto.de

While files from the Internet can be useful, this file type can potentially harm your computer. If you do not trust the source, do not run or save this software. What's the risk?



Save As

Save in: download

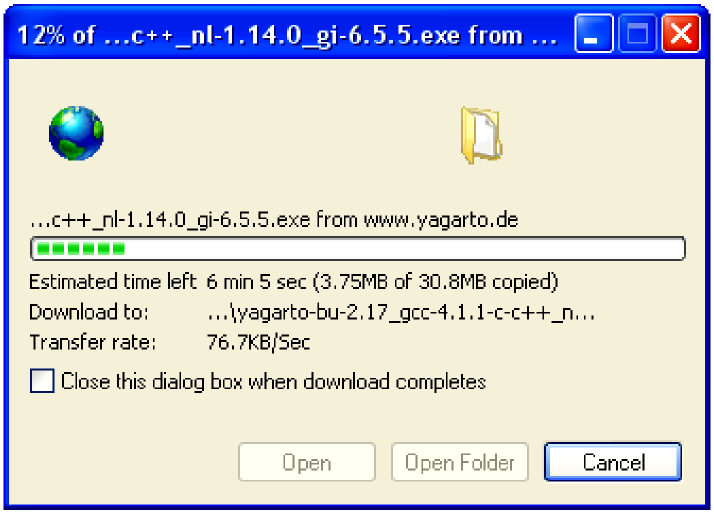
openocd-2007re131-setup-rc02.exe

yagarto-tools-20070303-setup.exe

File name: u-2.17_gcc-4.1.1-c-c++_nl-1.14.0_gi-6.5.5.exe

Save as type: Application

The “Yagarto GNU ARM toolchain” package takes several minutes to download since it is 30.8 Mb. Click on “**Close**” as shown below on the right to finish the download.



12% of …c++_nl-1.14.0_gi-6.5.5.exe from …

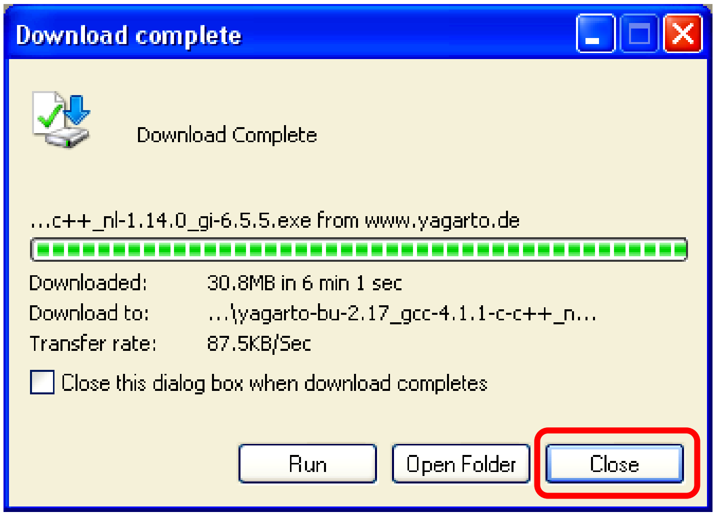
…c++_nl-1.14.0_gi-6.5.5.exe from www.yagarto.de

Estimated time left 6 min 5 sec (3.75MB of 30.8MB copied)

Download to: …\yagarto-bu-2.17_gcc-4.1.1-c-c++_n…

Transfer rate: 76.7KB/Sec

☐ Close this dialog box when download completes



Download complete

Download Complete

…c++_nl-1.14.0_gi-6.5.5.exe from www.yagarto.de

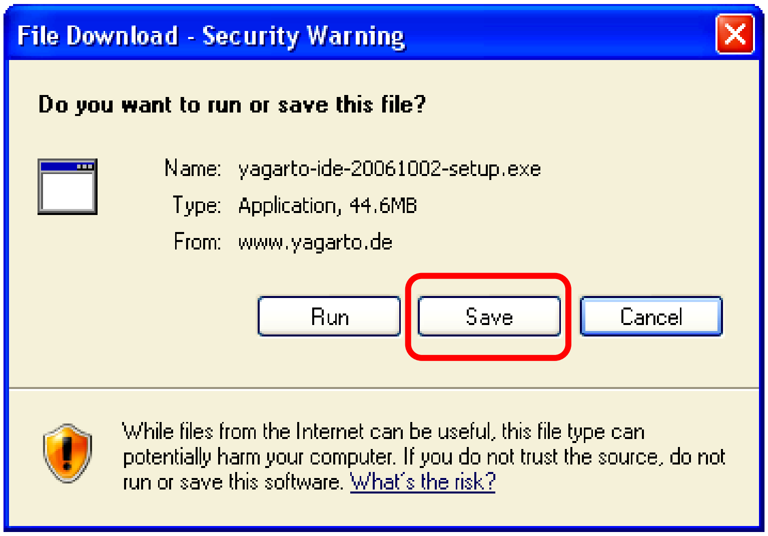
Downloaded: 30.8MB in 6 min 1 sec

Download to: …\yagarto-bu-2.17_gcc-4.1.1-c-c++_n…

Transfer rate: 87.5KB/Sec

☐ Close this dialog box when download completes

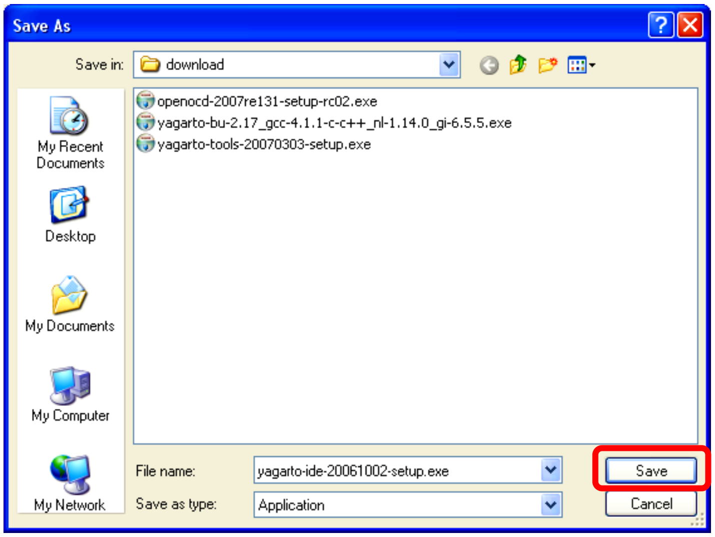
Click on the link for the “**Integrated Development Environment**” package as shown in the Yagarto Download section above. Select “**Save**” as shown below on the left and then specify the download folder “c:\download” as shown on the right below. Click “**Save**” in the “**Save As”** screen below on the right to start the download process.



Do you want to run or save this file?

Name: yagarto-ide-20061002-setup.exe
Type: Application, 44.6MB
From: www.yagarto.de

While files from the Internet can be useful, this file type can potentially harm your computer. If you do not trust the source, do not run or save this software. What's the risk?

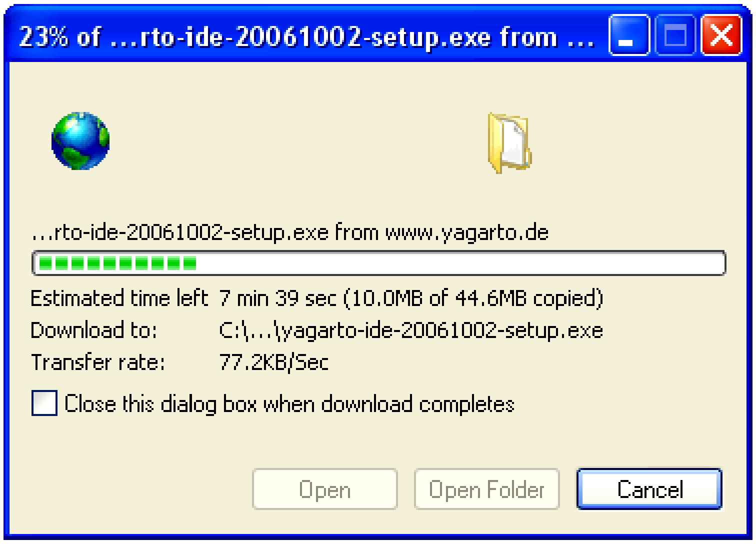


Save in: download

openocd-2007re131-setup-rc02.exe
yagarto-bu-2.17_gcc-4.1.1-c-c++_nl-1.14.0_gi-6.5.5.exe
yagarto-tools-20070303-setup.exe

File name: yagarto-ide-20061002-setup.exe
Save as type: Application

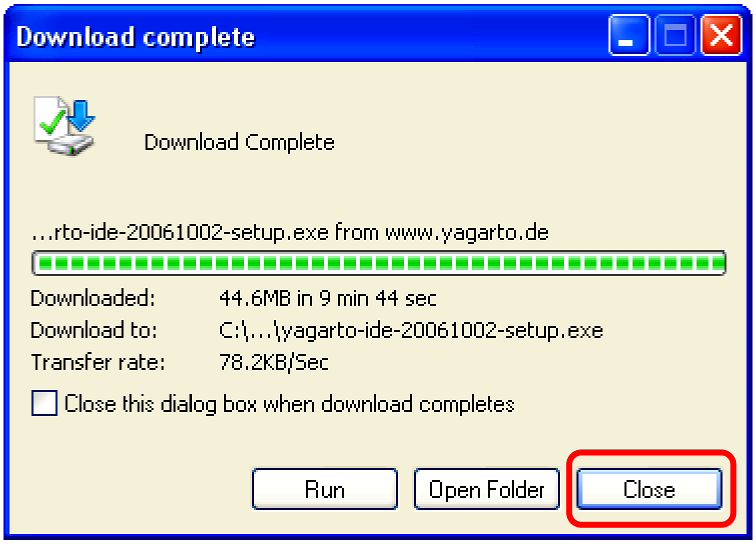
The “Integrated Development Environment” package takes several minutes to download since it is 44.6 Mb. Click on “**Close**” as shown below on the right to finish the download.



...rto-ide-20061002-setup.exe from www.yagarto.de

Estimated time left 7 min 39 sec (10.0MB of 44.6MB copied)
Download to: C:\...\yagarto-ide-20061002-setup.exe
Transfer rate: 77.2KB/Sec

□ Close this dialog box when download completes



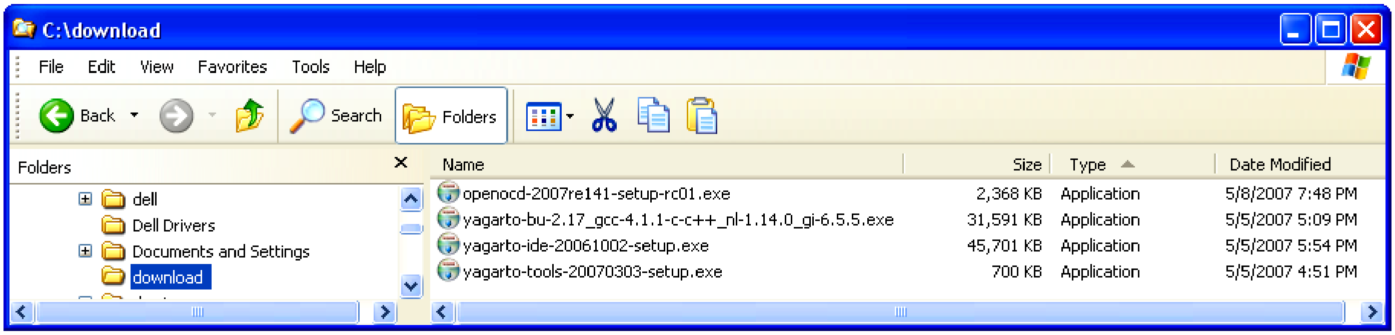
Download Complete

...rto-ide-20061002-setup.exe from www.yagarto.de

Downloaded: 44.6MB in 9 min 44 sec
Download to: C:\...\yagarto-ide-20061002-setup.exe
Transfer rate: 78.2KB/Sec

□ Close this dialog box when download completes

Now if you inspect the “c:\download” folder using Windows Explorer, you will see the four YAGARTO downloads. Each of these is an installer executable. We will double-click on each one in turn to install the various parts of our ARM Cross Development system.



Name	Size	Type	Date Modified
openocd-2007re141-setup-rc01.exe	2,368 KB	Application	5/8/2007 7:48 PM
yagarto-bu-2.17_gcc-4.1.1-c-c++_nl-1.14.0_gi-6.5.5.exe	31,591 KB	Application	5/5/2007 5:09 PM
yagarto-ide-20061002-setup.exe	45,701 KB	Application	5/5/2007 5:54 PM
yagarto-tools-20070303-setup.exe	700 KB	Application	5/5/2007 4:51 PM

There are four files in the c:\download folder:

openocd-2007re141-setup-rc01.exe	Installer for OpenOCD support
yagarto-tools-20070303-setup.exe	Installer for JLINK support
yagarto-bu-2.17_gcc-4.1.1-c-c++_nl-1.14.0_gi-6.5.5.exe	Installer for GNU compiler suite for ARM
yagarto-ide-20061002-setup.exe	Installer for Eclipse IDE

Note to Readers:

Michael Fischer is constantly improving the YAGARTO package. If you get a newer version when you download YAGARTO, rest assured that Michael has made sure that all the components work harmoniously together.

In this tutorial, the OpenOCD JTAG debugger system is stored in the following folder:

c**:\Program Files\openocd-2007re141\**

If Michael has posted a newer version, that folder name may change to:

c:\Program Files\openocd-2007re154\

For example, the OpenOCD executable and configuration files for this revision are stored in this folder: "**c:\Program Files\openocd-2007re141\bin**". Obviously, a newer revision will place these files is a different folder – you need to be aware of this if you download a newer version of YAGARTO.

We'll try to indicate throughout the tutorial those places where you will need to adjust the folder name to accommodate the new revision.

Downloading the Segger J-Link GDB Server

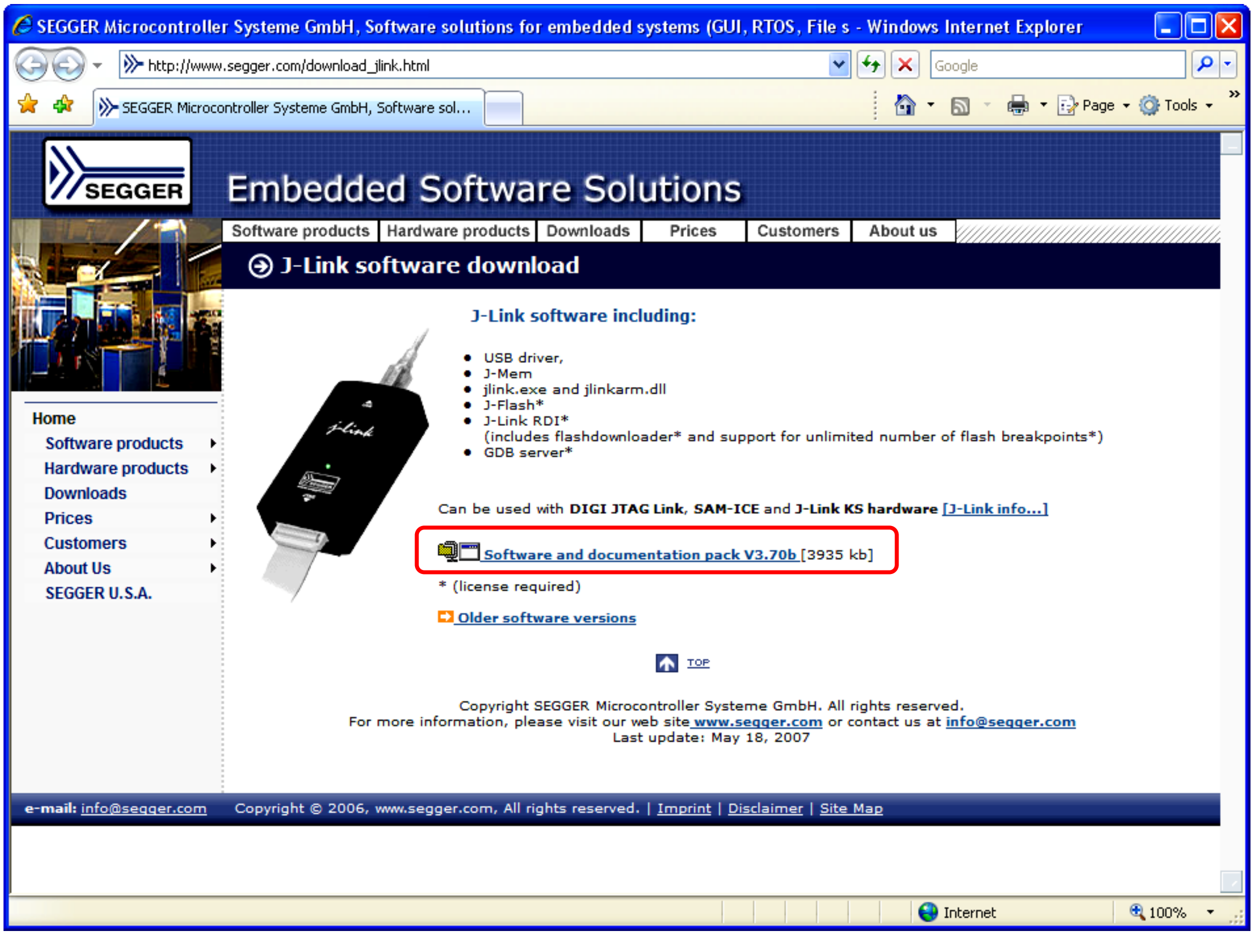
You may skip this section if you are planning to use the Olimex wiggler, the Olimex ARM-USB-OCD or the Amontec JTAGKey hardware debuggers.

If you have purchased the Atmel AT91SAM7S256-EK evaluation board, you may have also purchased a JTAG debugger called the SAM-ICE. In reality, this is a branded version of the Segger J-Link ARM Emulator that interfaces the Eclipse graphical debugger to the Atmel AT91SAM7S256 ARM chip's JTAG hardware interface. To use the SAM-ICE or the J-Link, we will need a Windows software program called the Segger J-Link GDB Server. The J-Link GDB Server can be downloaded from the following link:

http://www.segger.com/download_jlink.html

Note: there is a link to this on the Atmel www.at91.com web site but going directly to the Seeger web site guarantees access to the latest version.

This brings up the specific link to the J-Link ARM download. Click on "**Software and documentation pack V3.70b**" as shown below.



SEGGER Microcontroller Systeme GmbH, Software solutions for embedded systems (GUI, RTOS, File s - Windows Internet Explorer
http://www.segger.com/download_jlink.html
Google
SEGGER Microcontroller Systeme GmbH, Software sol...
Page
Tools
SEGGER
Embedded Software Solutions
Software products
Hardware products
Downloads
Prices
Customers
About us
Home
Software products
Hardware products
Downloads
Prices
Customers
About Us
SEGGER U.S.A.
Internet
100%

J-Link software download

J-Link software including:

- USB driver,
- J-Mem
- jlink.exe and jlinkarm.dll
- J-Flash*
- J-Link RDI*
  (includes flashdownloader* and support for unlimited number of flash breakpoints*)
- GDB server*

Can be used with **DIGI JTAG Link**, **SAM-ICE** and **J-Link KS hardware** [J-Link info...]

Software and documentation pack V3.70b [3935 kb]

* (license required)

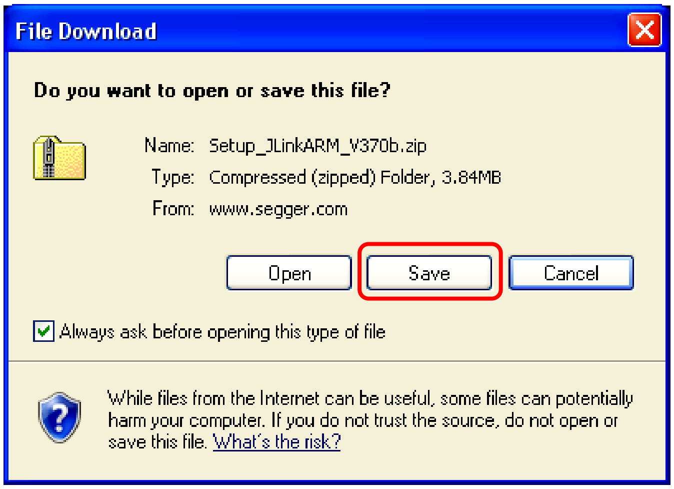
Older software versions

TOP

Copyright SEGGER Microcontroller Systeme GmbH. All rights reserved.
For more information, please visit our web site www.segger.com or contact us at info@segger.com
Last update: May 18, 2007

e-mail: info@segger.com Copyright © 2006, www.segger.com, All rights reserved. | Imprint | Disclaimer | Site Map

In the "File Download" screen shown below left, click "**Save**". Since we have a "c:\download" folder already set up, direct the Segger download to that folder as shown in the "Save As" screen shown below on the right. Click "**Save**" to start the download process.



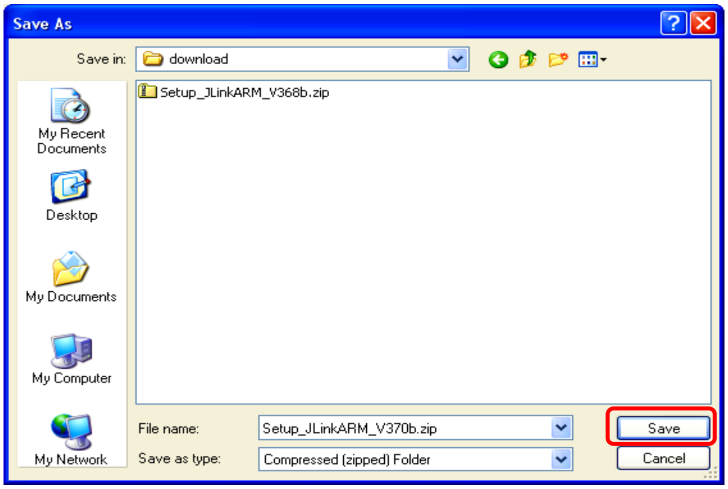
File Download

Do you want to open or save this file?

Name: Setup_JLinkARM_V370b.zip
Type: Compressed (zipped) Folder, 3.84MB
From: www.segger.com

☑ Always ask before opening this type of file

While files from the Internet can be useful, some files can potentially harm your computer. If you do not trust the source, do not open or save this file. What's the risk?



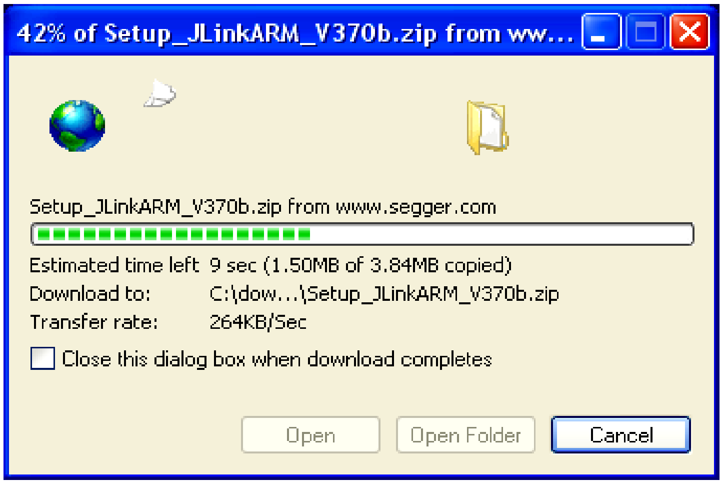
Save As

Save in: download

Setup_JLinkARM_V368b.zip

File name: Setup_JLinkARM_V370b.zip
Save as type: Compressed (zipped) Folder

The Segger J-Link download just takes a few seconds to download and leaves the Segger zip file in the "c:\download" folder. Click on "**Close**" when the download completes, as shown below right.

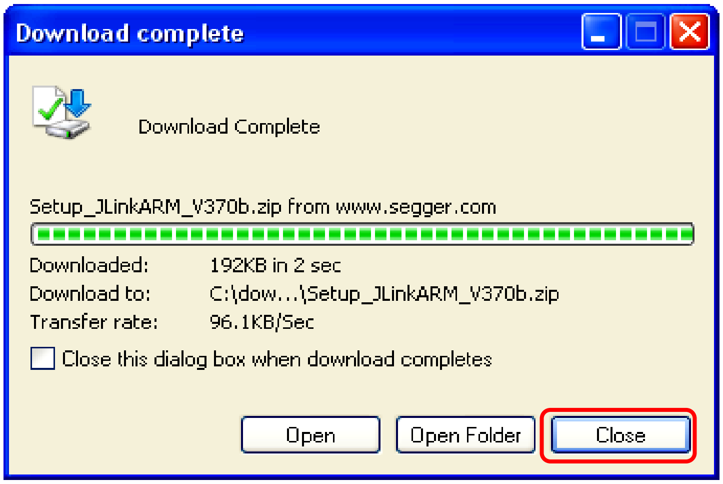


42% of Setup_JLinkARM_V370b.zip from ww...

Setup_JLinkARM_V370b.zip from www.segger.com

Estimated time left 9 sec (1.50MB of 3.84MB copied)
Download to: C:\dow...\Setup_JLinkARM_V370b.zip
Transfer rate: 264KB/Sec

☐ Close this dialog box when download completes



Download complete

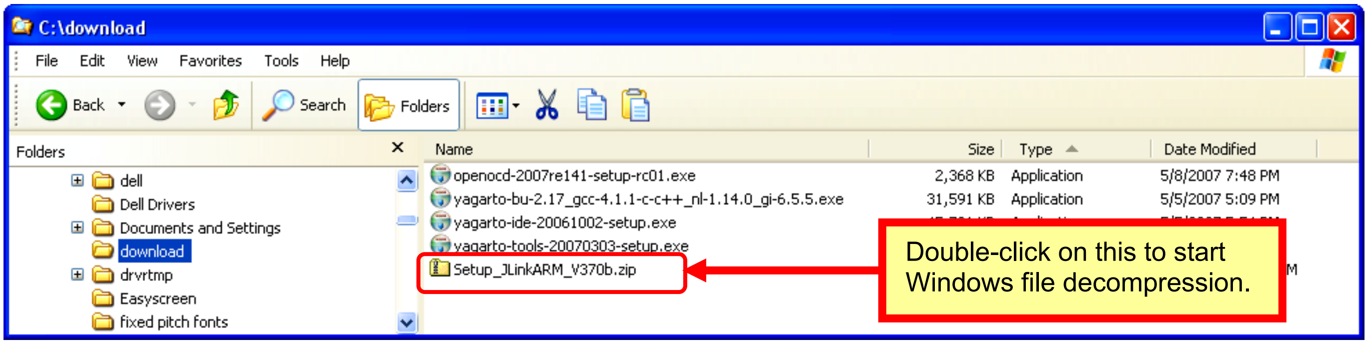
Download Complete

Setup_JLinkARM_V370b.zip from www.segger.com

Downloaded: 192KB in 2 sec
Download to: C:\dow...\Setup_JLinkARM_V370b.zip
Transfer rate: 96.1KB/Sec

☐ Close this dialog box when download completes

Double-click on the Segger zip file "**Setup_JLinkARM_V370b.zip**" as shown below and extract it to the c:\download folder using the standard Windows file decompression techniques.

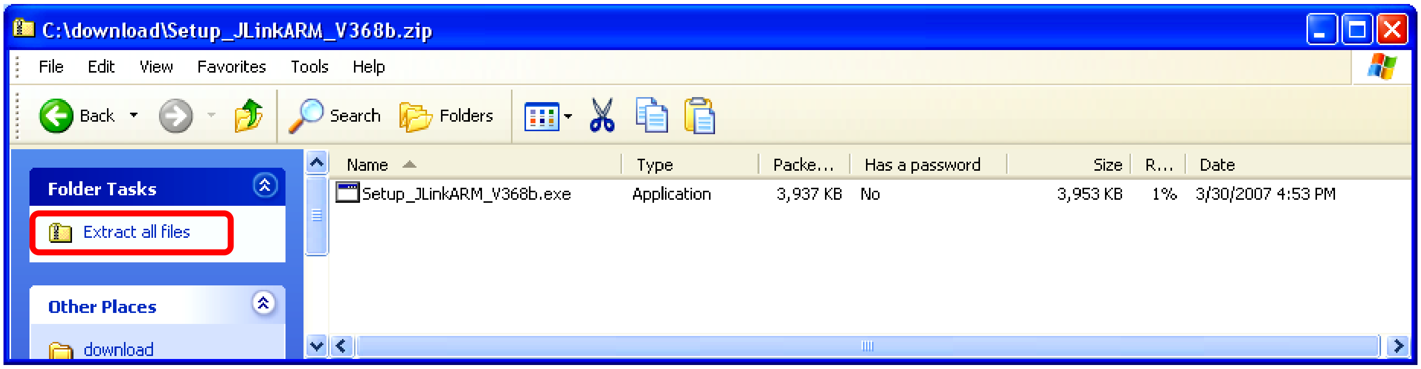


C:\download

Name	Size	Type	Date Modified
openocd-2007re141-setup-rc01.exe	2,368 KB	Application	5/8/2007 7:48 PM
yagarto-bu-2.17_gcc-4.1.1-c-c++_nl-1.14.0_gi-6.5.5.exe	31,591 KB	Application	5/5/2007 5:09 PM
yagarto-ide-20061002-setup.exe			
yagarto-tools-20070303-setup.exe			
Setup_JLinkARM_V370b.zip			

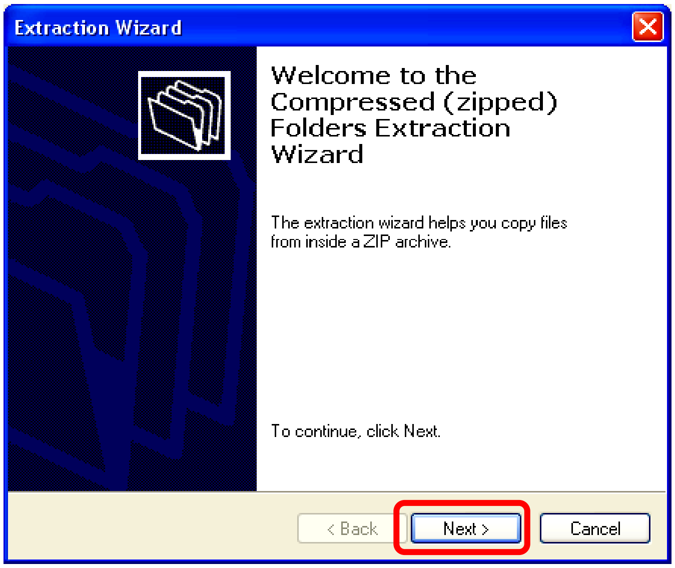
Double-click on this to start Windows file decompression.

The Windows file decompression wizard will start up; Click on "**Extract all files**" to start the decompression.



Name	Type	Packe...	Has a password	Size	R...	Date
Setup_JLinkARM_V368b.exe	Application	3,937 KB	No	3,953 KB	1%	3/30/2007 4:53 PM

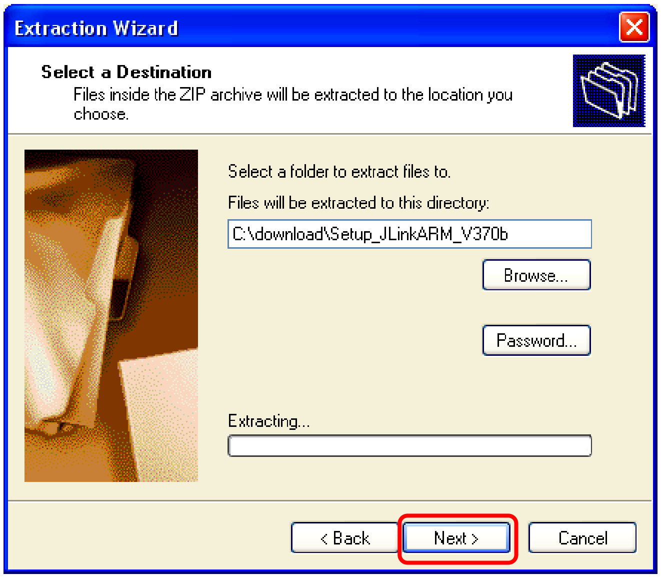
Click on "**Next**" on the "Welcome" screen on the left below. For the destination of the extracted file, take the default which will place it as a sub-folder of the c:\download directory. Click "**Next**" on the screen on the right below to actually start the file extraction process.



Welcome to the Compressed (zipped) Folders Extraction Wizard

The extraction wizard helps you copy files from inside a ZIP archive.

To continue, click Next.



Select a Destination
Files inside the ZIP archive will be extracted to the location you choose.

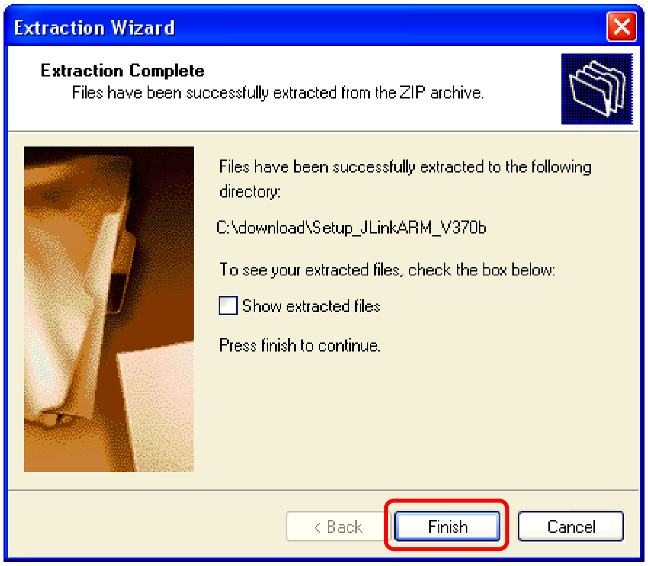
Select a folder to extract files to.

Files will be extracted to this directory:

C:\download\Setup_JLinkARM_V370b

Extracting...

Click on "**Finish**" below to complete the Windows file extraction operation.



Extraction Complete
Files have been successfully extracted from the ZIP archive.

Files have been successfully extracted to the following directory:

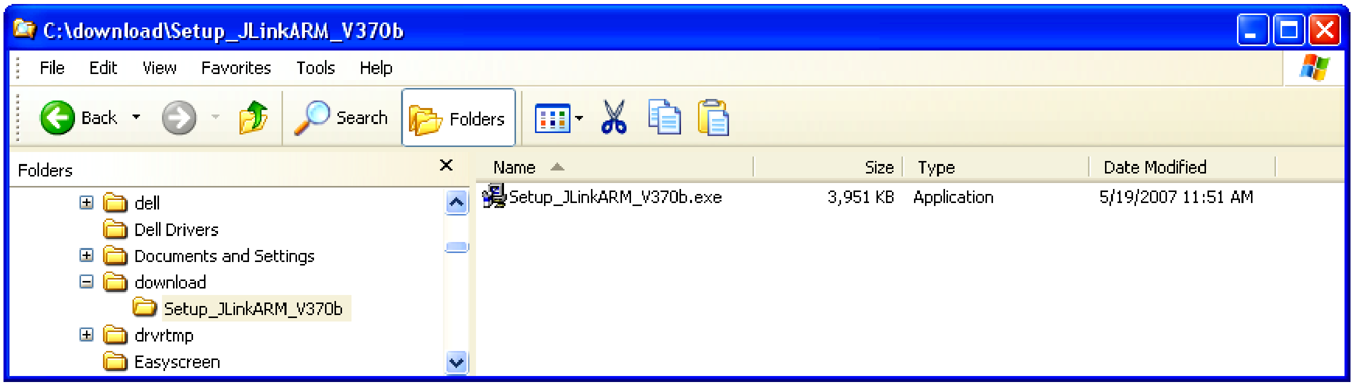
C:\download\Setup_JLinkARM_V370b

To see your extracted files, check the box below:

□ Show extracted files

Press finish to continue.

If you navigate down into the Segger sub-folder in the c:\download directory, you will see the Segger J-Link package installer. This is the application “**Setup_JLinkARM_V370b.exe**” as shown below. We will be installing the J-Link GDB Server later, assuming that you have the SAM-ICE JTAG hardware.



Name	Size	Type	Date Modified
Setup_JLinkARM_V370b.exe	3,951 KB	Application	5/19/2007 11:51 AM

Don’t be alarmed if the Segger web site shows a more recent revision of the Segger J-Link GDB Server, it is always prudent to use the latest and greatest version available!

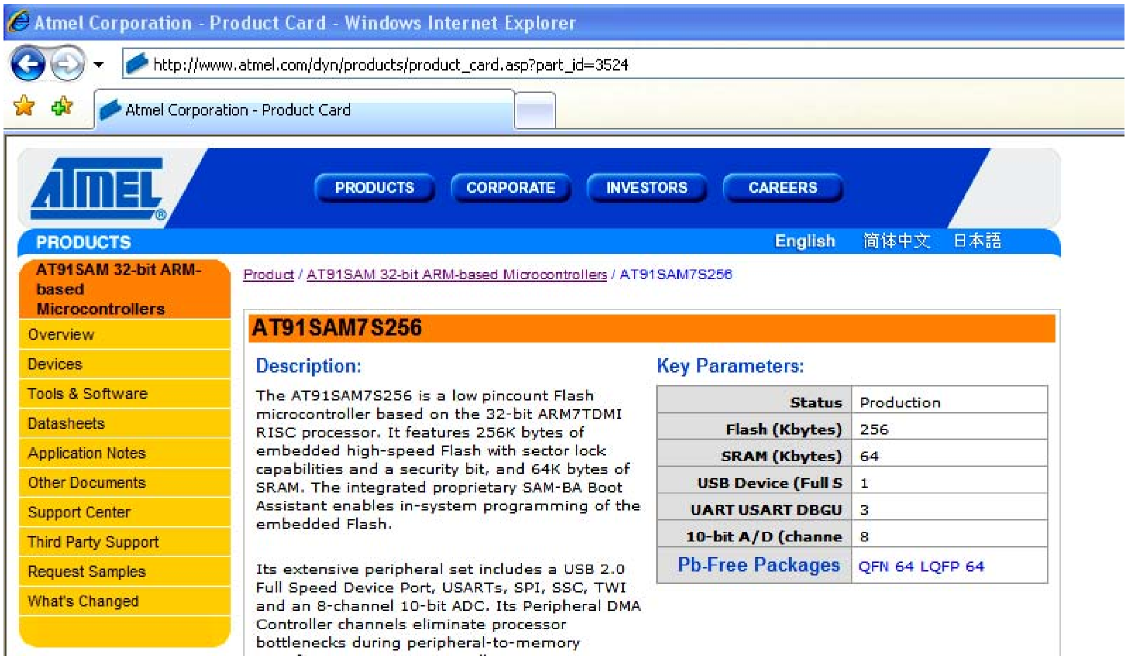
Downloading the Atmel SAM-BA Boot Assistant

Atmel provides a very nice Windows utility called the SAM Boot Assistant (SAM-BA) which can be used to program the onchip FLASH memory. SAM-BA can operate over the COM port with a standard RS-232 straight-through cable and also operate over the USB port if you have a standard USB cable. It can also connect via the USB port to the JTAG port if you have the SAM-ICE JTAG hardware interface. You cannot use the SAM-BA with the Olimex ARM-USB-OCD or the Amontec JTAGKey JTAG hardware interfaces; for those the OpenOCD software can be utilized to program the FLASH and debug the application. In any event, it makes good sense to have this handy utility available on your Eclipse cross development system.

To download SAM-BA, click on the following link:

http://www.atmel.com/dyn/products/product_card.asp?part_id=3524

The Atmel main web site for the AT91 family will appear as shown below.



Atmel Corporation - Product Card - Windows Internet Explorer
http://www.atmel.com/dyn/products/product_card.asp?part_id=3524
Atmel Corporation - Product Card
PRODUCTS
CORPORATE
INVESTORS
CAREERS
PRODUCTS
English
简体中文
日本語
AT91SAM 32-bit ARM-based Microcontrollers
Overview
Devices
Tools & Software
Datasheets
Application Notes
Other Documents
Support Center
Third Party Support
Request Samples
What's Changed
Product / AT91SAM 32-bit ARM-based Microcontrollers / AT91SAM7S256

AT91SAM7S256

Description:

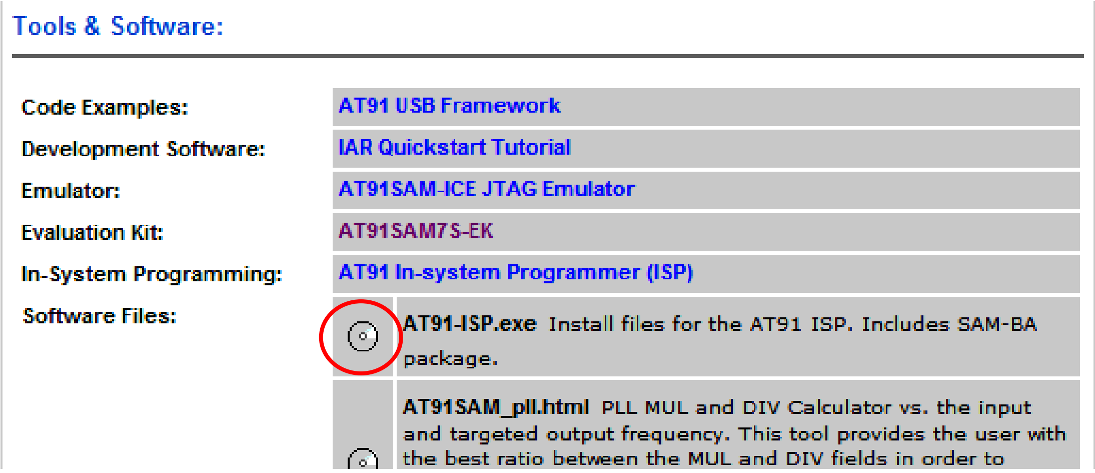
The AT91SAM7S256 is a low pincount Flash microcontroller based on the 32-bit ARM7TDMI RISC processor. It features 256K bytes of embedded high-speed Flash with sector lock capabilities and a security bit, and 64K bytes of SRAM. The integrated proprietary SAM-BA Boot Assistant enables in-system programming of the embedded Flash.

Its extensive peripheral set includes a USB 2.0 Full Speed Device Port, USARTs, SPI, SSC, TWI and an 8-channel 10-bit ADC. Its Peripheral DMA Controller channels eliminate processor bottlenecks during peripheral-to-memory

Key Parameters:

Status	Production
Flash (Kbytes)	256
SRAM (Kbytes)	64
USB Device (Full S	1
UART USART DBGU	3
10-bit A/D (channe	8
Pb-Free Packages	QFN 64 LQFP 64

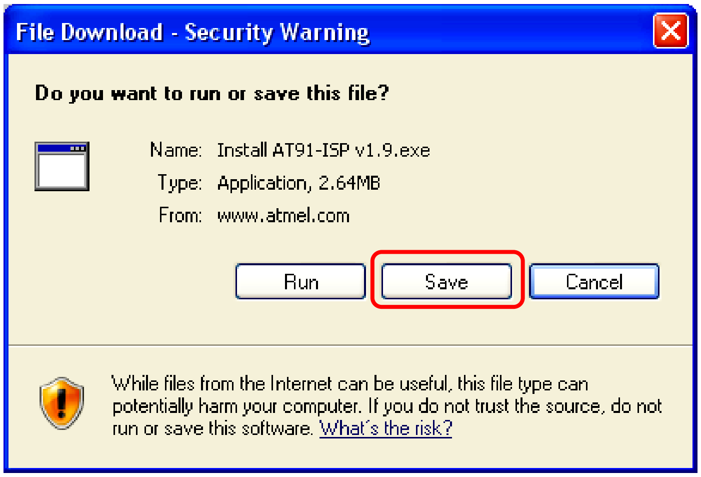
Scroll down until you see under "Tools and Software" the file "AT91-ISP.exe". Click on the CD-ROM symbol to start the download.



Tools & Software:

Code Examples:	AT91 USB Framework
Development Software:	IAR Quickstart Tutorial
Emulator:	AT91SAM-ICE JTAG Emulator
Evaluation Kit:	AT91SAM7S-EK
In-System Programming:	AT91 In-system Programmer (ISP)
Software Files:	AT91-ISP.exe Install files for the AT91 ISP. Includes SAM-BA package.
	AT91SAM_pll.html PLL MUL and DIV Calculator vs. the input and targeted output frequency. This tool provides the user with the best ratio between the MUL and DIV fields in order to

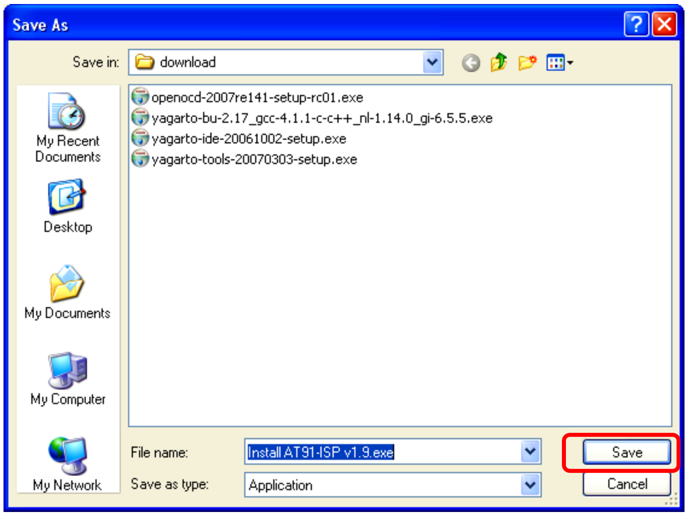
In the “File Download” window below left, click on “**Save**”. Select our “**c:\download**” folder as the destination and the click “**Save**” to start the download, as shown below right.


File Download - Security Warning

Do you want to run or save this file?

Name: Install AT91-ISP v1.9.exe
Type: Application, 2.64MB
From: www.atmel.com

While files from the Internet can be useful, this file type can potentially harm your computer. If you do not trust the source, do not run or save this software. What's the risk?

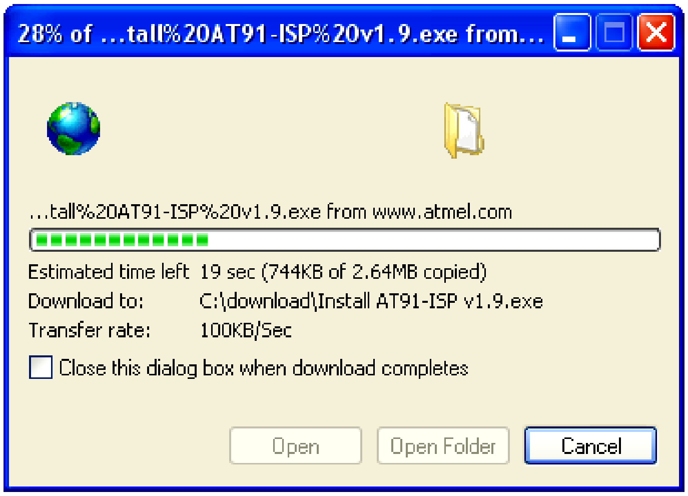

Save As

Save in: download

openocd-2007re141-setup-rc01.exe
yagarto-bu-2.17_gcc-4.1.1-c-c++_nl-1.14.0_gi-6.5.5.exe
yagarto-ide-20061002-setup.exe
yagarto-tools-20070303-setup.exe

File name: Install AT91-ISP v1.9.exe
Save as type: Application

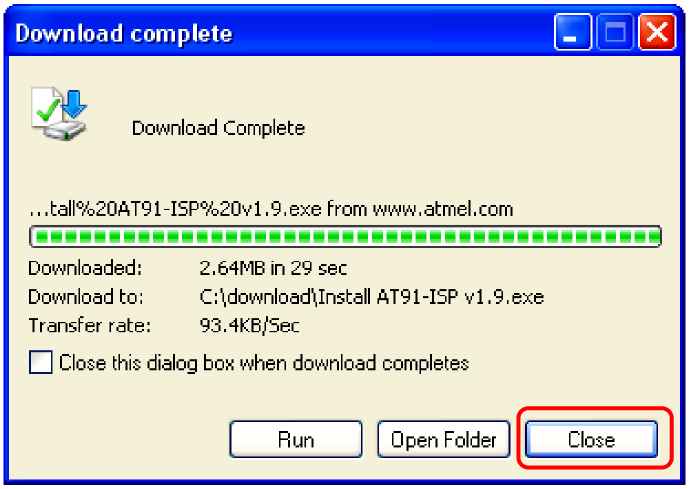
The SAM-BA installer will download in a few seconds. Click “**Close**” when the download completes.


28% of ...tall%20AT91-ISP%20v1.9.exe from...

...tall%20AT91-ISP%20v1.9.exe from www.atmel.com

Estimated time left 19 sec (744KB of 2.64MB copied)
Download to: C:\download\Install AT91-ISP v1.9.exe
Transfer rate: 100KB/Sec

□ Close this dialog box when download completes


Download complete

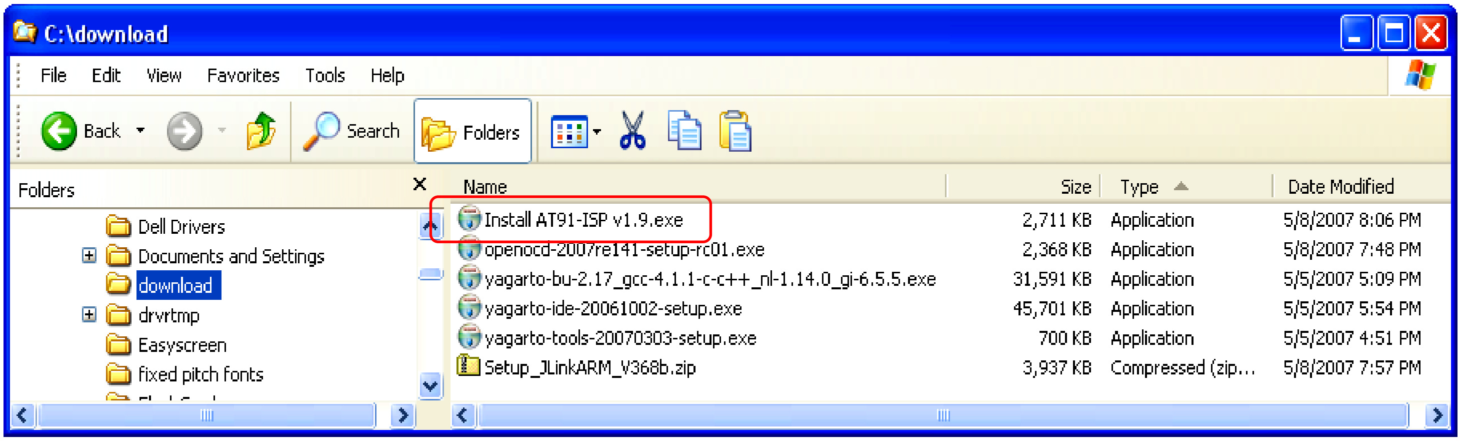
Download Complete

...tall%20AT91-ISP%20v1.9.exe from www.atmel.com

Downloaded: 2.64MB in 29 sec
Download to: C:\download\Install AT91-ISP v1.9.exe
Transfer rate: 93.4KB/Sec

□ Close this dialog box when download completes

The “c:\download” folder should now show the Atmel SAM-BA installer, called “Install AT91-ISP v1.9.exe”.


C:\download

Dell Drivers
Documents and Settings
download
drvrtmp
Easyscreen
fixed pitch fonts

Name	Size	Type	Date Modified
Install AT91-ISP v1.9.exe	2,711 KB	Application	5/8/2007 8:06 PM
openocd-2007re141-setup-rc01.exe	2,368 KB	Application	5/8/2007 7:48 PM
yagarto-bu-2.17_gcc-4.1.1-c-c++_nl-1.14.0_gi-6.5.5.exe	31,591 KB	Application	5/5/2007 5:09 PM
yagarto-ide-20061002-setup.exe	45,701 KB	Application	5/5/2007 5:54 PM
yagarto-tools-20070303-setup.exe	700 KB	Application	5/5/2007 4:51 PM
Setup_JLinkARM_V368b.zip	3,937 KB	Compressed (zip...	5/8/2007 7:57 PM

Install All Tools

Everything we need has been downloaded into the “**c:\download**” folder. Now we will install each tool individually. Michael Fischer has made everything simple, in most cases just take the defaults presented by the installers!

Install OpenOCD

Even if you are planning to use the Atmel SAM-ICE JTAG hardware debugger, we will install OpenOCD anyway because it contains the executable for the GNU make utility. Michael Fischer’s OpenOCD installer automatically places the location of the “make.exe” executable into the Windows path environment variable, making it easy for Eclipse to find it when you hit the “Build All” button.

Eclipse/CDT has a fabulous graphical source code debugger that is built on top of the venerable GNU GDB command line debugger. The only problem is how to connect it to a remote target such as a microprocessor circuit board. GDB communicates to the target via a Remote Serial Protocol that can be utilized over a parallel port or an internet port. To make the Eclipse/JTAG connection, we need a daemon (a program that runs in the background), waiting for GDB Remote Serial Protocol commands coming over the TCP port and then manipulate the AT91SAM7 microprocessor JTAG pins according to the JTAG protocol established by ARM.

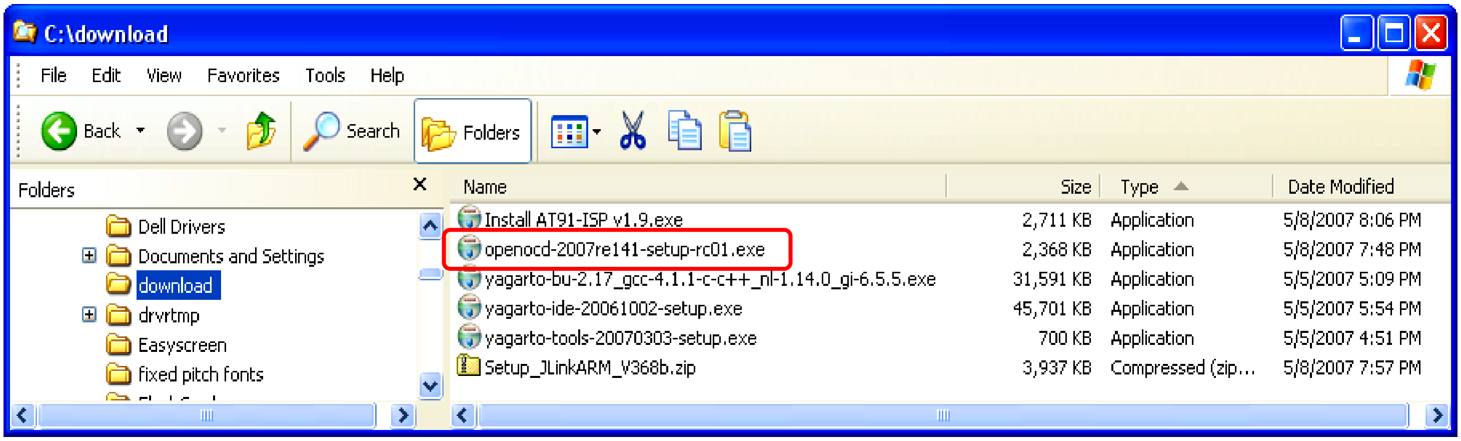
In the past, most people have used the Macraigor OCDRemote utility that reads GDB serial commands and manipulates the ARM JTAG lines using the PC’s parallel port and a simple hardware level-shifting device called a “wiggler”. The Macraigor OCDRemote utility has always been available for free (in binary form) but it is not open source. Macraigor could withdraw it at any time.

To the rescue is German college student Dominic Rath who developed an open source ARM JTAG debugger as his diploma thesis at the University of Applied Sciences, FH-Augsburg in Bavaria. Dominic’s thesis can be found here: http://openocd.berlios.de/thesis.pdf . Dominic also has a website on the Berlios Open Source repository here: http://openocd.berlios.de/web/

Finally, Dominic participates in the OpenOCD message board at the SparkFun site here: http://www.sparkfun.com

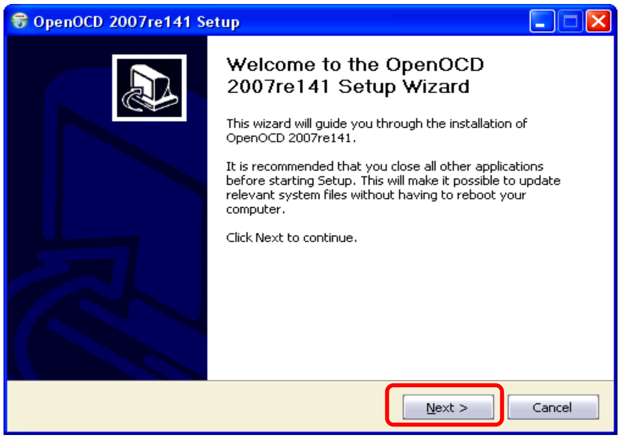
OpenOCD can be used with the inexpensive “wiggler” JTAG device as well as the USB JTAG devices such as the Amontec JTAGKey, the Olimex ARM-USB-OCD and others coming on the market. It cannot be used with the SAM-ICE JTAG interface.

Double-click on the file “**Openocd-2007re131-setup-rc01.exe”** to start the OpenOCD installer.



Name	Size	Type	Date Modified
Install AT91-ISP v1.9.exe	2,711 KB	Application	5/8/2007 8:06 PM
openocd-2007re141-setup-rc01.exe	2,368 KB	Application	5/8/2007 7:48 PM
yagarto-bu-2.17_gcc-4.1.1-c-c++_nl-1.14.0_gi-6.5.5.exe	31,591 KB	Application	5/5/2007 5:09 PM
yagarto-ide-20061002-setup.exe	45,701 KB	Application	5/5/2007 5:54 PM
yagarto-tools-20070303-setup.exe	700 KB	Application	5/5/2007 4:51 PM
Setup_JLinkARM_V368b.zip	3,937 KB	Compressed (zip...	5/8/2007 7:57 PM

In the “Welcome**”** screen below on the left, click the “**Next**” button. The next screen is a standard GNU license agreement; click the top radio button to accept the License Agreement and click the “**Next**” button to continue.



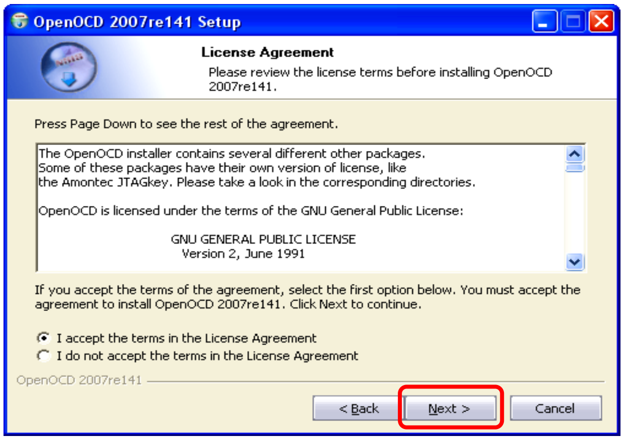
OpenOCD 2007re141 Setup

Welcome to the OpenOCD 2007re141 Setup Wizard

This wizard will guide you through the installation of OpenOCD 2007re141.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.



OpenOCD 2007re141 Setup

License Agreement

Please review the license terms before installing OpenOCD 2007re141.

Press Page Down to see the rest of the agreement.

The OpenOCD installer contains several different other packages. Some of these packages have their own version of license, like the Amontec JTAGkey. Please take a look in the corresponding directories.

OpenOCD is licensed under the terms of the GNU General Public License:

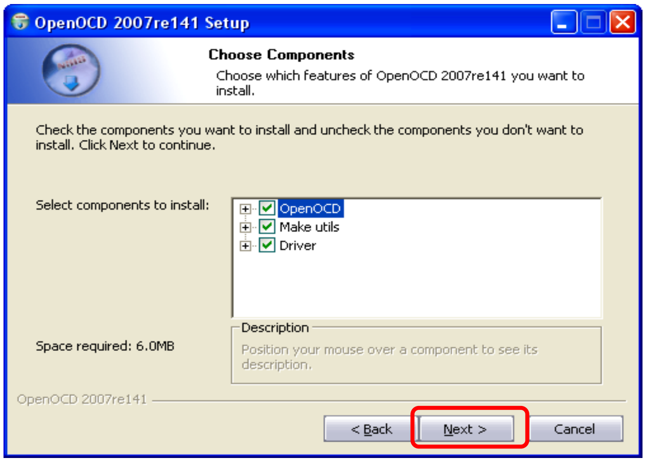
GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

If you accept the terms of the agreement, select the first option below. You must accept the agreement to install OpenOCD 2007re141. Click Next to continue.

● I accept the terms in the License Agreement
○ I do not accept the terms in the License Agreement

OpenOCD 2007re141

In the “Choose Components” screen shown below on the left, select all three components (**OpenOCD, Make Utils and Driver**). Click “**Next**” to continue. On the “Choose Install Location” screen below on the right, take the default location “**c:\Program Files\openocd-2007re141**” and click “**Next**” to continue.



OpenOCD 2007re141 Setup

Choose Components

Choose which features of OpenOCD 2007re141 you want to install.

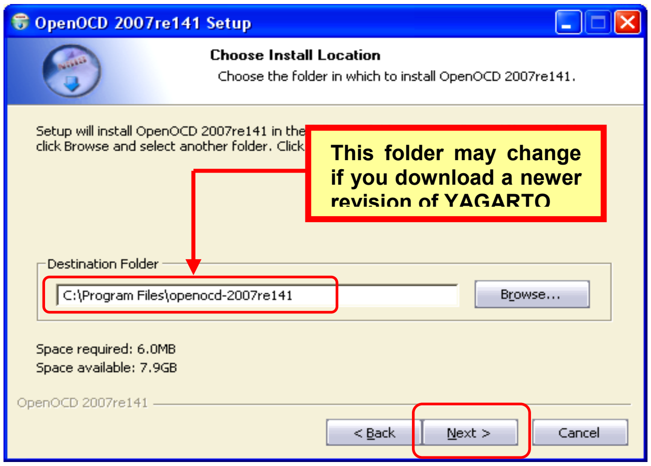
Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select components to install:
☑ OpenOCD
☑ Make utils
☑ Driver

Description
Position your mouse over a component to see its description.

Space required: 6.0MB

OpenOCD 2007re141



OpenOCD 2007re141 Setup

Choose Install Location

Choose the folder in which to install OpenOCD 2007re141.

Setup will install OpenOCD 2007re141 in the
click Browse and select another folder. Click

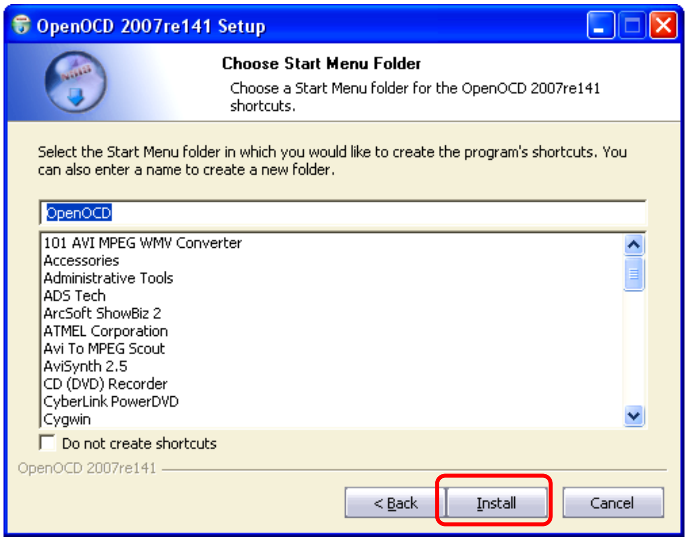
This folder may change if you download a newer revision of YAGARTO

Destination Folder
C:\Program Files\openocd-2007re141

Space required: 6.0MB
Space available: 7.9GB

OpenOCD 2007re141

Take the default in the “Choose Start Menu Folder” screen shown below left. The OpenOCD debugger will be normally called from within Eclipse, so execution from the Start menu would be rare. You could click the checkbox “Do not Create Shortcuts” if desired. Click “**Install**” to take the default and continue.



OpenOCD 2007re141 Setup

Choose Start Menu Folder

Choose a Start Menu folder for the OpenOCD 2007re141 shortcuts.

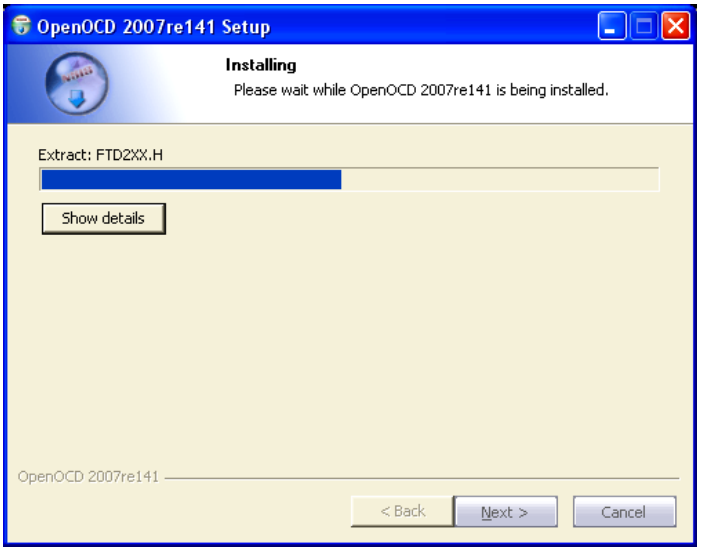
Select the Start Menu folder in which you would like to create the program's shortcuts. You can also enter a name to create a new folder.

OpenOCD

101 AVI MPEG WMV Converter
Accessories
Administrative Tools
ADS Tech
ArcSoft ShowBiz 2
ATMEL Corporation
Avi To MPEG Scout
AviSynth 2.5
CD (DVD) Recorder
CyberLink PowerDVD
Cygwin

□ Do not create shortcuts

OpenOCD 2007re141



OpenOCD 2007re141 Setup

Installing

Please wait while OpenOCD 2007re141 is being installed.

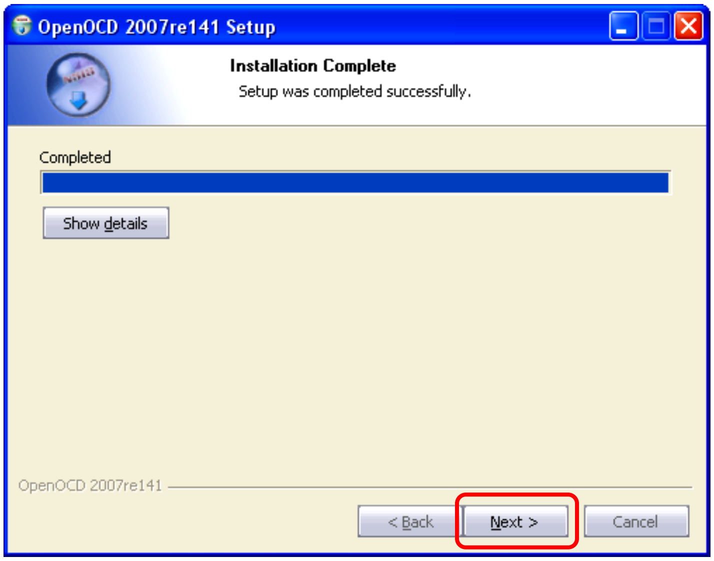
Extract: FTD2XX.H

OpenOCD 2007re141

OpenOCD installs very fast (less than a minute) as shown in the "Installing" screen above on the right.

Click "**Next**" when the installation completes, as shown in the screen below on the left.

Click "**Finish**", as shown on the screen below to the right, to terminate the OpenOCD installer.

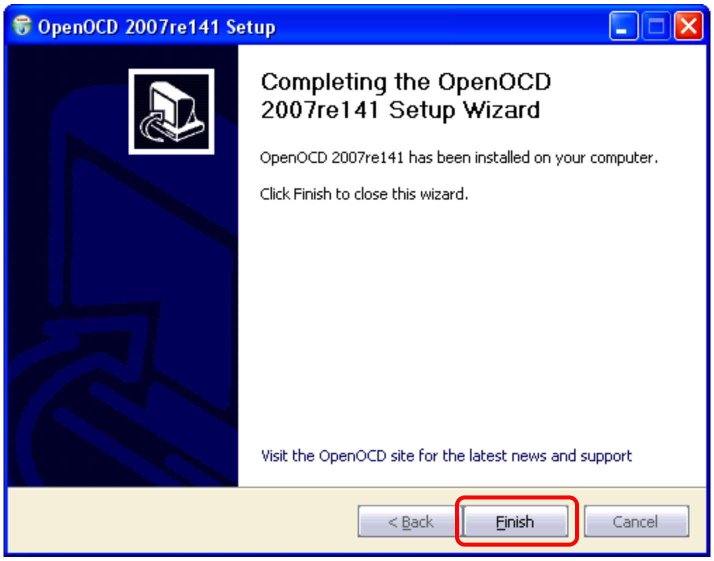


OpenOCD 2007re141 Setup

Installation Complete

Setup was completed successfully.

Completed



OpenOCD 2007re141 Setup

Completing the OpenOCD 2007re141 Setup Wizard

OpenOCD 2007re141 has been installed on your computer.

Click Finish to close this wizard.

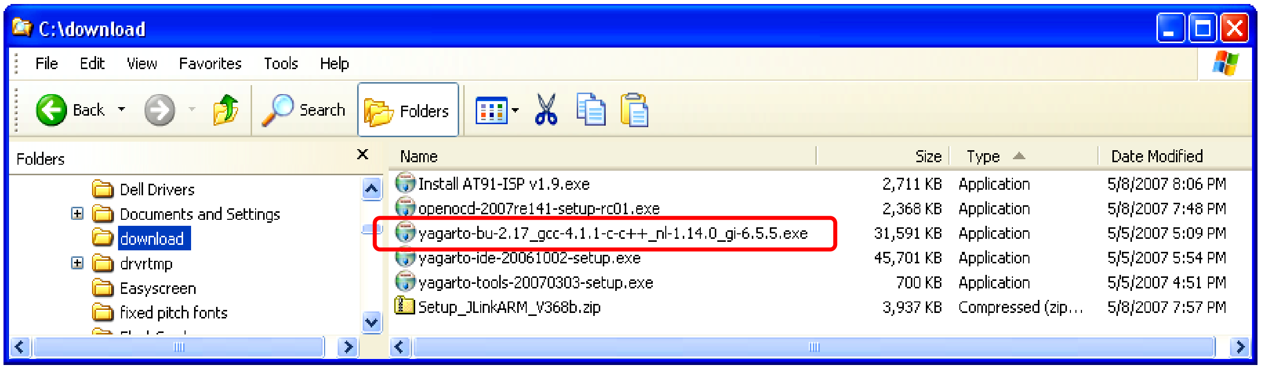
Visit the OpenOCD site for the latest news and support

Make a mental note that the installer has placed all OpenOCD components in the following folder: **c:\Program\Files\openocd-2007re141\.** If your download includes a more recent revision of OpenOCD, remember the folder address – we will use it later in the tutorial.

Install YAGARTO Tool Chain

There are a number of pre-built GNU ARM compiler toolsets available on the web and they are all very good. For this tutorial, we will be using the **YAGARTO** pre-built ARM compiler tool suite developed by Michael Fischer of Lohfelden, Germany. Michael's version of the GNU compiler toolset for ARM has been natively compiled for the Intel/Windows platform; therefore the Cygwin utilities are not needed. This makes the compiler run faster and simplifies the installation. Michael has also performed some tweaks on the included GNU GDB debugger to make it perform better in the Eclipse environment.

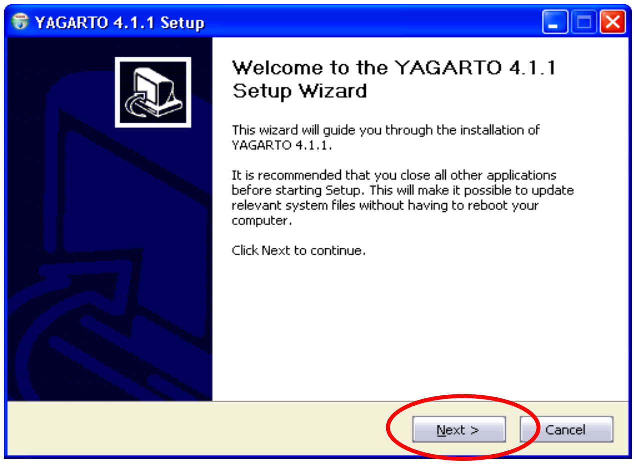
Double-click on the file **Yagarto-bu-2.17_gcc-4.1.1-c-c++_nl-1.14.0_gi-6.5.5.exe** to start the YAGARTO tool chain installer.



C:\download

Name	Size	Type	Date Modified
Install AT91-ISP v1.9.exe	2,711 KB	Application	5/8/2007 8:06 PM
openocd-2007re141-setup-rc01.exe	2,368 KB	Application	5/8/2007 7:48 PM
yagarto-bu-2.17_gcc-4.1.1-c-c++_nl-1.14.0_gi-6.5.5.exe	31,591 KB	Application	5/5/2007 5:09 PM
yagarto-ide-20061002-setup.exe	45,701 KB	Application	5/5/2007 5:54 PM
yagarto-tools-20070303-setup.exe	700 KB	Application	5/5/2007 4:51 PM
Setup_JLinkARM_V368b.zip	3,937 KB	Compressed (zip...	5/8/2007 7:57 PM

In the “Welcome” screen shown on the left below, click on “**Next**” to continue. Click the ”**I Accept …**” radio button on the “License Agreement” screen below on the right and then click “**Next**” to continue.

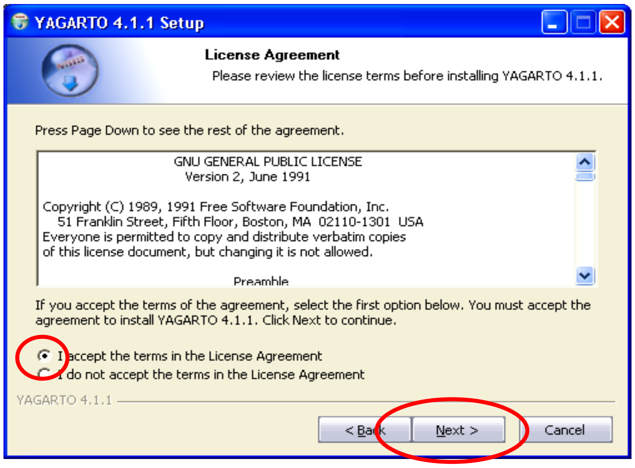


Welcome to the YAGARTO 4.1.1 Setup Wizard

This wizard will guide you through the installation of YAGARTO 4.1.1.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.



License Agreement

Please review the license terms before installing YAGARTO 4.1.1.

Press Page Down to see the rest of the agreement.

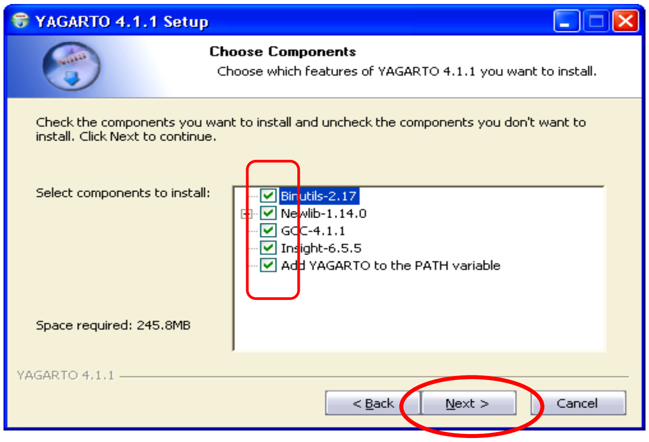
```
                GNU GENERAL PUBLIC LICENSE
                   Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
   51 Franklin Street, Fifth Floor, Boston, MA  02110-1301  USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

                        Preamble
```

If you accept the terms of the agreement, select the first option below. You must accept the agreement to install YAGARTO 4.1.1. Click Next to continue.

● I accept the terms in the License Agreement

○ I do not accept the terms in the License Agreement

In the “Choose Components” screen on the left below, take all the defaults by simply clicking “**Next**” to continue. Note that this installs the Insight debugger that we will not use, but no harm is done including it. On the “Choose Install Location” screen on the right below, take the default again by clicking “**Next**” to continue.



Choose Components

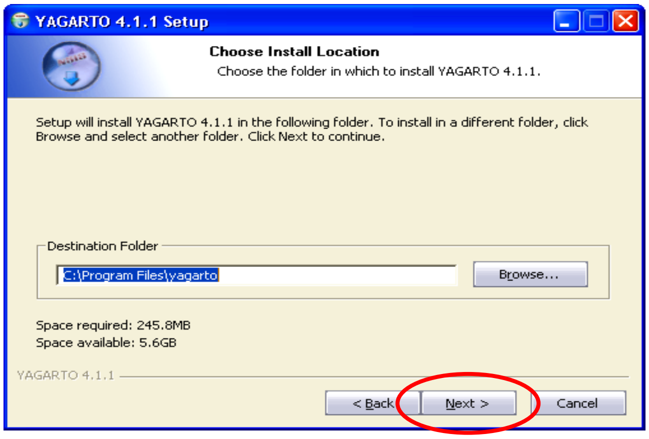
Choose which features of YAGARTO 4.1.1 you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select components to install:

☑ Binutils-2.17
☑ Newlib-1.14.0
☑ GCC-4.1.1
☑ Insight-6.5.5
☑ Add YAGARTO to the PATH variable

Space required: 245.8MB



Choose Install Location

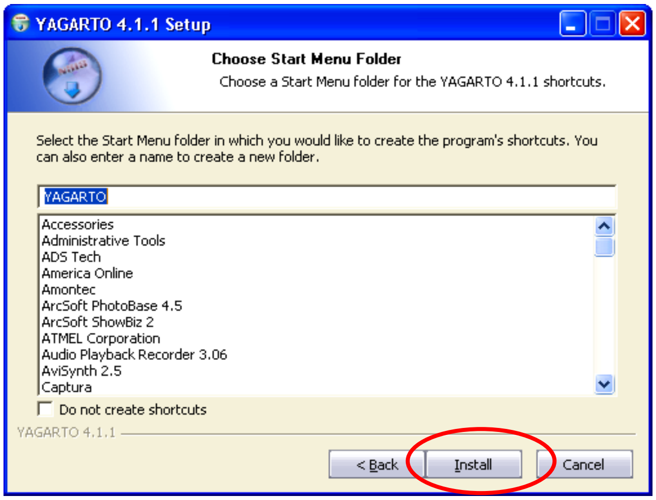
Choose the folder in which to install YAGARTO 4.1.1.

Setup will install YAGARTO 4.1.1 in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder: C:\Program Files\yagarto

Space required: 245.8MB
Space available: 5.6GB

Click “**Install**” on the “Choose Start Menu Folder” shown below left and the YAGARTO tool chain installer will commence. This installation takes several minutes.



Choose Start Menu Folder

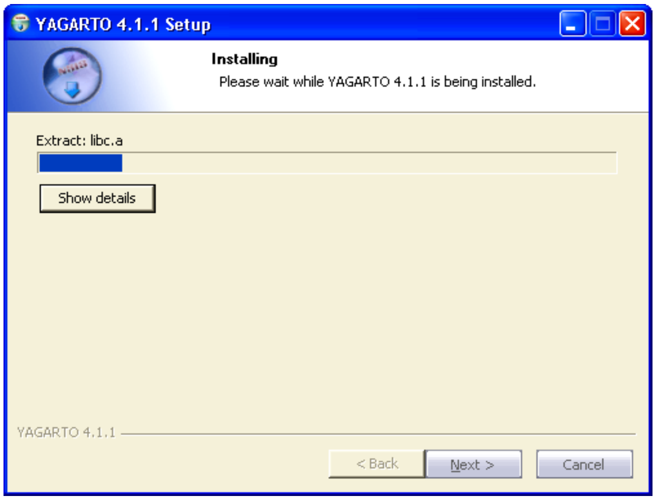
Choose a Start Menu folder for the YAGARTO 4.1.1 shortcuts.

Select the Start Menu folder in which you would like to create the program's shortcuts. You can also enter a name to create a new folder.

YAGARTO

Accessories
Administrative Tools
ADS Tech
America Online
Amontec
ArcSoft PhotoBase 4.5
ArcSoft ShowBiz 2
ATMEL Corporation
Audio Playback Recorder 3.06
AviSynth 2.5
Captura

□ Do not create shortcuts



Installing

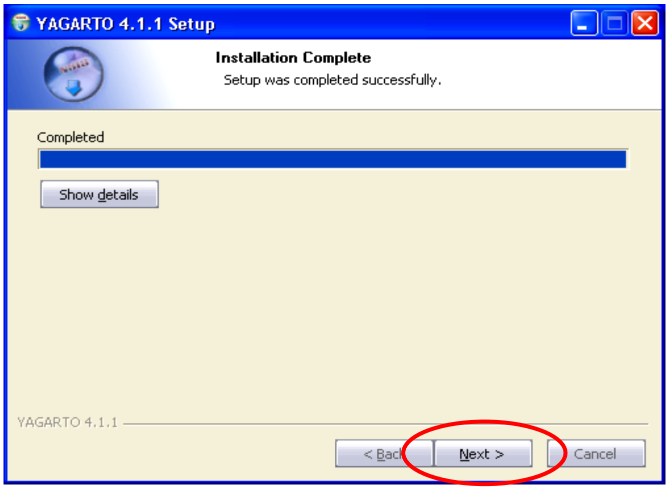
Please wait while YAGARTO 4.1.1 is being installed.

Extract: libc.a

When tool chain installation completes, click “**Next**” as shown below on the left followed by clicking “**Finish**” on YAGARTO completion screen shown on the right below. This will terminate the YAGARTO installer.

Make a mental note that the YAGARTO compiler tool chain is installed in the following folder:

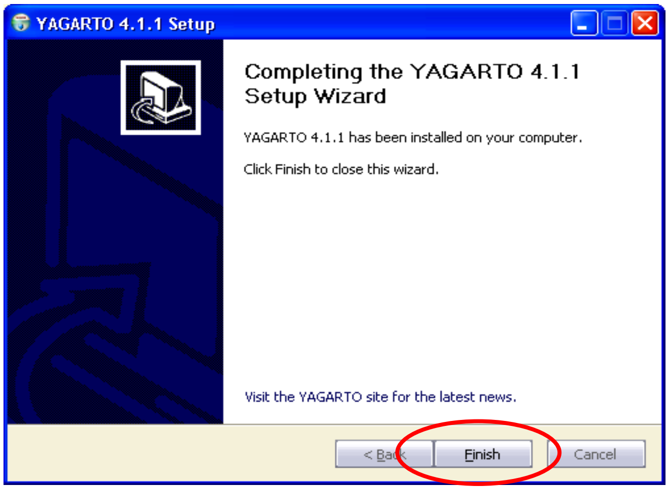
C:\Program Files\Yagarto\



Installation Complete

Setup was completed successfully.

Completed



Completing the YAGARTO 4.1.1 Setup Wizard

YAGARTO 4.1.1 has been installed on your computer.

Click Finish to close this wizard.

Visit the YAGARTO site for the latest news.

Install Eclipse IDE

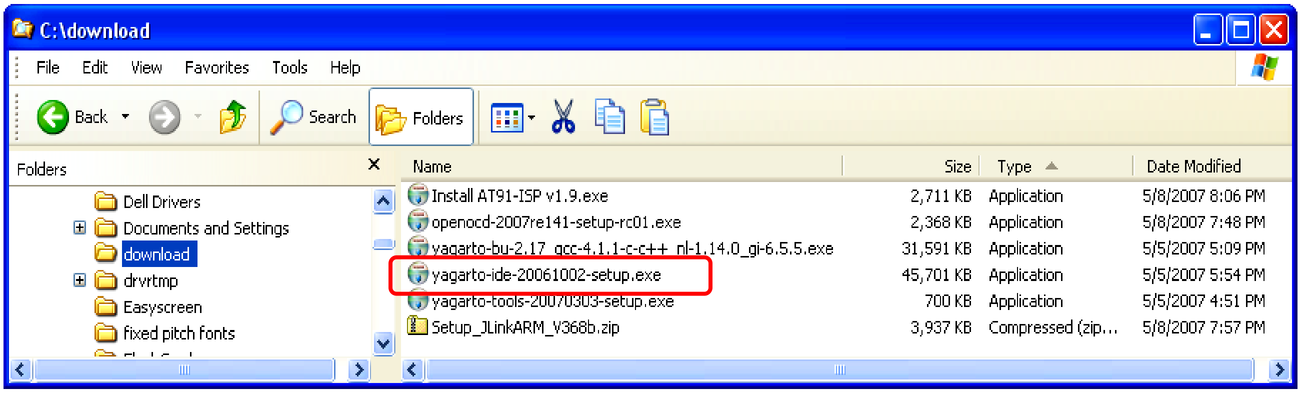
IBM has been a competitor in recent years to Microsoft and at one time was building an alternative to Microsoft's Visual Studio (specifically for the purpose of developing JAVA software). This effort was called the Eclipse Project and in 2004 IBM donated Eclipse to the Open Software movement, created an independent Eclipse Foundation to support it and invited programmers worldwide to contribute to it. The result has been an avalanche of activity that has catapulted Eclipse from a simple JAVA editor to a multi-platform tool for developing just about any language, including C/C++ projects.

Eclipse by itself makes a wonderful Integrated Development Environment (IDE) for JAVA software. There are numerous books available on the Eclipse JAVA platform and many PC and Web applications are being built with it. Be sure to visit the Eclipse web site: **www.eclipse.org**

Our purpose is to build an IDE for embedded software development; this normally implies C/C++ programming. To do this, we need to install the CDT (**C** **D**evelopment **T**oolkit) plug-in. The problem is that Eclipse/CDT has had difficulties working with remote debuggers. Oyvind Harboe and the Norwegian company Zylin has developed, with the cooperation of the CDT team, a custom version of the CDT plug-in that solves these problems. The Zylin version of CDT properly starts the remote debugger in idle mode so you can start execution, single-step, etc.

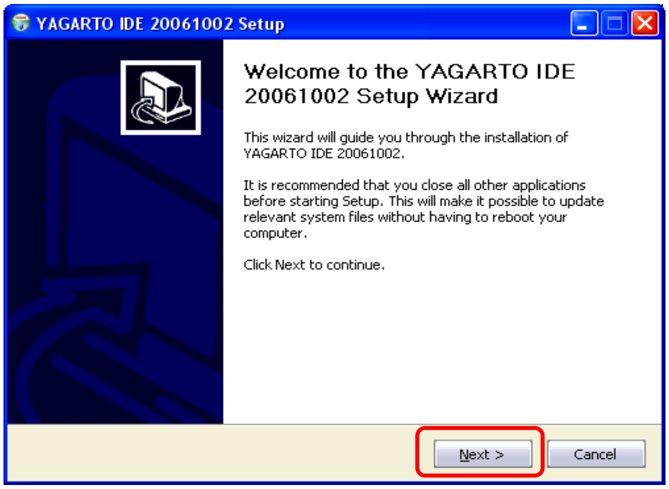
The only proviso is that we must select a version of Eclipse compatible with the Zylin CDT plug-in. Rest assured that the Zylin CDT included in the YAGARTO download was chosen for its compatibility with the new Eclipse 3.2 release. The Zylin website is at this address: **www.zylin.com**

Double-click on the file "**Yagarto-ide-20061002-setup.exe"** to start the Eclipse IDE installer.



Name	Size	Type	Date Modified
Install AT91-ISP v1.9.exe	2,711 KB	Application	5/8/2007 8:06 PM
openocd-2007re141-setup-rc01.exe	2,368 KB	Application	5/8/2007 7:48 PM
yagarto-bu-2.17_gcc-4.1.1-c-c++_nl-1.14.0_gi-6.5.5.exe	31,591 KB	Application	5/5/2007 5:09 PM
yagarto-ide-20061002-setup.exe	45,701 KB	Application	5/5/2007 5:54 PM
yagarto-tools-20070303-setup.exe	700 KB	Application	5/5/2007 4:51 PM
Setup_JLinkARM_V368b.zip	3,937 KB	Compressed (zip...	5/8/2007 7:57 PM

The initial "Welcome" screen is shown below to the left; click on "**Next**" to continue. Accept the terms of the license agreement by clicking the "**I accept …**" radio button in the screen below right and then click "**Next**" to continue.

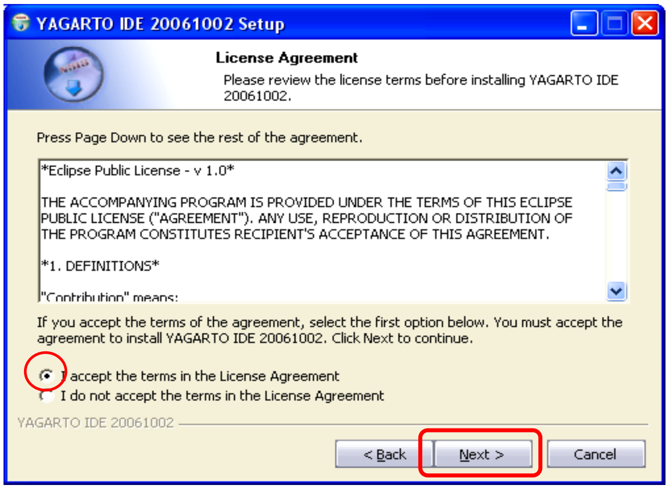


Welcome to the YAGARTO IDE 20061002 Setup Wizard

This wizard will guide you through the installation of YAGARTO IDE 20061002.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.



License Agreement

Please review the license terms before installing YAGARTO IDE 20061002.

Press Page Down to see the rest of the agreement.

Eclipse Public License - v 1.0

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

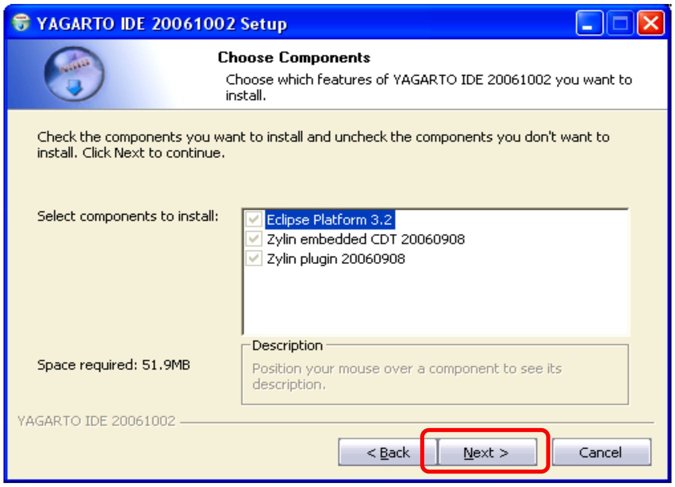
1. DEFINITIONS

"Contribution" means:

If you accept the terms of the agreement, select the first option below. You must accept the agreement to install YAGARTO IDE 20061002. Click Next to continue.

● I accept the terms in the License Agreement
○ I do not accept the terms in the License Agreement

You are forced to select Eclipse and the Zylin plug-ins in the “Choose Components” screen shown below to the left. Click on “**Next**” to continue. Take the default in the “Choose Install Location” screen below on the right. Click “**Next**” to continue.



YAGARTO IDE 20061002 Setup

Choose Components

Choose which features of YAGARTO IDE 20061002 you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

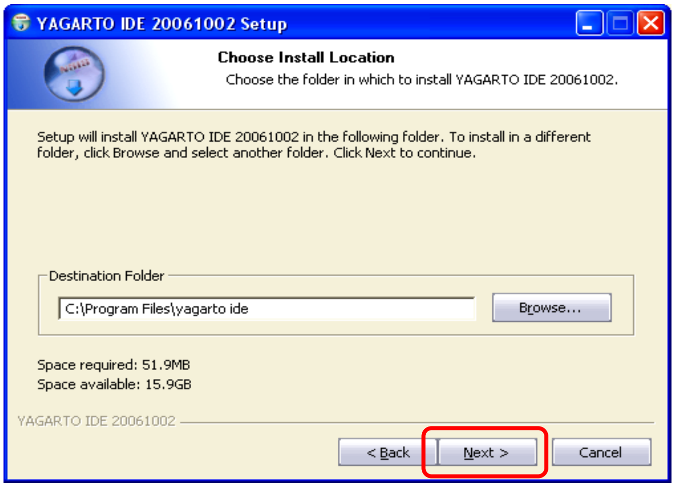
Select components to install:

☑ Eclipse Platform 3.2
☑ Zylin embedded CDT 20060908
☑ Zylin plugin 20060908

Space required: 51.9MB

Description
Position your mouse over a component to see its description.

YAGARTO IDE 20061002



YAGARTO IDE 20061002 Setup

Choose Install Location

Choose the folder in which to install YAGARTO IDE 20061002.

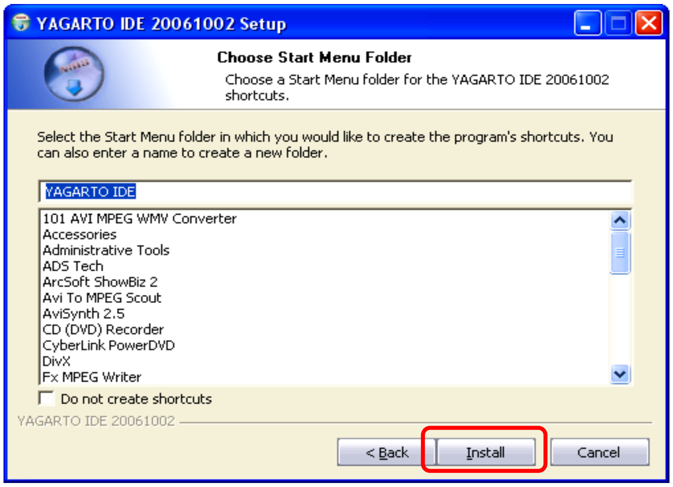
Setup will install YAGARTO IDE 20061002 in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder: C:\Program Files\yagarto ide

Space required: 51.9MB
Space available: 15.9GB

YAGARTO IDE 20061002

The Eclipse IDE can be added to the Start menu as shown in the “Choose Start Menu Folder” screen below on the left. Click “**Install**” to start the Eclipse installer. The Eclipse installer will commence and it will just take at most a couple of minutes. Click “**Next**” when the Eclipse installer finishes, as shown below to the right.



YAGARTO IDE 20061002 Setup

Choose Start Menu Folder

Choose a Start Menu folder for the YAGARTO IDE 20061002 shortcuts.

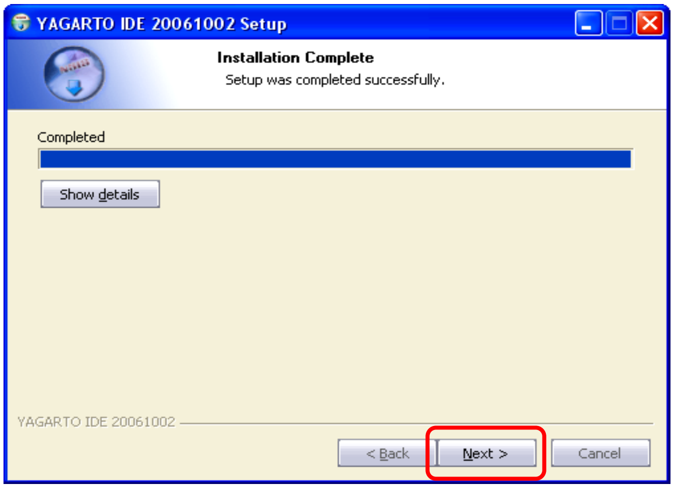
Select the Start Menu folder in which you would like to create the program's shortcuts. You can also enter a name to create a new folder.

YAGARTO IDE

101 AVI MPEG WMV Converter
Accessories
Administrative Tools
ADS Tech
ArcSoft ShowBiz 2
Avi To MPEG Scout
AviSynth 2.5
CD (DVD) Recorder
CyberLink PowerDVD
DivX
Fx MPEG Writer

□ Do not create shortcuts

YAGARTO IDE 20061002



YAGARTO IDE 20061002 Setup

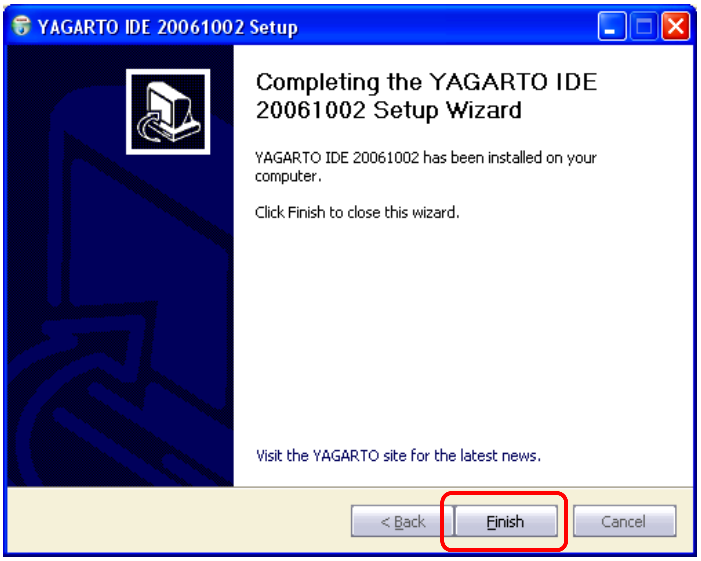
Installation Complete

Setup was completed successfully.

Completed

YAGARTO IDE 20061002

Finally, click “**Finish**” to exit the Eclipse installer as shown in the screen below. Make a mental note that YAGARTO installed the Eclipse components in the following folder: **c:\Program Files\Yagarto IDE**



YAGARTO IDE 20061002 Setup

Completing the YAGARTO IDE 20061002 Setup Wizard

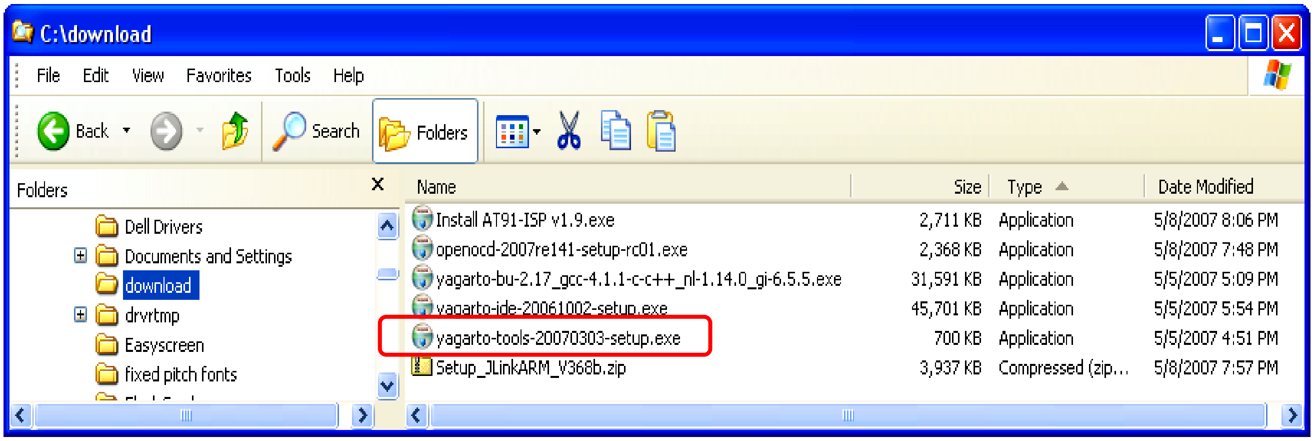
YAGARTO IDE 20061002 has been installed on your computer.

Click Finish to close this wizard.

Visit the YAGARTO site for the latest news.

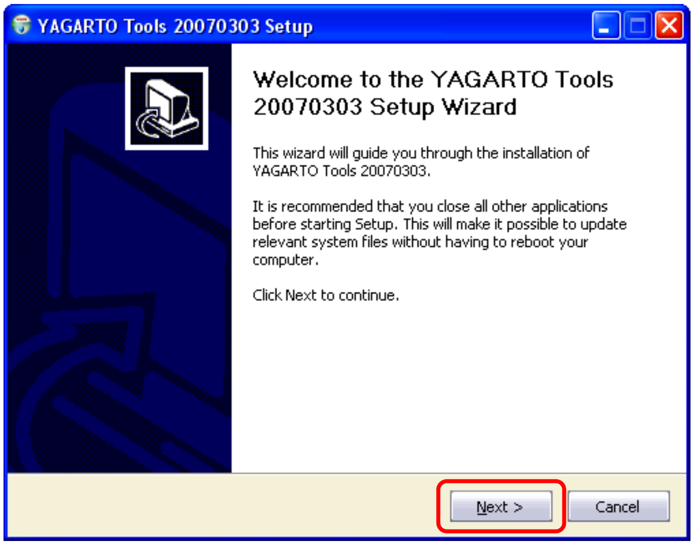
Install YAGARTO Tools

The YAGARTO Tools includes the GNU Make utility. Double-click on the file **Yagarto-tools-20070303-setup.exe** to start the YAGARTO tools installer.



Name	Size	Type	Date Modified
Install AT91-ISP v1.9.exe	2,711 KB	Application	5/8/2007 8:06 PM
openocd-2007re141-setup-rc01.exe	2,368 KB	Application	5/8/2007 7:48 PM
yagarto-bu-2.17_gcc-4.1.1-c-c++_nl-1.14.0_gi-6.5.5.exe	31,591 KB	Application	5/5/2007 5:09 PM
yagarto-ide-20061002-setup.exe	45,701 KB	Application	5/5/2007 5:54 PM
yagarto-tools-20070303-setup.exe	700 KB	Application	5/5/2007 4:51 PM
Setup_JLinkARM_V368b.zip	3,937 KB	Compressed (zip...	5/8/2007 7:57 PM

In the “Welcome” screen below on the left, click “**Next**” to continue. In the “License Agreement” screen below on the right, check the radio button to accept the license agreement and then click “**Next**” to proceed.

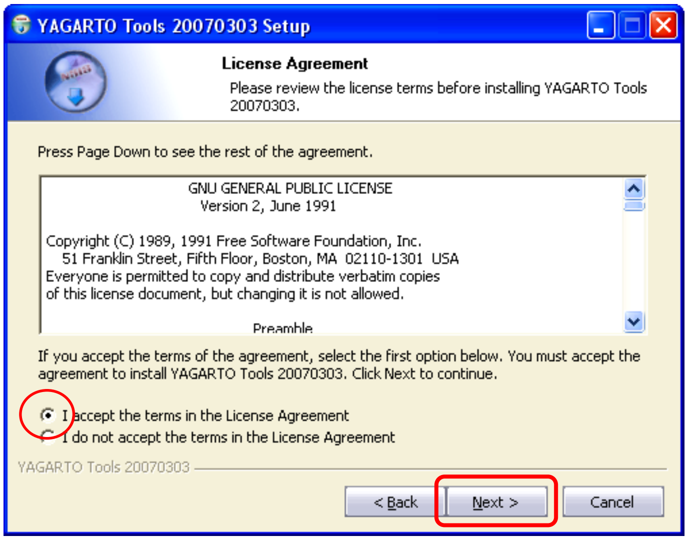


Welcome to the YAGARTO Tools 20070303 Setup Wizard

This wizard will guide you through the installation of YAGARTO Tools 20070303.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.



License Agreement

Please review the license terms before installing YAGARTO Tools 20070303.

Press Page Down to see the rest of the agreement.

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

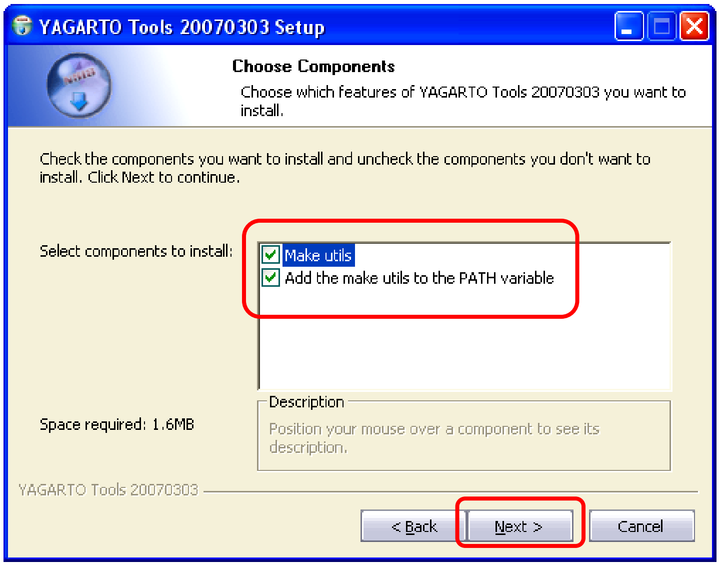
Preamble

If you accept the terms of the agreement, select the first option below. You must accept the agreement to install YAGARTO Tools 20070303. Click Next to continue.

● I accept the terms in the License Agreement
○ I do not accept the terms in the License Agreement

In the “Choose Components” screen below on the left, take the default (select both components) and click “**Next**” to continue.

In the “Choose Install Location” screen below on the right, take the default which is the destination folder “c:\Program Files\yagarto-tools-20070303\”. Click on “**Next**” to proceed.



YAGARTO Tools 20070303 Setup

Choose Components

Choose which features of YAGARTO Tools 20070303 you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select components to install:

☑ Make utils

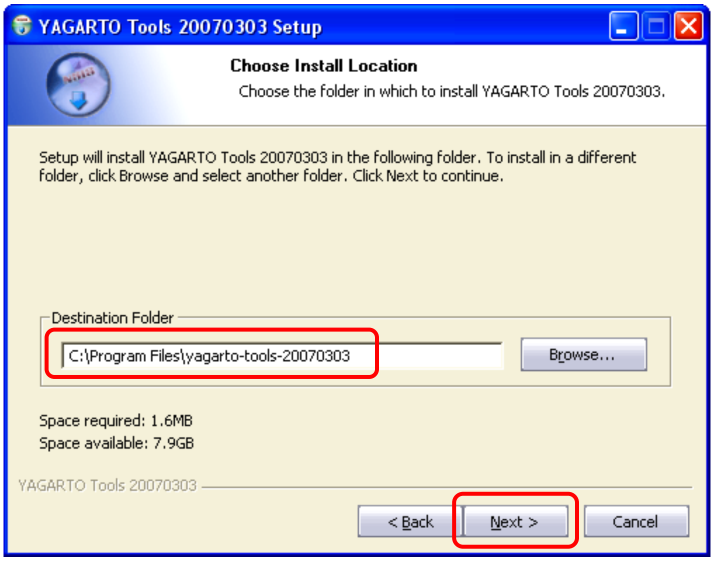
☑ Add the make utils to the PATH variable

Space required: 1.6MB

Description

Position your mouse over a component to see its description.

YAGARTO Tools 20070303



YAGARTO Tools 20070303 Setup

Choose Install Location

Choose the folder in which to install YAGARTO Tools 20070303.

Setup will install YAGARTO Tools 20070303 in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder

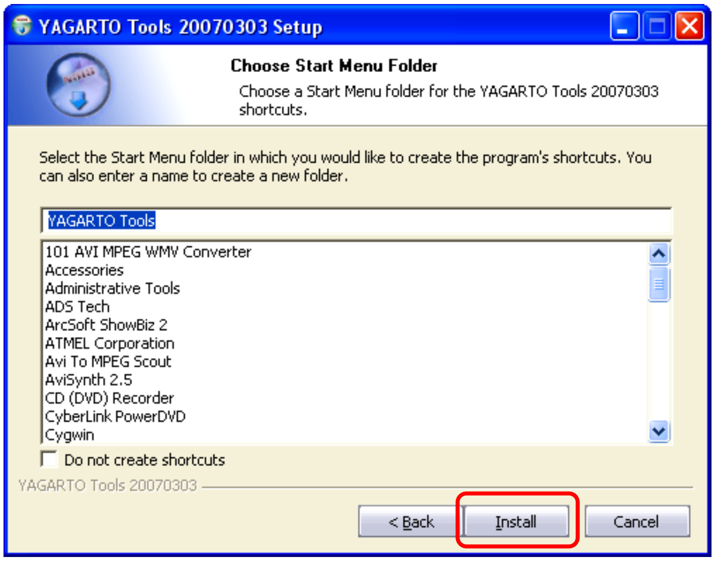
C:\Program Files\yagarto-tools-20070303

Space required: 1.6MB

Space available: 7.9GB

YAGARTO Tools 20070303

In the “Choose Start Menu Folder” screen below on the left, click “**Install**” to start the installation. You could elect to check the box labeled “Do not create shortcuts” since the Make utility is typically started from within Eclipse. The Yagarto tools will install quickly. When the “Installation Complete” screen appears as shown below in the right; click “**Next**” to continue.



YAGARTO Tools 20070303 Setup

Choose Start Menu Folder

Choose a Start Menu folder for the YAGARTO Tools 20070303 shortcuts.

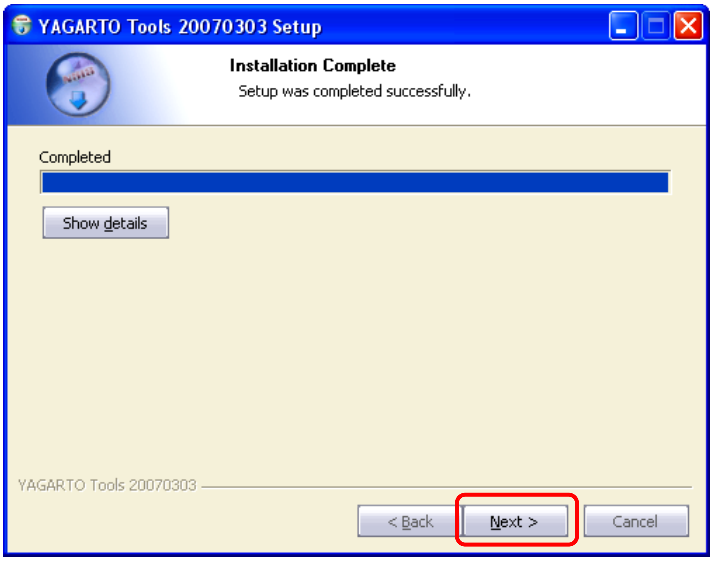
Select the Start Menu folder in which you would like to create the program's shortcuts. You can also enter a name to create a new folder.

YAGARTO Tools

101 AVI MPEG WMV Converter
Accessories
Administrative Tools
ADS Tech
ArcSoft ShowBiz 2
ATMEL Corporation
Avi To MPEG Scout
AviSynth 2.5
CD (DVD) Recorder
CyberLink PowerDVD
Cygwin

□ Do not create shortcuts

YAGARTO Tools 20070303



YAGARTO Tools 20070303 Setup

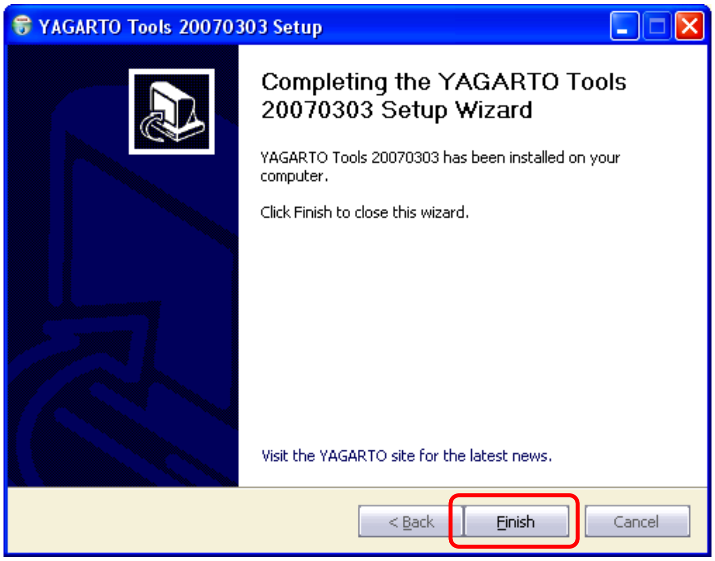
Installation Complete

Setup was completed successfully.

Completed

YAGARTO Tools 20070303

Finally, click on “**Finish**” as shown below to complete installation of the Yagarto tools.



YAGARTO Tools 20070303 Setup

Completing the YAGARTO Tools 20070303 Setup Wizard

YAGARTO Tools 20070303 has been installed on your computer.

Click Finish to close this wizard.

Visit the YAGARTO site for the latest news.

Install the Segger J-Link GDB Server

If you have purchased the Atmel SAM-ICE JTAG hardware interface, install the Seeger J-Link GDB Server as shown in this section.

SEGGER Microcontroller Systeme GmbH of Hilden, Germany supply hardware and software tools for the embedded software industry. They are manufacturers of the J-Link JTAG hardware debuggers and supply numerous software products in support thereof. One product of special interest is the J-Link GDB Server for connection to the Eclipse/GDB graphical debugger and another is the J-Flash EPROM programmer which can program on-chip and off-chip flash for a wide variety of microcontrollers.

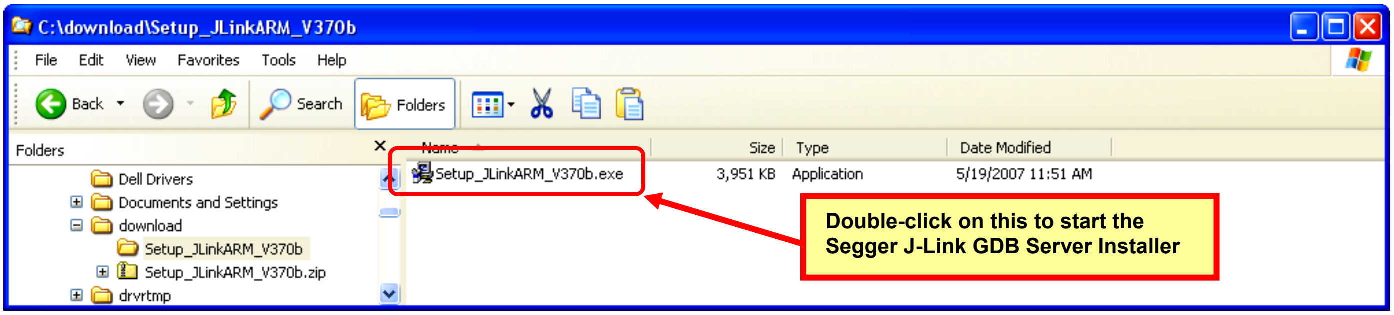
An Atmel branded version of the J-Link ARM debugger hardware, called the SAM-ICE, is available for use with the Atmel AT91SAM7 evaluation boards for $129 (US). That's a pretty good deal given that a standard Segger J-Link ARM USB JTAG hardware debugger retails for $327(US) or €248(Euro).

The Segger J-Link GDB Server interfaces the GDB Remote Serial Protocol emitted by Eclipse to the SAM-ICE JTAG Debugger. It operates as a daemon, a Windows program that operates in the background waiting for commands to process. If you have purchased an Atmel SAM-ICE, you automatically have an unlimited license to use the Segger J-Link GDB Server. The Atmel license for the Segger J-Link GDB Server software is a very good value since the commercial license for this product is $261(US) or €198(Euro).

The J-Link GDB Server cannot be used to program the onchip flash. Again, a Segger software package called J-Flash is available to do this for $525(US) or €398(Euro). Fortunately, the free Atmel SAM-BA utility can be used to program flash via the SAM-ICE JTAG Interface.

In the photo shown directly below, the SAM-ICE hardware debugger is connected to the PC's USB port and to the AT91SAM7S256-EK evaluation board's 20-pin JTAG connector. The board power is supplied by a standard 9 volt DC "wall wart" power supply.

If you are using the Atmel SAM-ICE or Segger J-Link JTAG hardware, the following installation will give Eclipse a compatible J-Link GDB Server to communicate to the target's JTAG port. Click on the installer "**Setup_JLinkARM_V370b.exe"** in the "**c:\download\Setup_JLinkARM_V370b**" folder.

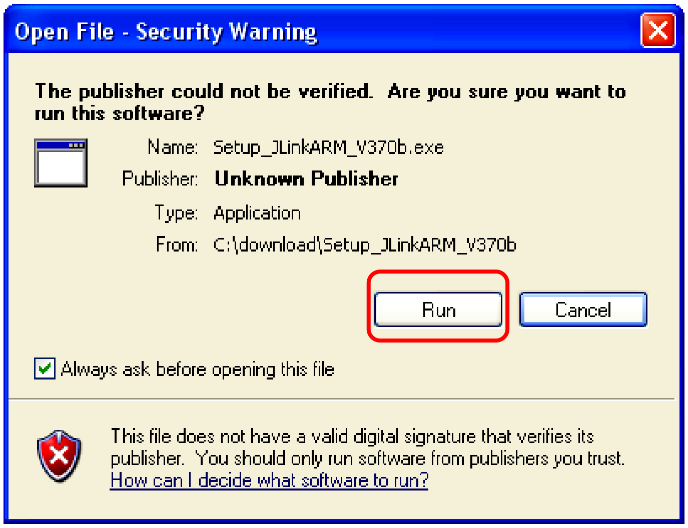


Name	Size	Type	Date Modified
Setup_JLinkARM_V370b.exe	3,951 KB	Application	5/19/2007 11:51 AM

Double-click on this to start the Segger J-Link GDB Server Installer

Ignore the Windows belly-aching about publisher verification and click on "**Run**" to continue.

The Segger "License Agreement" will be presented. Click on "**Yes**" to accept it.

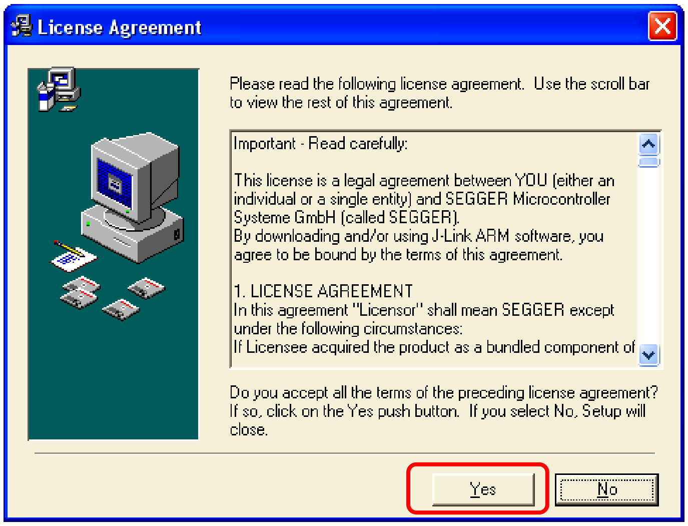

Open File - Security Warning

The publisher could not be verified. Are you sure you want to run this software?

Name: Setup_JLinkARM_V370b.exe
Publisher: **Unknown Publisher**
Type: Application
From: C:\download\Setup_JLinkARM_V370b

☑ Always ask before opening this file

This file does not have a valid digital signature that verifies its publisher. You should only run software from publishers you trust. How can I decide what software to run?


License Agreement

Please read the following license agreement. Use the scroll bar to view the rest of this agreement.

Important - Read carefully:

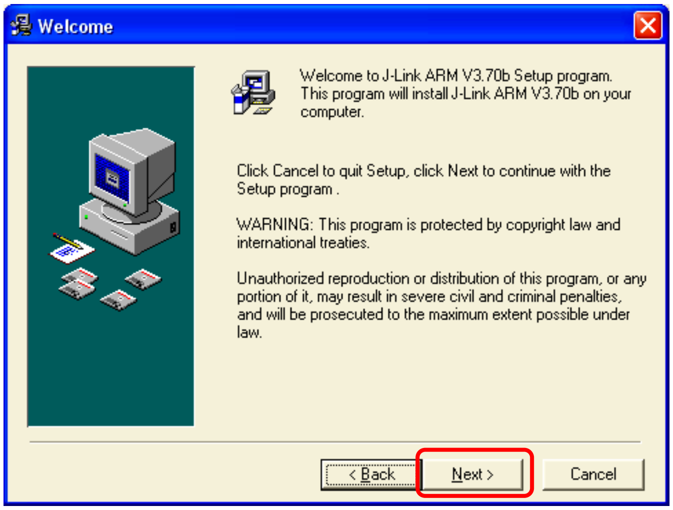
This license is a legal agreement between YOU (either an individual or a single entity) and SEGGER Microcontroller Systeme GmbH (called SEGGER).
By downloading and/or using J-Link ARM software, you agree to be bound by the terms of this agreement.

1. LICENSE AGREEMENT
In this agreement "Licensor" shall mean SEGGER except under the following circumstances:
If Licensee acquired the product as a bundled component of

Do you accept all the terms of the preceding license agreement? If so, click on the Yes push button. If you select No, Setup will close.

In the "Welcome" screen shown below left, click on "**Next**" to get started.

In the "Choose Destination Location" screen shown below right, take the default which will put the Segger components in a "c:\Program Files" subfolder. Click "**Next**" to continue.

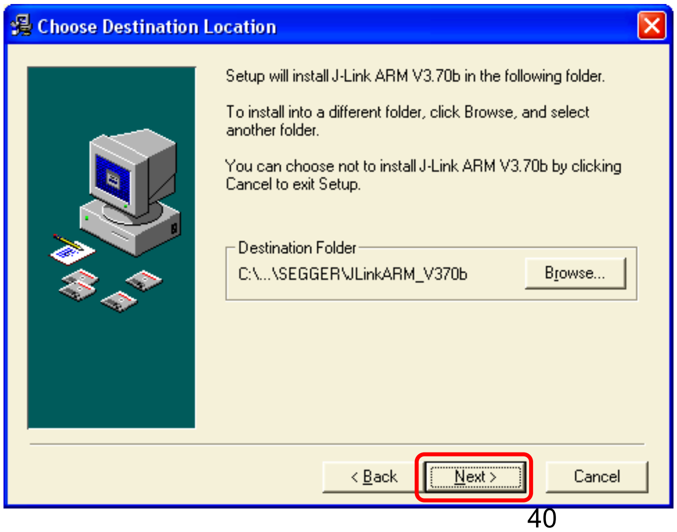

Welcome

Welcome to J-Link ARM V3.70b Setup program. This program will install J-Link ARM V3.70b on your computer.

Click Cancel to quit Setup, click Next to continue with the Setup program .

WARNING: This program is protected by copyright law and international treaties.

Unauthorized reproduction or distribution of this program, or any portion of it, may result in severe civil and criminal penalties, and will be prosecuted to the maximum extent possible under law.


Choose Destination Location

Setup will install J-Link ARM V3.70b in the following folder.

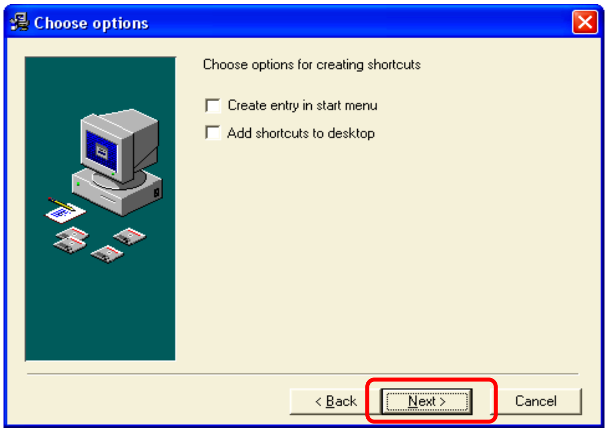
To install into a different folder, click Browse, and select another folder.

You can choose not to install J-Link ARM V3.70b by clicking Cancel to exit Setup.

Destination Folder
C:\...\SEGGER\JLinkARM_V370b

In the “**Choose Options**” panel shown below left, un-check the “**Create entry in start menu**” since we will be starting the Segger J-Link GDB Server from within Eclipse itself. Click “**Next** to continue.

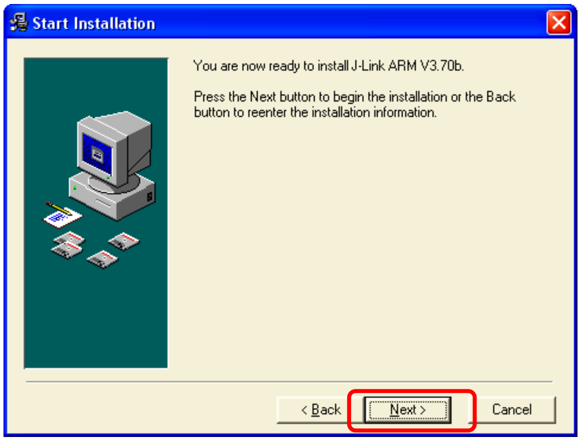
Click on “**Next**” to start the installation in the “Start Installation” screen shown below on the right.



Choose options for creating shortcuts

□ Create entry in start menu

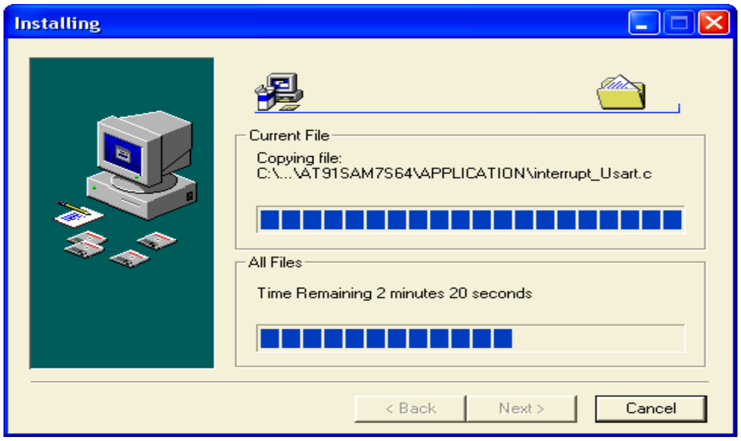
□ Add shortcuts to desktop



You are now ready to install J-Link ARM V3.70b.

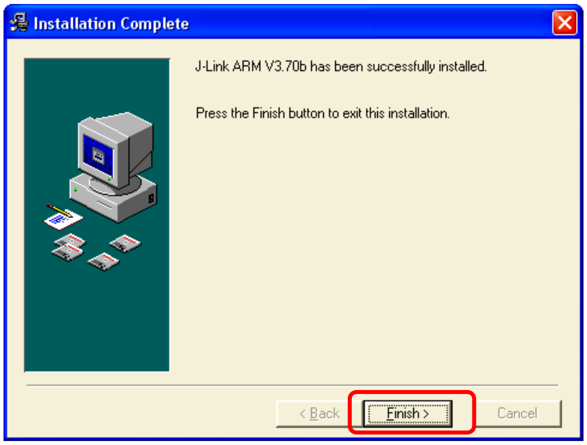
Press the Next button to begin the installation or the Back button to reenter the installation information.

The Segger J-Link package will install quickly; click on “**Finish**” as shown in the screen below right.



Copying file:
C:\...\AT91SAM7S64\APPLICATION\interrupt_Usart.c

Time Remaining 2 minutes 20 seconds



J-Link ARM V3.70b has been successfully installed.

Press the Finish button to exit this installation.

Install the Wiggler Parallel Port Driver

If you have purchased the Olimex ARM-JTAG hardware interface (called the “wiggler”), install the giveio.sys parallel port driver as shown in this section.

Unless you are a perfect programmer, you will occasionally require the services of a debugger to trap and identify software bugs. The AT91SAM7S256 microprocessor has special debug circuits on chip that can start and stop execution, read and write memory, and provide two hardware-assisted breakpoints. The interface to the outside world is a standard JTAG interface (essentially a very complicated and slow serial shift register protocol). You need a device called a JTAG debugger to connect your PC to the ARM chip’s JTAG pins. You also need a software program to operate that debugger and interface the JTAG protocol to the Eclipse/GDB source code debugger protocol; that software program is OpenOCD and you’ve already installed it.

One way to connect your PC to the AT91SAM7S-EK target board’s JTAG connector is to use an inexpensive device called a “wiggler”. This can be purchased from Olimex for $19.00 (US). It’s just a simple voltage level-shifter and it plugs into your PC’s parallel port.

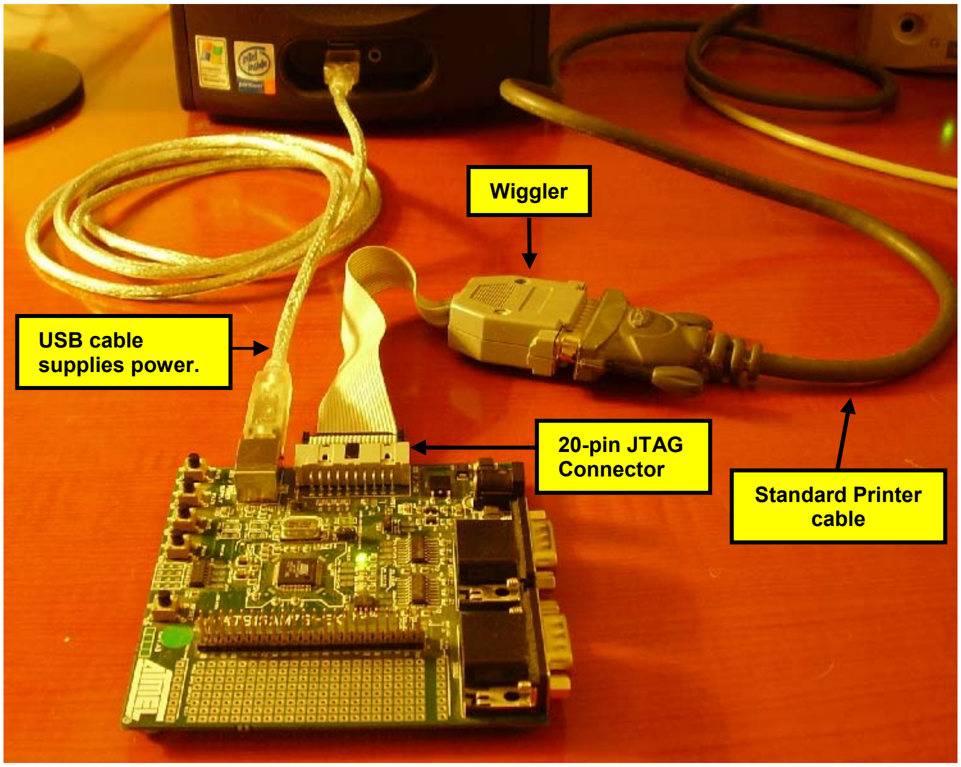
The ARM-JTAG device is available from:

www.olimex.com

www.sparkfun.com

www.microcontrollershop.com

The following is the hardware setup to debug the Atmel AT91SAM7S-EK evaluation board using the inexpensive “wiggler” device. A standard USB cable is connected to supply board power. The ARM-JTAG interface is attached to the PC’s printer port; in the author’s setup, a stock parallel port cable from the local computer store was employed. The JTAG 20-pin connector is keyed so it can’t be inserted improperly.

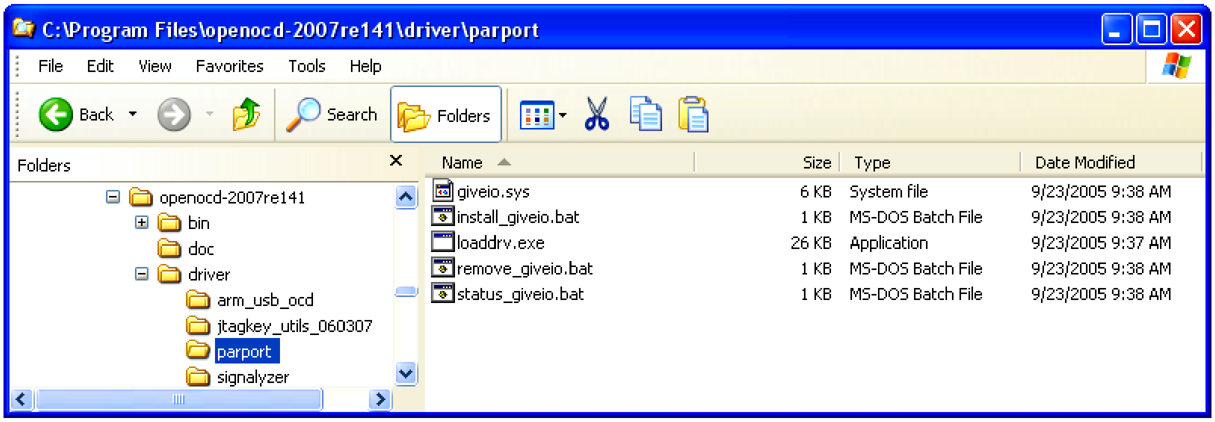


There are two well known criticisms of the “wiggler” device. First, the printer port of a PC limits operation of the JTAG to 500 Khz and this translates into slow downloading. Second, many PCs are now being manufactured without the customary serial and parallel port; the PC world is gravitating to the USB protocol.

If you are planning to use the inexpensive “wiggler” JTAG interface, a special **giveio.sys** driver has to be installed. This only needs to be done once.

The giveio.sys driver is in the folder: **c:\Program Files\openocd-2007re141\driver\parport\**

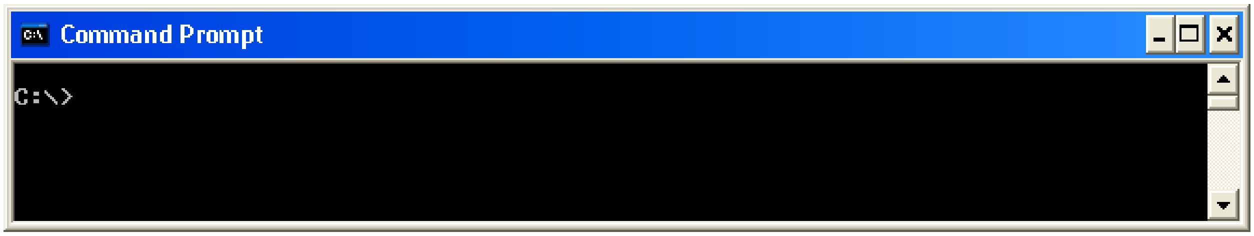
Note: check if this folder name has changed due to a newer YAGARTO release



Name	Size	Type	Date Modified
giveio.sys	6 KB	System file	9/23/2005 9:38 AM
install_giveio.bat	1 KB	MS-DOS Batch File	9/23/2005 9:38 AM
loaddrv.exe	26 KB	Application	9/23/2005 9:37 AM
remove_giveio.bat	1 KB	MS-DOS Batch File	9/23/2005 9:38 AM
status_giveio.bat	1 KB	MS-DOS Batch File	9/23/2005 9:38 AM

Start the installation of the giveio.sys driver by opening up a Command Prompt window (for really experienced readers, that’s the old DOS window). The “command prompt” can be found in your Windows start menu “**Start – All Programs – Accessories**”. If your Command Prompt window is not at the root folder **c:\**, you can type the **CD ** command shown below to locate yourself at the root folder.

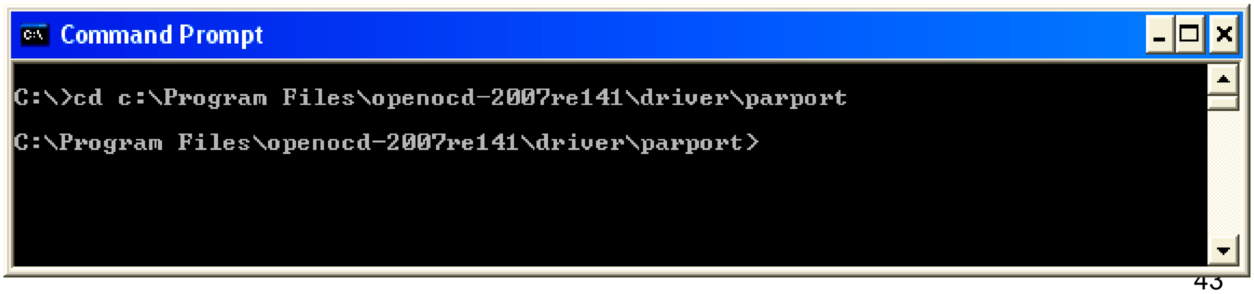
**>cd **



```
C:\>
```

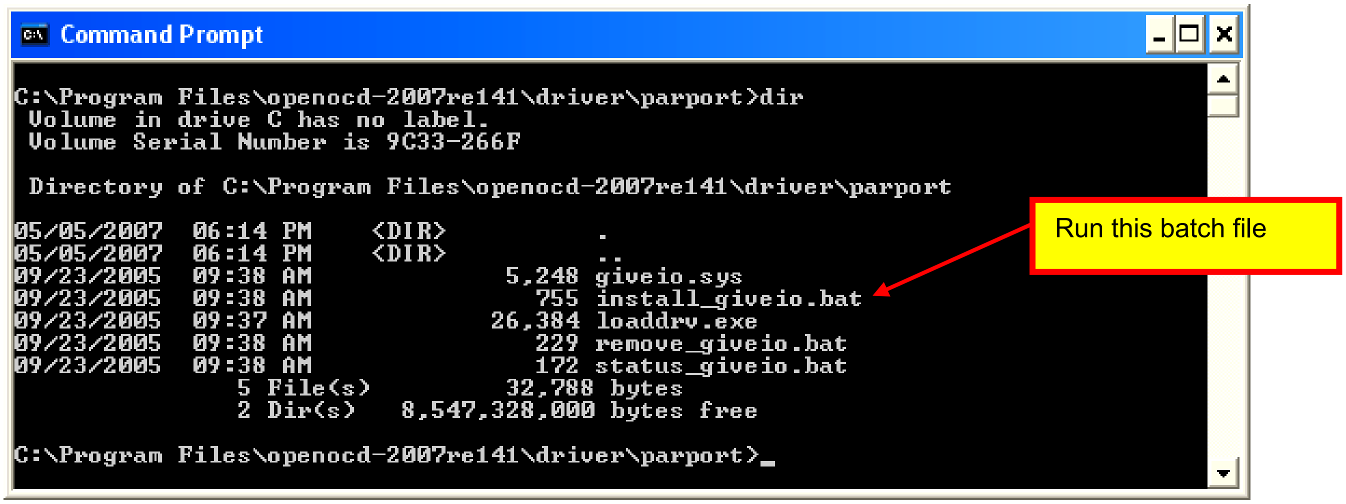
We need to change to the directory: **c:\Program Files\openocd-2007re141\driver\parport\** since it contains the giveio.bat installation batch file. Type the **CD** command again as shown below to do this. Now the command prompt window will show that we are inside that folder.

>cd c:\Program Files\openocd-2007re141\driver\parport\



```
C:\>cd c:\Program Files\openocd-2007re141\driver\parport

C:\Program Files\openocd-2007re141\driver\parport>
```

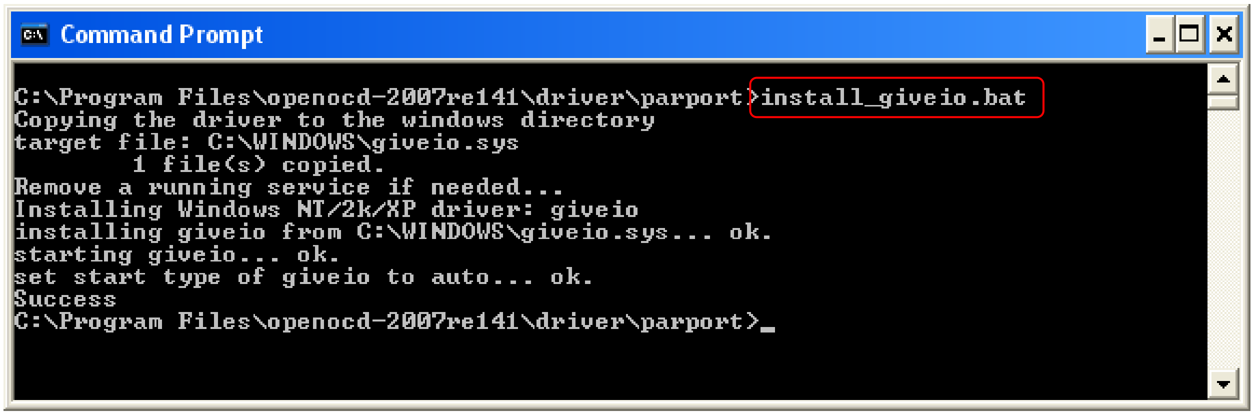
Now take a look at the contents of this folder by typing the **DIR** command.



```
C:\Program Files\openocd-2007re141\driver\parport>dir
 Volume in drive C has no label.
 Volume Serial Number is 9C33-266F

 Directory of C:\Program Files\openocd-2007re141\driver\parport

05/05/2007  06:14 PM    <DIR>          .
05/05/2007  06:14 PM    <DIR>          ..
09/23/2005  09:38 AM             5,248 giveio.sys
09/23/2005  09:38 AM               755 install_giveio.bat
09/23/2005  09:37 AM            26,384 loaddrv.exe
09/23/2005  09:38 AM               229 remove_giveio.bat
09/23/2005  09:38 AM               172 status_giveio.bat
               5 File(s)         32,788 bytes
               2 Dir(s)   8,547,328,000 bytes free

C:\Program Files\openocd-2007re141\driver\parport>_
```

Run this batch file

We want to run the command batch file “**install_giveio.bat**”. This will install the giveio.sys driver and load and start it. The batch file may be run by entering its name on the command line and hitting “**Enter**”. As you can see from the command history below, giveio was successfully installed as a Windows driver.



```
C:\Program Files\openocd-2007re141\driver\parport>install_giveio.bat
Copying the driver to the windows directory
target file: C:\WINDOWS\giveio.sys
        1 file(s) copied.
Remove a running service if needed...
Installing Windows NT/2k/XP driver: giveio
installing giveio from C:\WINDOWS\giveio.sys... ok.
starting giveio... ok.
set start type of giveio to auto... ok.
Success
C:\Program Files\openocd-2007re141\driver\parport>_
```

The giveio.sys driver is a permanent installation; you only have to do this once.

Install the Amontec JTAGkey USB Drivers

If you have purchased one of the Amontec JTAG hardware interfaces, install the Amontec USB drivers as shown in this section.

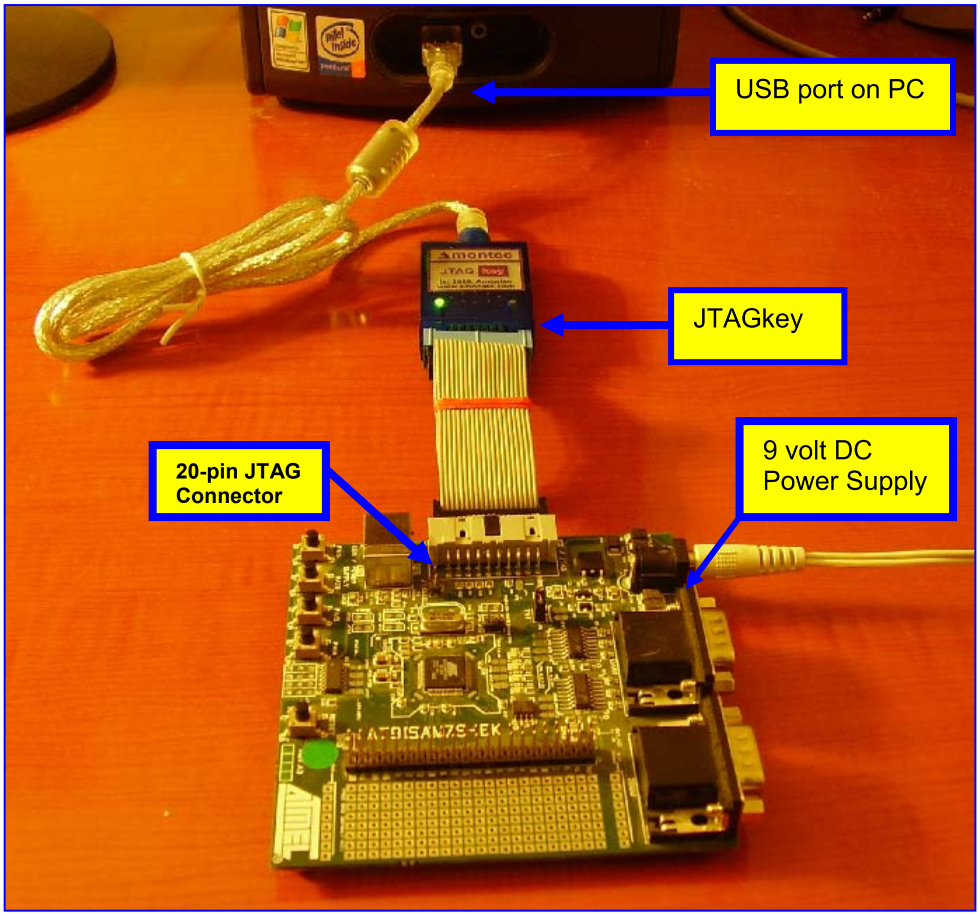
In Dominic Rath’s thesis about the OpenOCD project, a USB-JTAG interface based on the FTDI FT2232C engine was described with a schematic. The Swiss engineering firm Amontec has developed and marketed a professional version of this USB JTAG interface called the JTAGkey. Its price is є139 (euros) or $177 (us). The JTAGkey is professionally designed and manufactured with additional bells and whistles, such as status LEDs and ESD protection. JTAGkey also automatically senses and adjusts the level shifters for the ARM voltage level; this will come in handy when lower voltage versions (e.g. 1.8 volts) of the ARM become available. The Amontec JTAGKey can be purchased online from here:

http://www.amontec.com/jtagkey.shtml

Amontec has also addressed the hobbyist and student market with the JTAGkey-Tiny device, priced at є29 (euros) or $37 (us) and illustrated below. This smaller JTAGkey-Tiny device plugs directly into the 20-pin JTAG connector and uses a mini-USB cable to attach to the PC (you have to supply this cable – it’s similar to the USB cables supplied with digital cameras). The installation procedure is similar to that of the more expensive JTAGkey shown below.

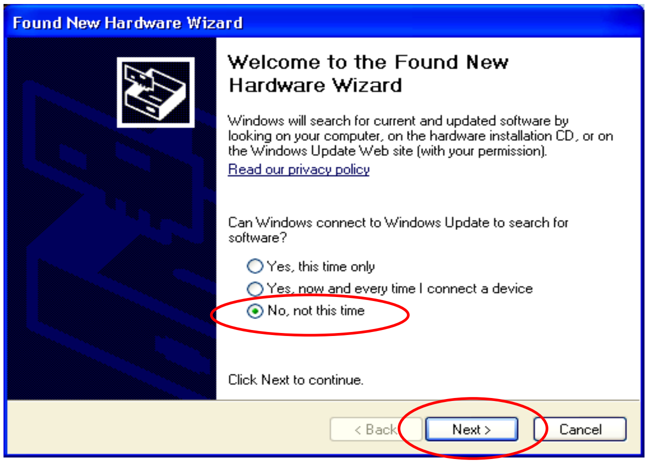
Professionals would tend to select the more expensive JTAGkey for its ESD protection and the flat ribbon cable that attaches to the prototype system, as seen in the hardware setup coming up. It also has an integrated USB cable fitted with a ferrite filter. The JTAGkey-Tiny plugs directly into the application board’s 20-pin JTAG connector and therefore must have the vertical clearance to permit this fitting.

The hardware setup, shown below, includes the Amontec JTAGkey plugged into the 20-pin JTAG header on the AT91SAM7S-EK target board and also into the PC’s USB port. The JTAG does not supply board power, so in this example a 9-volt DC “wall wart” power supply is fitted to the power connector. If you have a spare USB port on your PC, you could use another USB cable to supply board power instead.



Plug in the Amontec JTAGkey into the USB port. You should hear the familiar USB "beep" sound followed by the following screen indicating that new USB hardware has been detected.

The virtual device drivers are already on our "c:\Program Files\openocd" folder thanks to Michael Fischer's OpenOCD installation program we ran earlier in this tutorial. Therefore, advise Windows NOT to search for the drivers by clicking on "**No, not this time**" as shown below. Click "**Next**" to continue.



Found New Hardware Wizard

Welcome to the Found New Hardware Wizard

Windows will search for current and updated software by looking on your computer, on the hardware installation CD, or on the Windows Update Web site (with your permission).

Read our privacy policy

Can Windows connect to Windows Update to search for software?

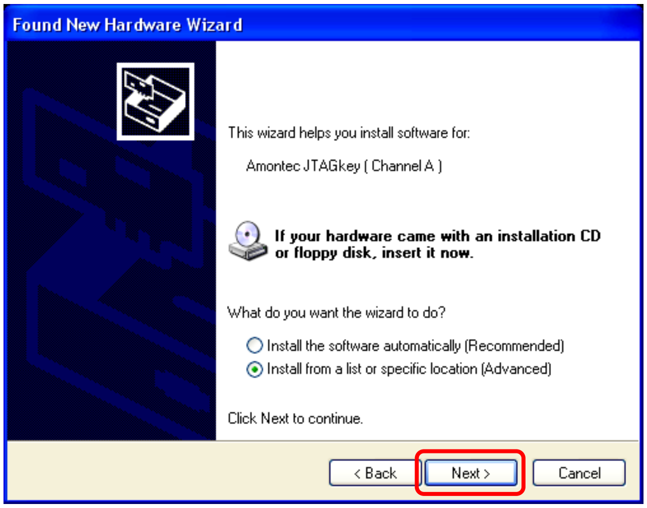
○ Yes, this time only

○ Yes, now and every time I connect a device

● No, not this time

Click Next to continue.

Instruct Windows to "**Install from a list or specific location (Advanced)**" as shown on the left hand screen below. Click "**Next** to continue. Now use the "**Browse**" button to find the directory "**c:\Program Files\openocd-2007re141\driver\jtagkey_utils_060307\**" as shown below on the right hand screen. Click "**Next** to continue.



This wizard helps you install software for:

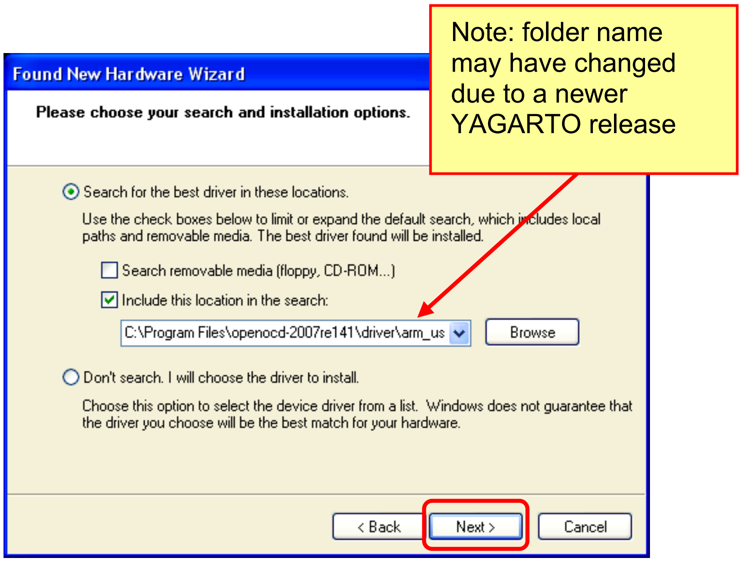
Amontec JTAGkey (Channel A)

If your hardware came with an installation CD or floppy disk, insert it now.

What do you want the wizard to do?

○ Install the software automatically (Recommended)
● Install from a list or specific location (Advanced)

Click Next to continue.



Please choose your search and installation options.

● Search for the best driver in these locations.

Use the check boxes below to limit or expand the default search, which includes local paths and removable media. The best driver found will be installed.

☐ Search removable media (floppy, CD-ROM...)
☑ Include this location in the search:

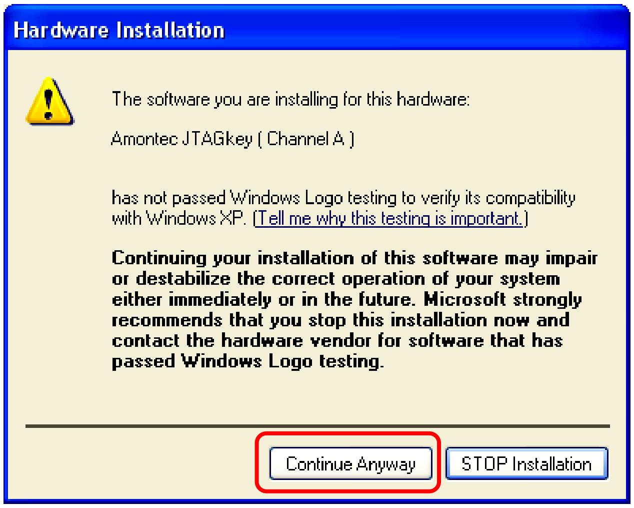
C:\Program Files\openocd-2007re141\driver\arm_us

○ Don't search. I will choose the driver to install.

Choose this option to select the device driver from a list. Windows does not guarantee that the driver you choose will be the best match for your hardware.

Note: folder name may have changed due to a newer YAGARTO release

Pay no attention to Windows complaints about Logo testing by clicking on "**Continue Anyway**" on the left screen below. The virtual device driver installation for Channel A will now run to completion.

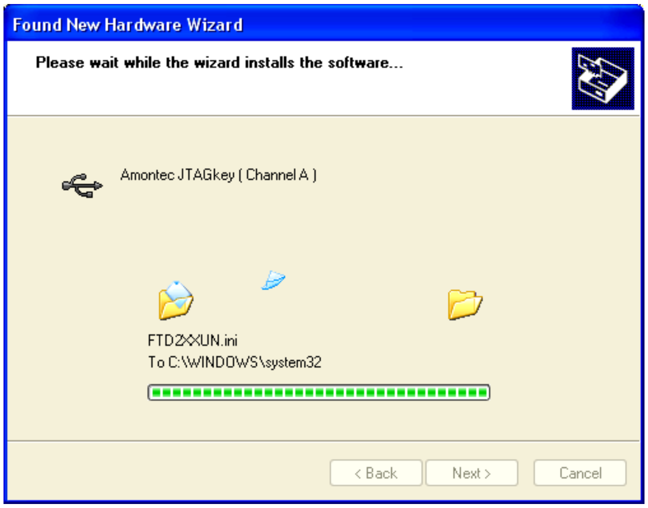


The software you are installing for this hardware:

Amontec JTAGkey (Channel A)

has not passed Windows Logo testing to verify its compatibility with Windows XP. (Tell me why this testing is important.)

Continuing your installation of this software may impair or destabilize the correct operation of your system either immediately or in the future. Microsoft strongly recommends that you stop this installation now and contact the hardware vendor for software that has passed Windows Logo testing.

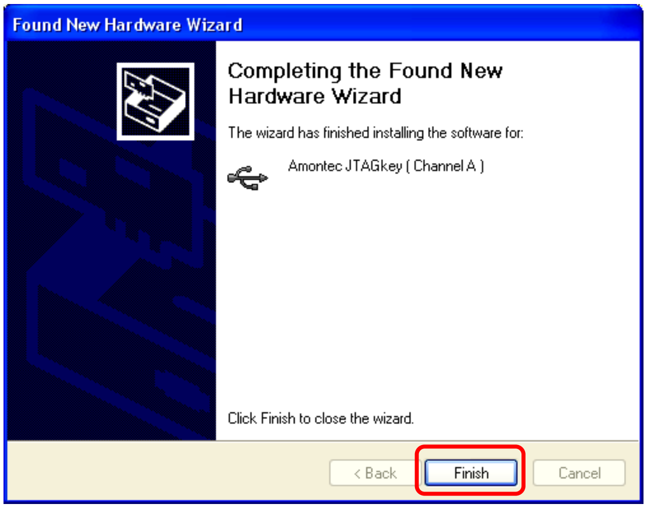


Please wait while the wizard installs the software...

Amontec JTAGkey (Channel A)

FTD2XXUN.ini
To C:\WINDOWS\system32

Click "**Finish**" to complete installation of the Channel A driver.



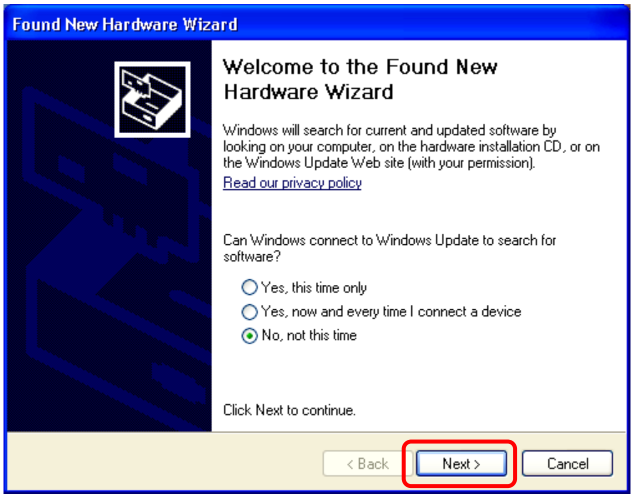
Completing the Found New Hardware Wizard

The wizard has finished installing the software for:

Amontec JTAGkey (Channel A)

Click Finish to close the wizard.

The JTAGkey is built around the FTDI FT2232C engine which has two channels. Exactly the same installation sequence is required for channel B. Follow the screens on this page in sequence, exactly like the channel A virtual device driver installation.



Found New Hardware Wizard

Welcome to the Found New Hardware Wizard

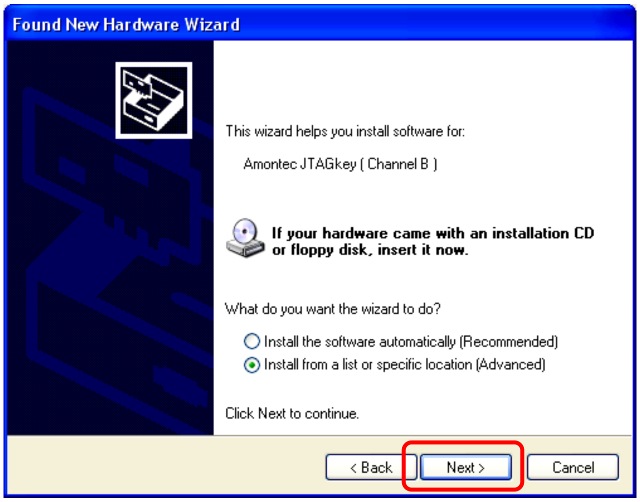
Windows will search for current and updated software by looking on your computer, on the hardware installation CD, or on the Windows Update Web site (with your permission).

Read our privacy policy

Can Windows connect to Windows Update to search for software?

○ Yes, this time only
○ Yes, now and every time I connect a device
● No, not this time

Click Next to continue.



Found New Hardware Wizard

This wizard helps you install software for:

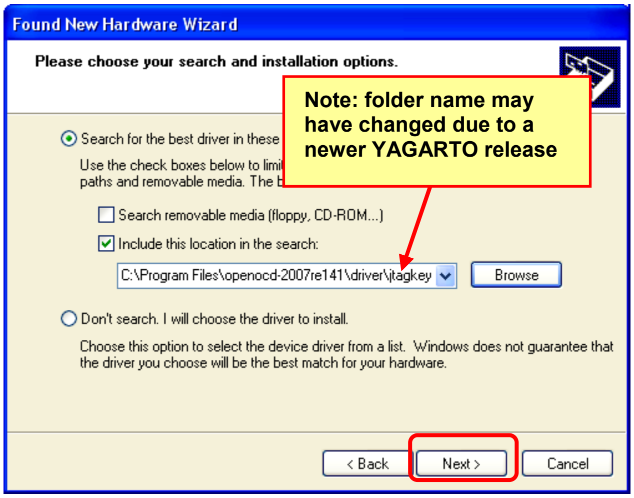
Amontec JTAGkey (Channel B)

If your hardware came with an installation CD or floppy disk, insert it now.

What do you want the wizard to do?

○ Install the software automatically (Recommended)
● Install from a list or specific location (Advanced)

Click Next to continue.



Found New Hardware Wizard

Please choose your search and installation options.

● Search for the best driver in these

Use the check boxes below to limi
paths and removable media. The b

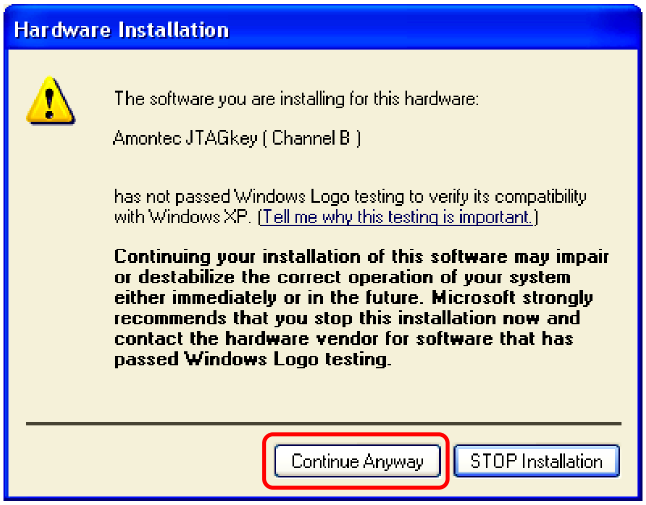
□ Search removable media (floppy, CD-ROM...)
☑ Include this location in the search:

C:\Program Files\openocd-2007re141\driver\jtagkey

○ Don't search. I will choose the driver to install.

Choose this option to select the device driver from a list. Windows does not guarantee that the driver you choose will be the best match for your hardware.

Note: folder name may have changed due to a newer YAGARTO release



Hardware Installation

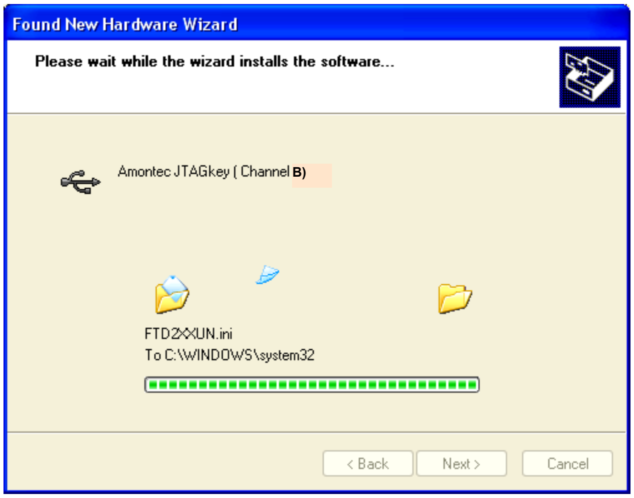
The software you are installing for this hardware:

Amontec JTAGkey (Channel B)

has not passed Windows Logo testing to verify its compatibility with Windows XP. (Tell me why this testing is important.)

Continuing your installation of this software may impair or destabilize the correct operation of your system either immediately or in the future. Microsoft strongly recommends that you stop this installation now and contact the hardware vendor for software that has passed Windows Logo testing.

When the Channel B driver has completed, you will see the screen below right indicating successful installation. Click “**Finish**” to exit the channel B installation as shown below.

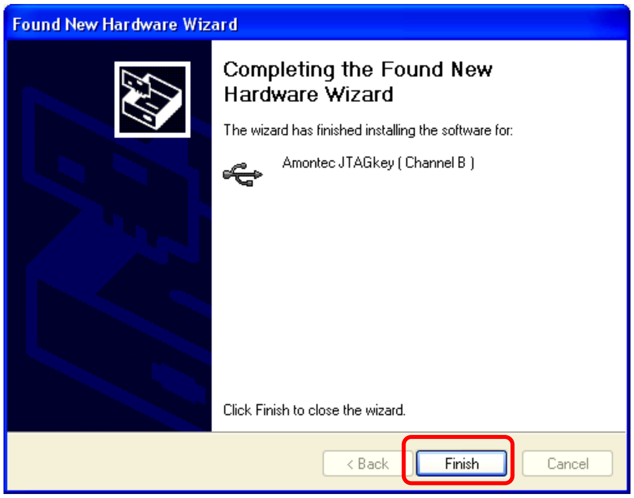


Found New Hardware Wizard

Please wait while the wizard installs the software...

Amontec JTAGkey (Channel B)

FTD2XXUN.ini
To C:\WINDOWS\system32



Found New Hardware Wizard

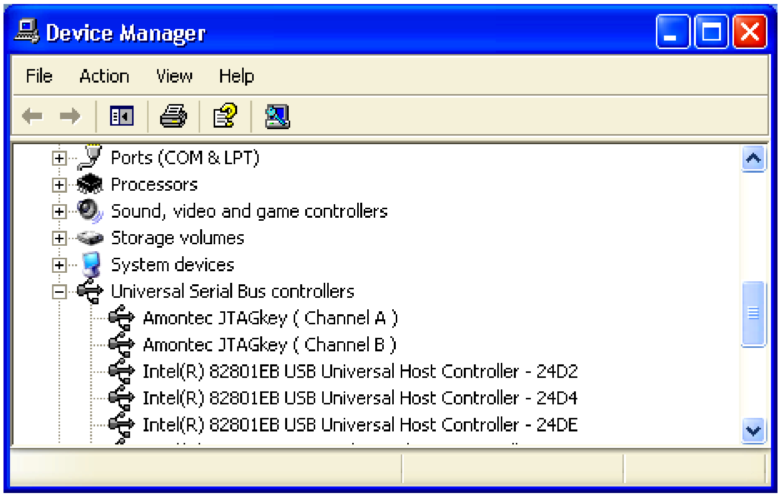
Completing the Found New Hardware Wizard

The wizard has finished installing the software for:

Amontec JTAGkey (Channel B)

Click Finish to close the wizard.

To be sure of successful installation of these JTAGkey virtual device drivers, use the Windows Start menu to look at the "**Control Panel – System – Hardware - Device Manager**", inspecting carefully the USB controllers. As can be seen below, the Amontec JTAGkey channel A and channel B USB ports are successfully installed.



```
+ Ports (COM & LPT)
+ Processors
+ Sound, video and game controllers
+ Storage volumes
+ System devices
- Universal Serial Bus controllers
    Amontec JTAGkey ( Channel A )
    Amontec JTAGkey ( Channel B )
    Intel(R) 82801EB USB Universal Host Controller - 24D2
    Intel(R) 82801EB USB Universal Host Controller - 24D4
    Intel(R) 82801EB USB Universal Host Controller - 24DE
```

Install the Olimex ARM-USB-OCD USB Drivers

If you have purchased one of the Olimex JTAG hardware interfaces, install the Olimex USB drivers as shown in this section.

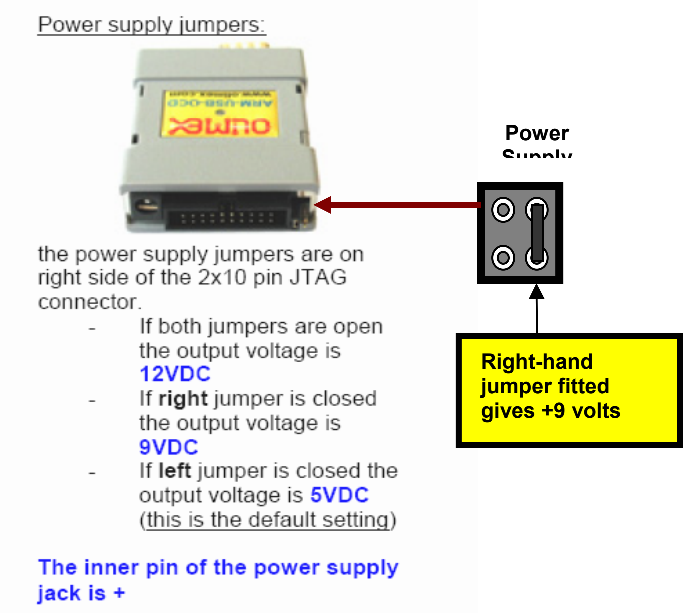
Olimex also developed a version of the USB-based JTAG debugger mentioned in Dominic Rath's OpenOCD thesis. It includes a couple of unique features such as an extra serial port (might come in handy if you have a laptop with no serial port) and a DC power supply that can be strapped for 5v, 9v or 12v operation. This DC supply includes a cable that can power your board, if needed. The Olimex ARM-USB-OCD debugger is є55 (euros) or $69.95 (US). If you want to use the Olimex ARM-USB-OCD JTAG device to program on chip flash memory, it would be better to use a wall-wart external power supply for the target board since the ARM-USB-OCD device doesn't supply enough power for the Atmel AT91SAM7S256-EK board during flash programming operations.

Recently, Olimex has added a low end USB-based JTAG debugger called the ARM-USB-Tiny. It costs $49.95 (US) or є37.34 (euros) and comes without the extra serial port or power supply.

Olimex ARM-USB-OCD

Olimex ARM-USB-Tiny

To use the ARM-USB-OCD power supply, there are jumpers to set the voltage. While the Atmel specification for the AT91SAM7S256-EK board is 7 – 12 volts for the DC supply, it worked for the author at all the above voltage ranges. Just to be safe, strap the Olimex ARM-USB-OCD DC supply to +9 volts (right-hand jumper installed).



Power supply jumpers:

the power supply jumpers are on right side of the 2x10 pin JTAG connector.

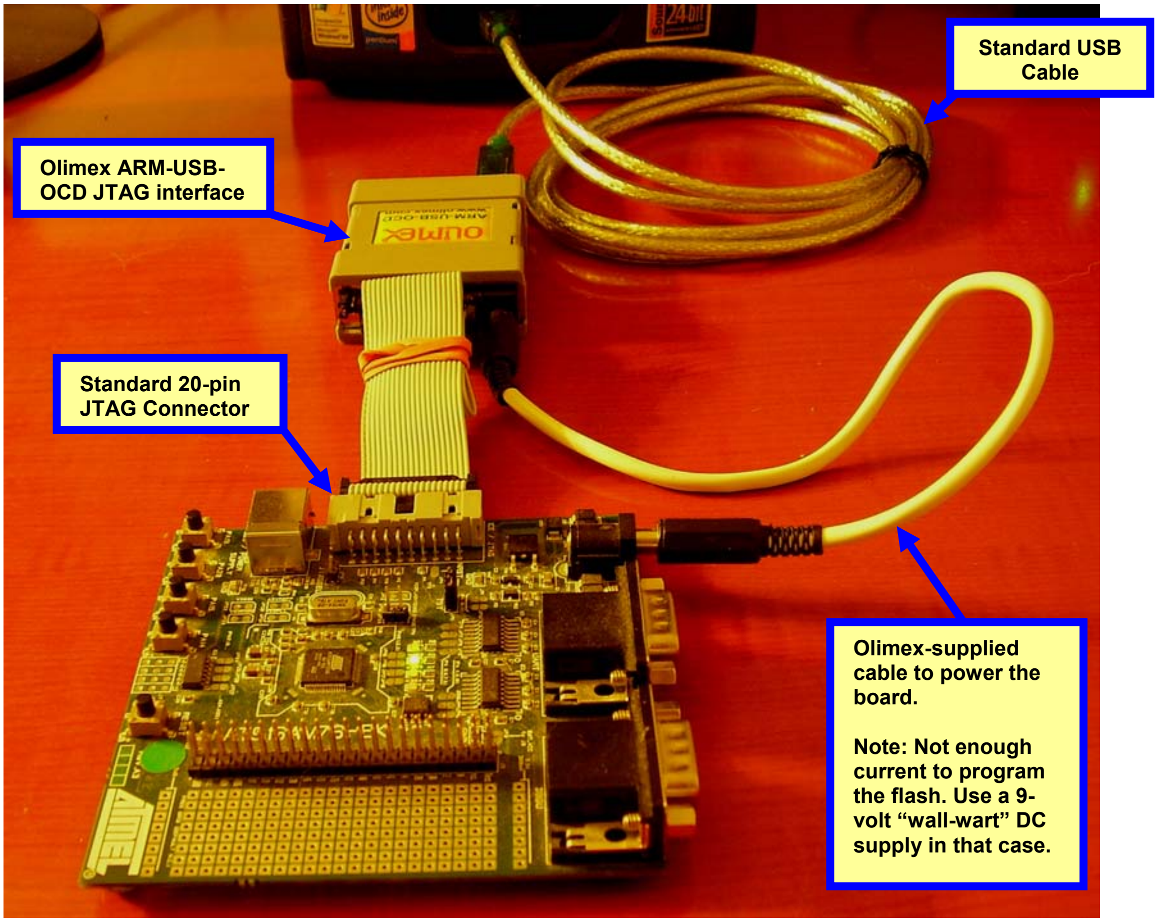
- If both jumpers are open the output voltage is **12VDC**
- If **right** jumper is closed the output voltage is **9VDC**
- If **left** jumper is closed the output voltage is **5VDC** (this is the default setting)

The inner pin of the power supply jack is +

Power Supply

Right-hand jumper fitted gives +9 volts

The hardware setup for the Atmel AT91SAM7S256-EK board is shown below. The 20-pin JTAG ribbon cable connectors are keyed so they can't be fitted improperly. The DC supply cable from the ARM-USB-OCD dongle powers the board.

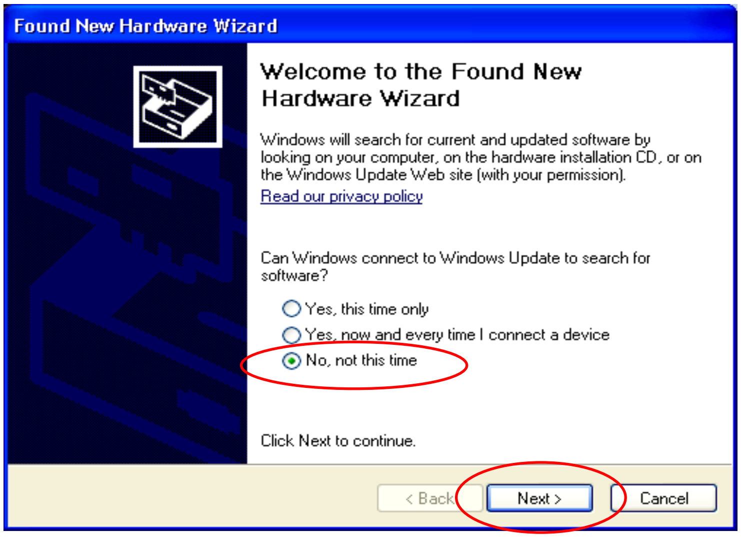

Standard USB Cable
Olimex ARM-USB-OCD JTAG interface
Standard 20-pin JTAG Connector

Olimex-supplied cable to power the board.

Note: Not enough current to program the flash. Use a 9-volt "wall-wart" DC supply in that case.

Plug in the Olimex ARM-USB-OCD dongle into the USB port. You should hear the familiar USB "beep" sound followed by the following screen indicating that new USB hardware has been detected.

The virtual device drivers are already on our "c:\Program Files\openocd-2007re141\driver\arm_usb_ocd\" folder thanks to Michael Fischer's OpenOCD installation program we ran earlier in this tutorial. Therefore, advise Windows NOT to search for the drivers by clicking on "**No, not this time**" as shown below. Click "**Next**" to continue.



Found New Hardware Wizard

Welcome to the Found New Hardware Wizard

Windows will search for current and updated software by looking on your computer, on the hardware installation CD, or on the Windows Update Web site (with your permission).

Read our privacy policy

Can Windows connect to Windows Update to search for software?

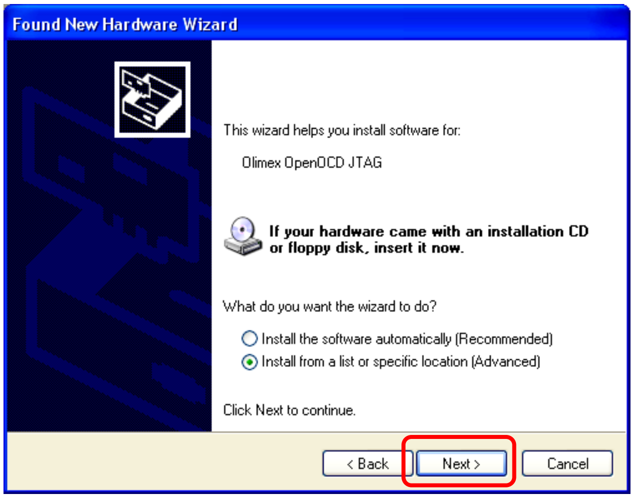
○ Yes, this time only

○ Yes, now and every time I connect a device

● No, not this time

Click Next to continue.

Instruct Windows to “**Install from a list or specific location (Advanced)**” as shown on the left hand screen below. Click “**Next** to continue. Now use the “**Browse**” button to find the directory “**c:\Program Files\openocd-2007re141\driver\arm_usb_ocd\**” as shown below on the right hand screen. Click “**Next** to continue.



Found New Hardware Wizard

This wizard helps you install software for:

Olimex OpenOCD JTAG

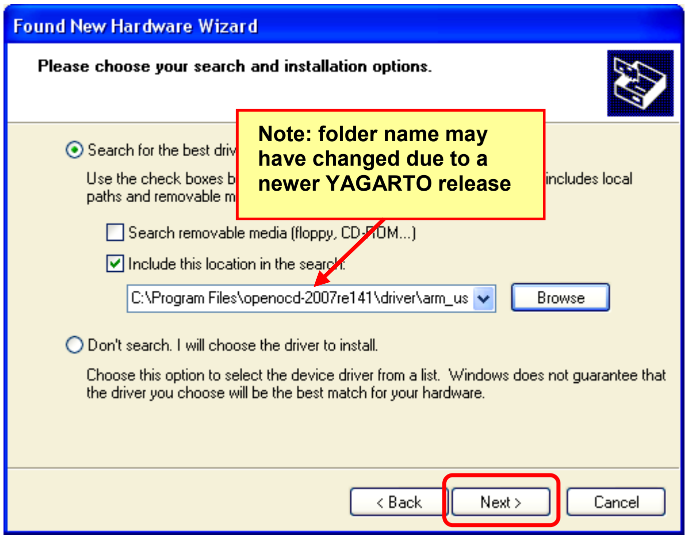
If your hardware came with an installation CD or floppy disk, insert it now.

What do you want the wizard to do?

○ Install the software automatically (Recommended)

● Install from a list or specific location (Advanced)

Click Next to continue.



Found New Hardware Wizard

Please choose your search and installation options.

● Search for the best driv

Use the check boxes b   includes local paths and removable m

□ Search removable media (floppy, CD-ROM...)

☑ Include this location in the search:

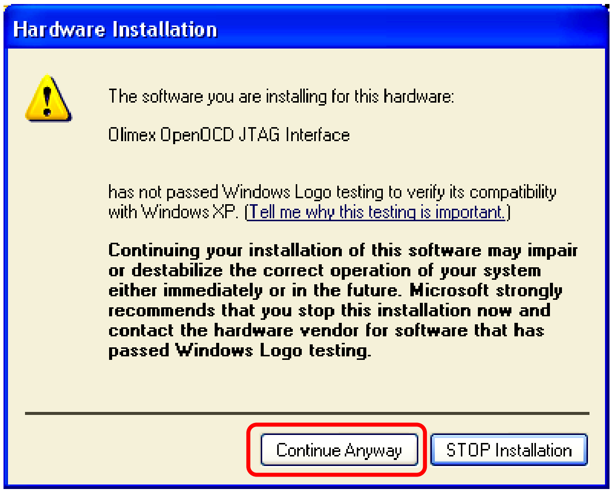
C:\Program Files\openocd-2007re141\driver\arm_us

○ Don't search. I will choose the driver to install.

Choose this option to select the device driver from a list. Windows does not guarantee that the driver you choose will be the best match for your hardware.

Note: folder name may have changed due to a newer YAGARTO release

Ignore the Windows XP complaint about “Logo Testing” by clicking “**Continue Anyway**” as shown on the left below. The installer will now start installation activities.



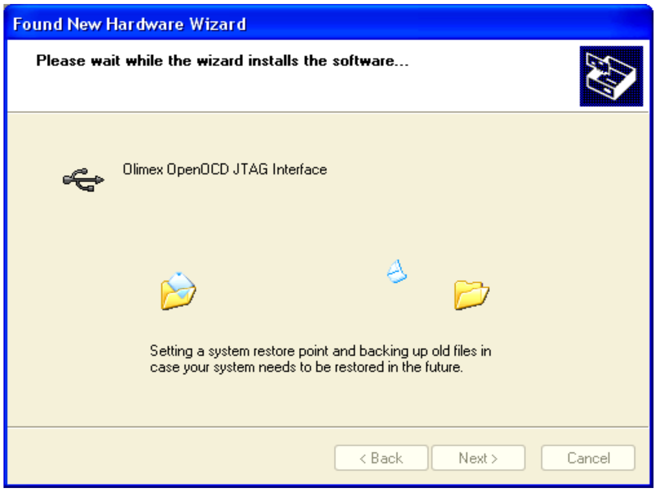
Hardware Installation

The software you are installing for this hardware:

Olimex OpenOCD JTAG Interface

has not passed Windows Logo testing to verify its compatibility with Windows XP. (Tell me why this testing is important.)

Continuing your installation of this software may impair or destabilize the correct operation of your system either immediately or in the future. Microsoft strongly recommends that you stop this installation now and contact the hardware vendor for software that has passed Windows Logo testing.



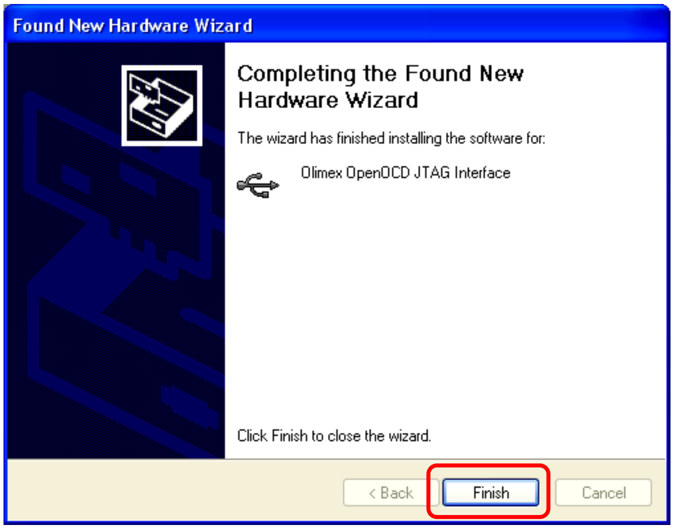
Found New Hardware Wizard

Please wait while the wizard installs the software...

Olimex OpenOCD JTAG Interface

Setting a system restore point and backing up old files in case your system needs to be restored in the future.

When the driver installation for the Olimex ARM-USB-OCD JTAG debugger is done, click on “**Finish**” on the screen shown below to exit the installer.



Found New Hardware Wizard

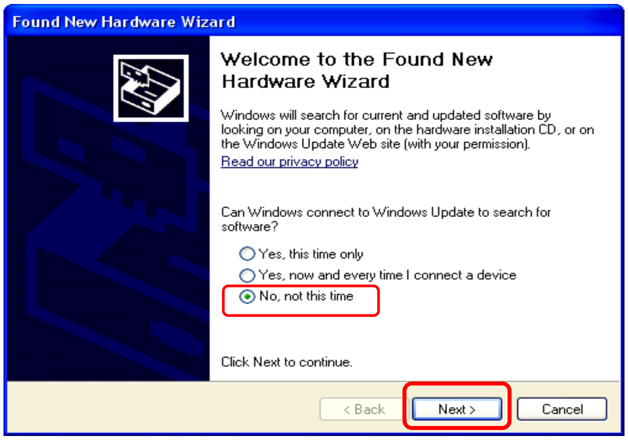
Completing the Found New Hardware Wizard

The wizard has finished installing the software for:

Olimex OpenOCD JTAG Interface

Click Finish to close the wizard.

Remember that the Olimex ARM-USB-OCD also supports a auxillary serial port. Windows will now start a dialog to install that virtual driver. Since we know exactly where the driver files are, click the radio button "**No, not this time**" on the window below left and click "**Next**" to continue. Also click the "**Install from a list or specific location (Advanced)**" radio button below on the right and then click "**Next**" to continue.



Found New Hardware Wizard

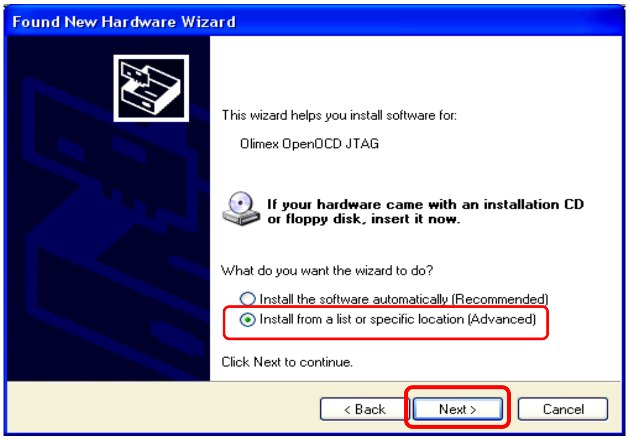
Welcome to the Found New Hardware Wizard

Windows will search for current and updated software by looking on your computer, on the hardware installation CD, or on the Windows Update Web site (with your permission).
Read our privacy policy

Can Windows connect to Windows Update to search for software?

○ Yes, this time only
○ Yes, now and every time I connect a device
● No, not this time

Click Next to continue.



Found New Hardware Wizard

This wizard helps you install software for:

Olimex OpenOCD JTAG

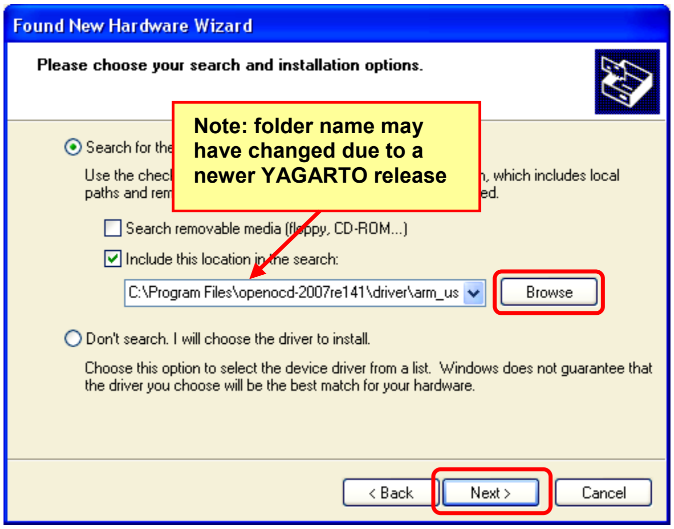
If your hardware came with an installation CD or floppy disk, insert it now.

What do you want the wizard to do?

○ Install the software automatically (Recommended)
● Install from a list or specific location (Advanced)

Click Next to continue.

Now use the "**Browse**" button to find the directory "**c:\Program Files\openocd-2007re141\driver\arm_usb_ocd\**" as shown below on the left hand screen. Click "**Next** to continue. Once again, ignore the Windows complaints about Logo testing and click "**Continue Anyway**" as shown below right.



Found New Hardware Wizard

Please choose your search and installation options.

● Search for the

Use the chec ..., which includes local paths and rem ...ed.

☐ Search removable media (floppy, CD-ROM...)

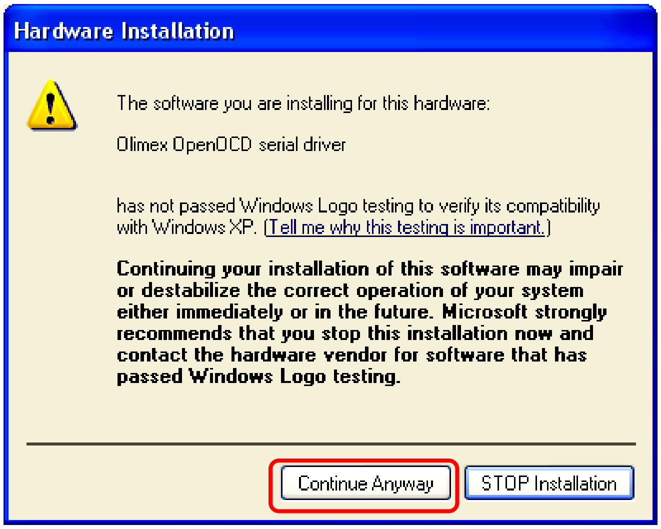
☑ Include this location in the search:

C:\Program Files\openocd-2007re141\driver\arm_us

○ Don't search. I will choose the driver to install.

Choose this option to select the device driver from a list. Windows does not guarantee that the driver you choose will be the best match for your hardware.

Note: folder name may have changed due to a newer YAGARTO release



Hardware Installation

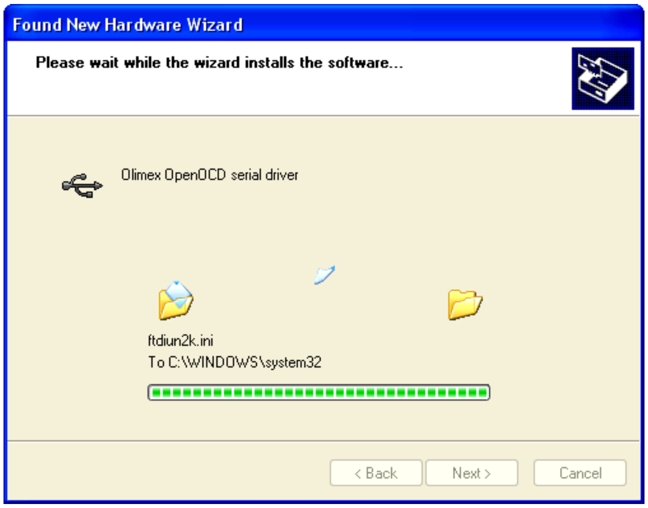
The software you are installing for this hardware:

Olimex OpenOCD serial driver

has not passed Windows Logo testing to verify its compatibility with Windows XP. (Tell me why this testing is important.)

Continuing your installation of this software may impair or destabilize the correct operation of your system either immediately or in the future. Microsoft strongly recommends that you stop this installation now and contact the hardware vendor for software that has passed Windows Logo testing.

The serial driver installs very rapidly. When the "Found New Hardware Wizard" screen reappears, click "**Finish**" to exit. Installation of the Olimex ARM-USB-OCD drivers is now completed.

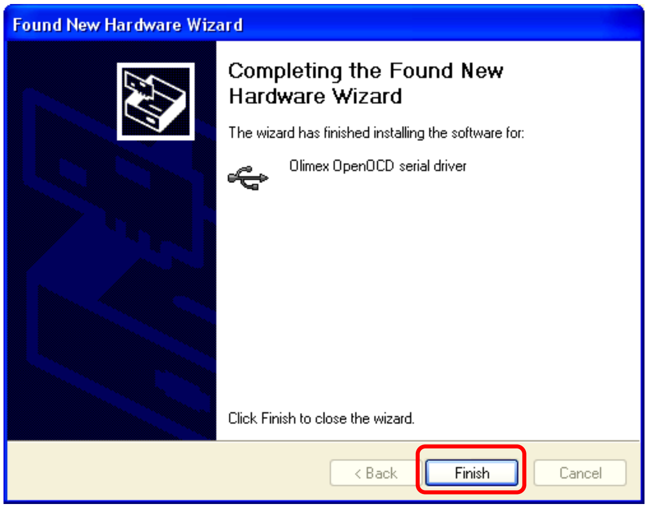


Found New Hardware Wizard

Please wait while the wizard installs the software...

Olimex OpenOCD serial driver

ftdiun2k.ini
To C:\WINDOWS\system32



Found New Hardware Wizard

Completing the Found New Hardware Wizard

The wizard has finished installing the software for:

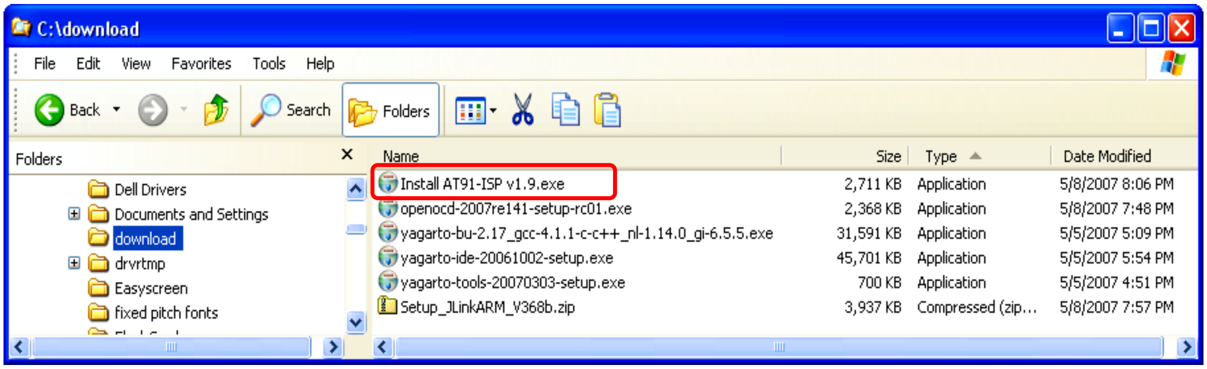
Olimex OpenOCD serial driver

Click Finish to close the wizard.

Install the Atmel SAM-BA Flash Programming Utility

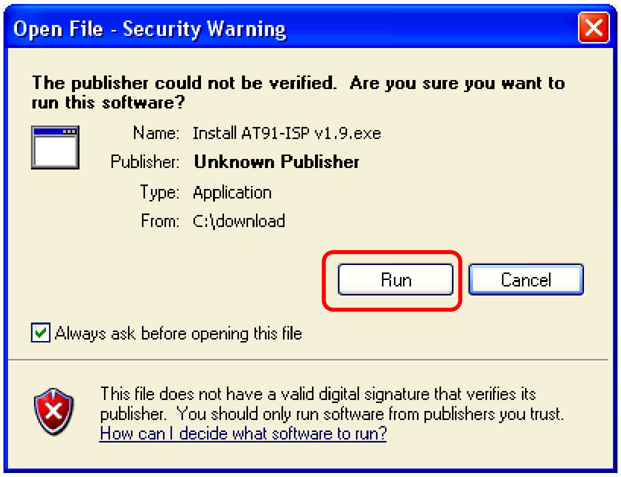
No matter what JTAG hardware interface you have purchased, it still behooves you to install and become familiar with the Atmel SAM-BA flash programming utility. It works with the COM port, the SAM-ICE USB-based JTAG interface or just a simple USB cable.

Click on "**Install AT91-ISP v1.9.exe**" in the c:\download folder, as shown below.



Name	Size	Type	Date Modified
Install AT91-ISP v1.9.exe	2,711 KB	Application	5/8/2007 8:06 PM
openocd-2007re141-setup-rc01.exe	2,368 KB	Application	5/8/2007 7:48 PM
yagarto-bu-2.17_gcc-4.1.1-c-c++_nl-1.14.0_gi-6.5.5.exe	31,591 KB	Application	5/5/2007 5:09 PM
yagarto-ide-20061002-setup.exe	45,701 KB	Application	5/5/2007 5:54 PM
yagarto-tools-20070303-setup.exe	700 KB	Application	5/5/2007 4:51 PM
Setup_JLinkARM_V368b.zip	3,937 KB	Compressed (zip...	5/8/2007 7:57 PM

Ignore the Windows belly-aching about software verification and click "**Run**" to start the installation as shown below left. When the setup wizard appears, click "**Next**" to continue as shown below right.

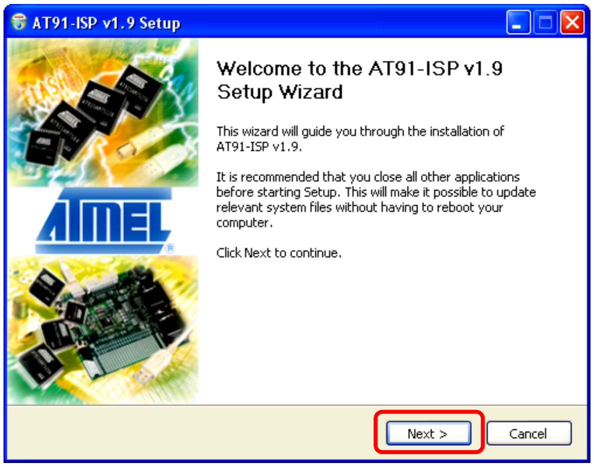

Open File - Security Warning

The publisher could not be verified. Are you sure you want to run this software?

Name: Install AT91-ISP v1.9.exe
Publisher: **Unknown Publisher**
Type: Application
From: C:\download

☑ Always ask before opening this file

This file does not have a valid digital signature that verifies its publisher. You should only run software from publishers you trust. How can I decide what software to run?


AT91-ISP v1.9 Setup

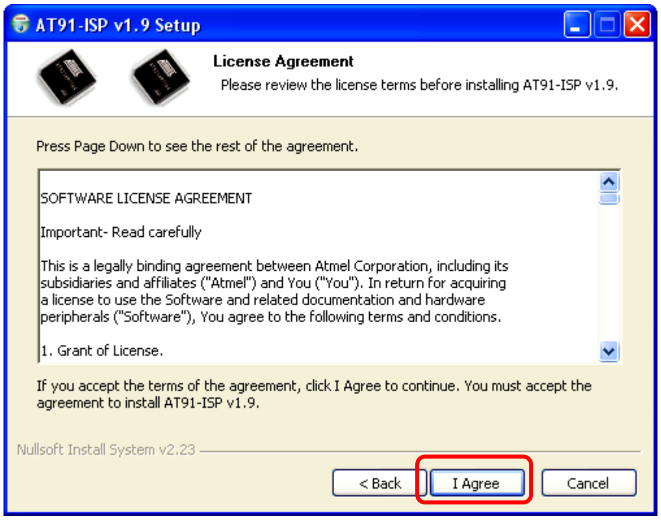
Welcome to the AT91-ISP v1.9 Setup Wizard

This wizard will guide you through the installation of AT91-ISP v1.9.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

In the two "License Agreement" screens below, click on "**I agree**" and "**Next** to continue.


AT91-ISP v1.9 Setup

License Agreement
Please review the license terms before installing AT91-ISP v1.9.

Press Page Down to see the rest of the agreement.

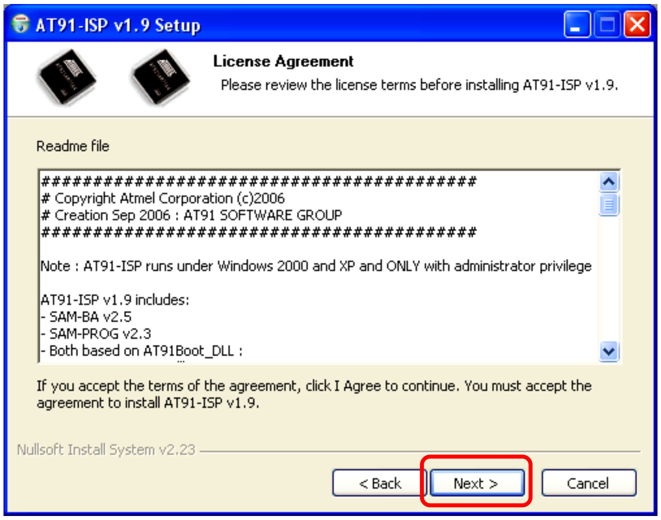
SOFTWARE LICENSE AGREEMENT

Important- Read carefully

This is a legally binding agreement between Atmel Corporation, including its subsidiaries and affiliates ("Atmel") and You ("You"). In return for acquiring a license to use the Software and related documentation and hardware peripherals ("Software"), You agree to the following terms and conditions.

1. Grant of License.

If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install AT91-ISP v1.9.


AT91-ISP v1.9 Setup

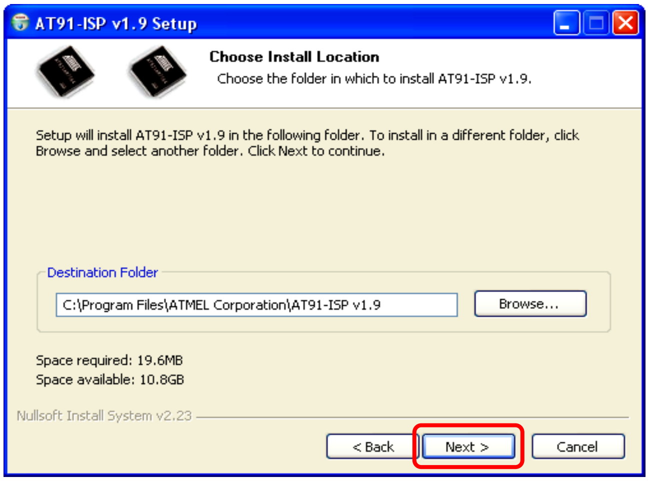
License Agreement
Please review the license terms before installing AT91-ISP v1.9.

Readme file

```
###########################################
# Copyright Atmel Corporation (c)2006
# Creation Sep 2006 : AT91 SOFTWARE GROUP
###########################################

Note : AT91-ISP runs under Windows 2000 and XP and ONLY with administrator privilege

AT91-ISP v1.9 includes:
- SAM-BA v2.5
- SAM-PROG v2.3
- Both based on AT91Boot_DLL :
```

If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install AT91-ISP v1.9.

Take the default install location by clicking “**Next**” below left. Also take the default start menu folder by clicking “**Install**” as shown below right.



AT91-ISP v1.9 Setup

Choose Install Location

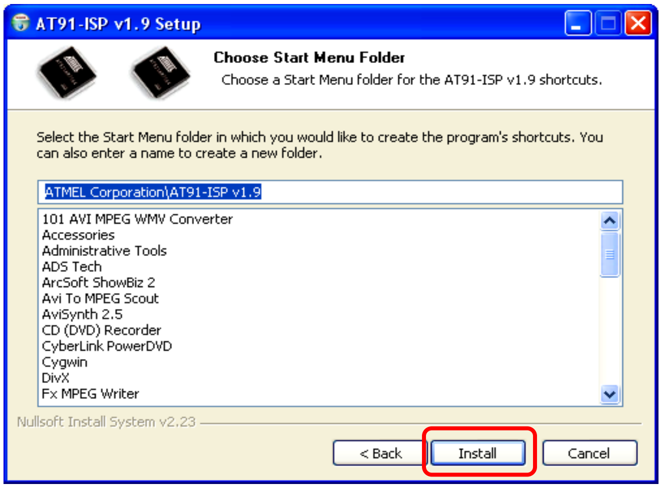
Choose the folder in which to install AT91-ISP v1.9.

Setup will install AT91-ISP v1.9 in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder: C:\Program Files\ATMEL Corporation\AT91-ISP v1.9

Space required: 19.6MB
Space available: 10.8GB

Nullsoft Install System v2.23



AT91-ISP v1.9 Setup

Choose Start Menu Folder

Choose a Start Menu folder for the AT91-ISP v1.9 shortcuts.

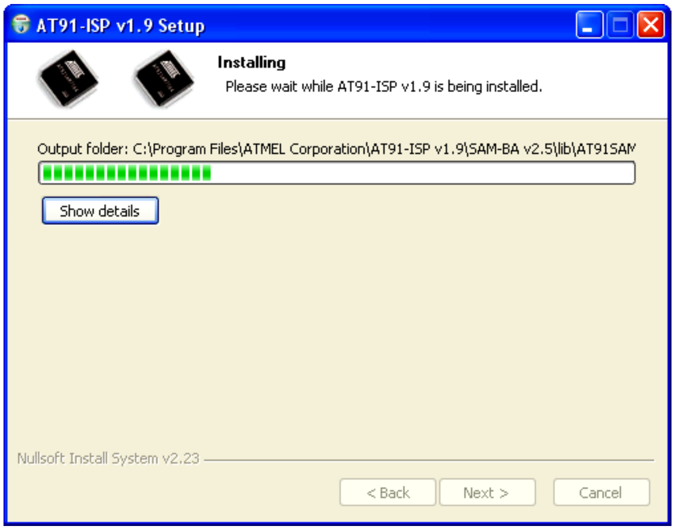
Select the Start Menu folder in which you would like to create the program's shortcuts. You can also enter a name to create a new folder.

ATMEL Corporation\AT91-ISP v1.9

101 AVI MPEG WMV Converter
Accessories
Administrative Tools
ADS Tech
ArcSoft ShowBiz 2
Avi To MPEG Scout
AviSynth 2.5
CD (DVD) Recorder
CyberLink PowerDVD
Cygwin
DivX
Fx MPEG Writer

Nullsoft Install System v2.23

When installation completes, click “**Next**” as shown below right to continue.



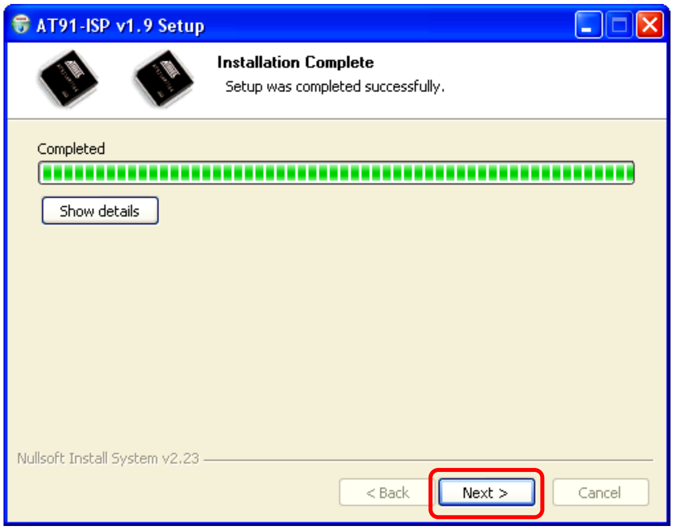
AT91-ISP v1.9 Setup

Installing

Please wait while AT91-ISP v1.9 is being installed.

Output folder: C:\Program Files\ATMEL Corporation\AT91-ISP v1.9\SAM-BA v2.5\lib\AT91SAM

Nullsoft Install System v2.23



AT91-ISP v1.9 Setup

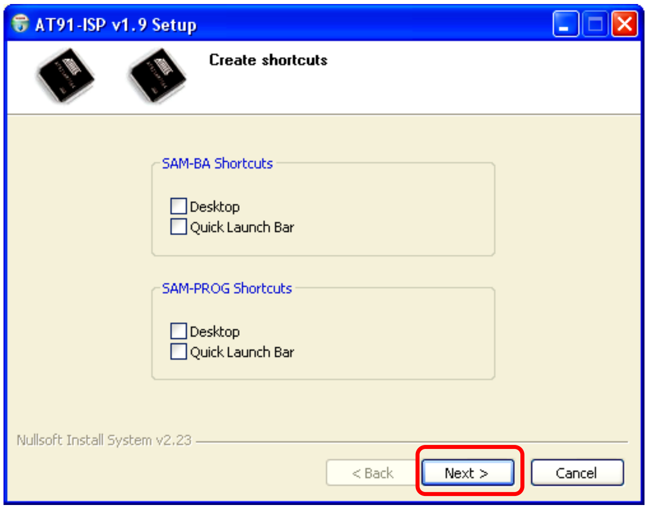
Installation Complete

Setup was completed successfully.

Completed

Nullsoft Install System v2.23

Since we will be starting SAM-BA from the Eclipse “Run” pull-down menu, **uncheck** all the shortcuts as shown on the screen below left and click “**Next**”. Finally, click “**Reboot now**” followed by “**Next**” to complete the installation. The SAM-BA utility is registered in the Windows registry and you need to re-boot your computer.



AT91-ISP v1.9 Setup

Create shortcuts

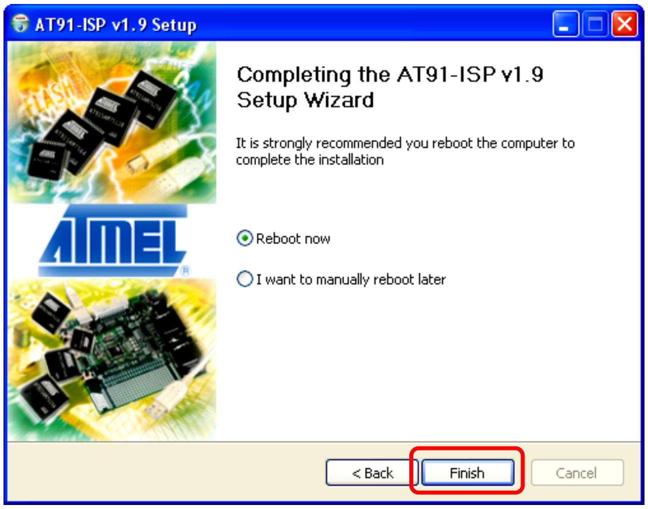
SAM-BA Shortcuts

□ Desktop
□ Quick Launch Bar

SAM-PROG Shortcuts

□ Desktop
□ Quick Launch Bar

Nullsoft Install System v2.23



AT91-ISP v1.9 Setup

Completing the AT91-ISP v1.9 Setup Wizard

It is strongly recommended you reboot the computer to complete the installation

● Reboot now

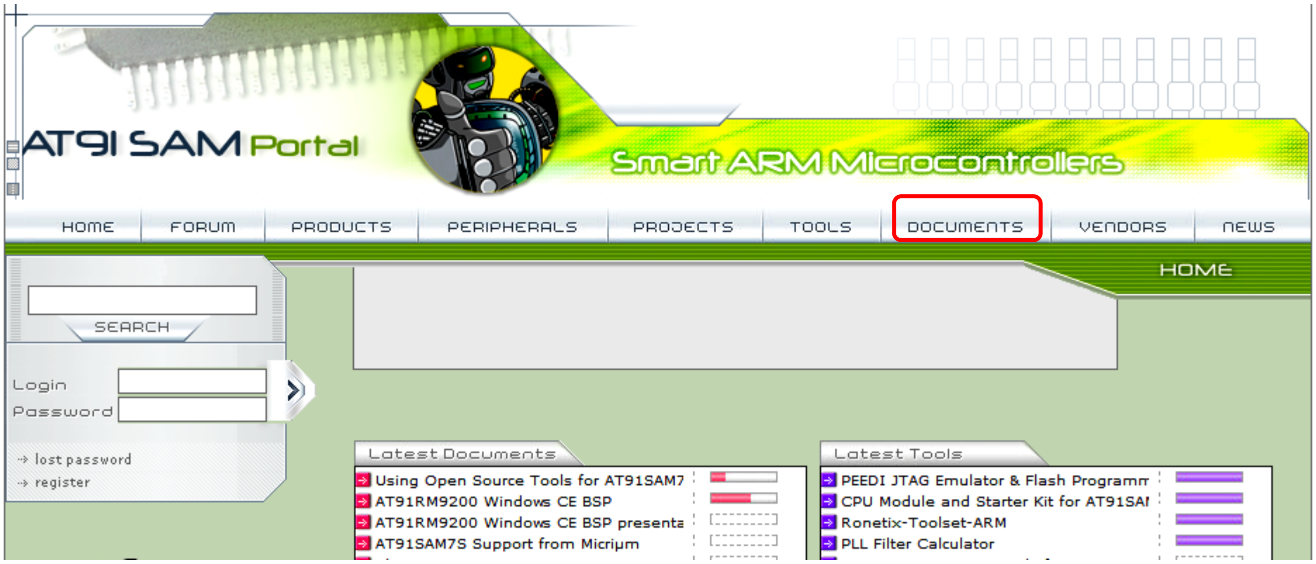
○ I want to manually reboot later

Download the Tutorial Sample Projects

Before we start up the Eclipse IDE, let’s first download the tutorial source and OpenOCD configuration files. This material may be downloaded from the Atmel ARM Product support site using this link:

http://www.at91.com

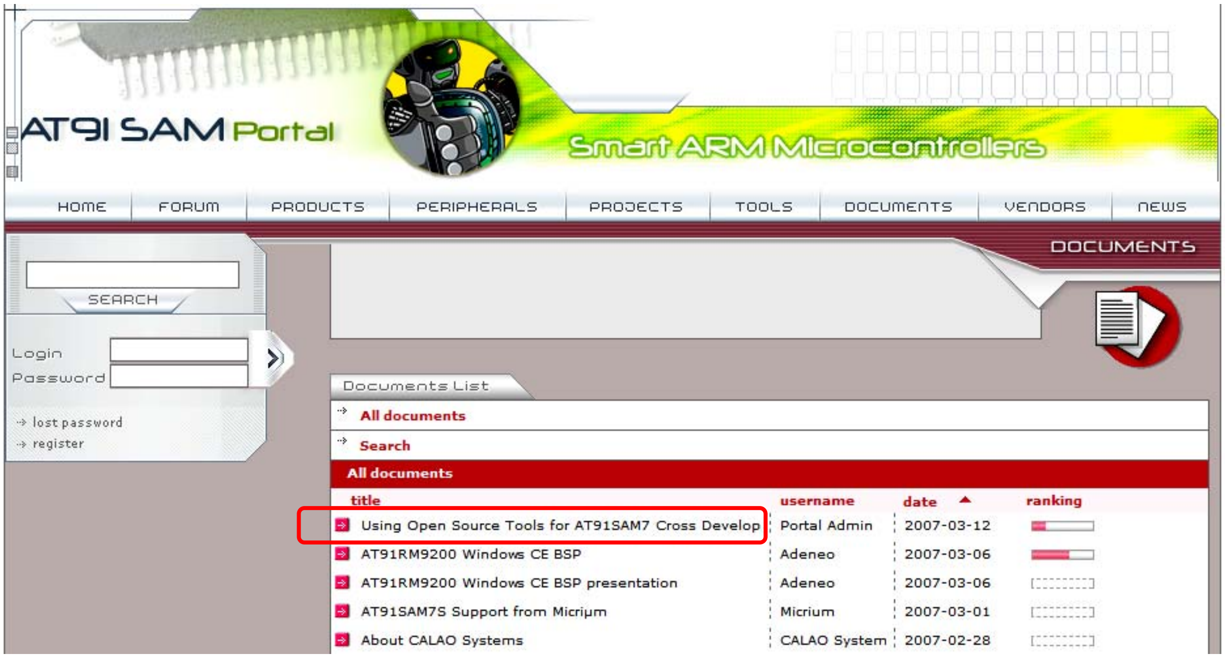
Click on “**Documents**” as shown below. If you are reading this tutorial, you have probably already done all this anyway.



Latest Documents
- Using Open Source Tools for AT91SAM7
- AT91RM9200 Windows CE BSP
- AT91RM9200 Windows CE BSP presenta
- AT91SAM7S Support from Micrium

Latest Tools
- PEEDI JTAG Emulator & Flash Programm
- CPU Module and Starter Kit for AT91SAI
- Ronetix-Toolset-ARM
- PLL Filter Calculator

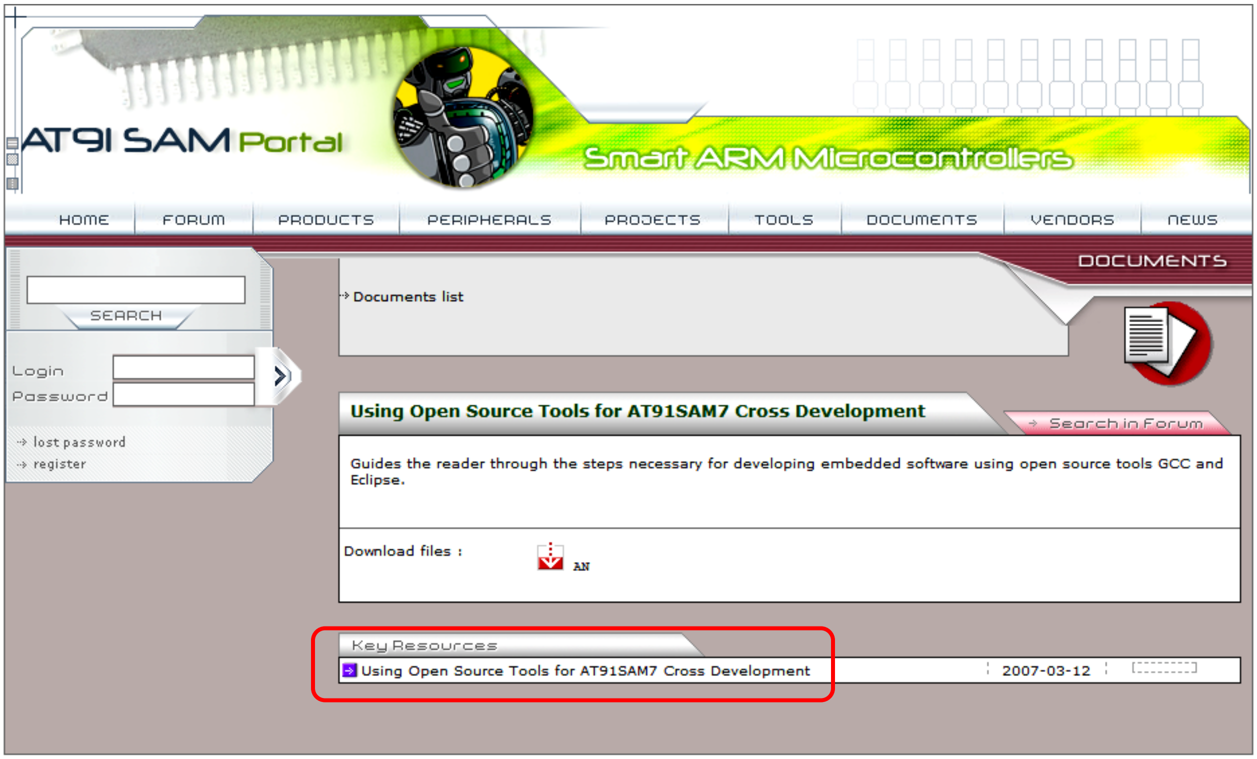
Now browse through the available documents until you see “**Using Open Source Tools for AT91SAM7 Cross Development**” and then click on it.



All documents

title	username	date	ranking
Using Open Source Tools for AT91SAM7 Cross Develop	Portal Admin	2007-03-12	
AT91RM9200 Windows CE BSP	Adeneo	2007-03-06	
AT91RM9200 Windows CE BSP presentation	Adeneo	2007-03-06	
AT91SAM7S Support from Micrium	Micrium	2007-03-01	
About CALAO Systems	CALAO System	2007-02-28	

Under the “Key Resources” tab, click on “**Using Open Source Tools for AT91SAM7 Cross Development**”; this will bring up the download for the tutorial in pdf format, the sample projects and OpenOCD configuration files. Clicking on just the “**AN**” icon will simply download and display a one page summary of the tutorial.



→ Documents list

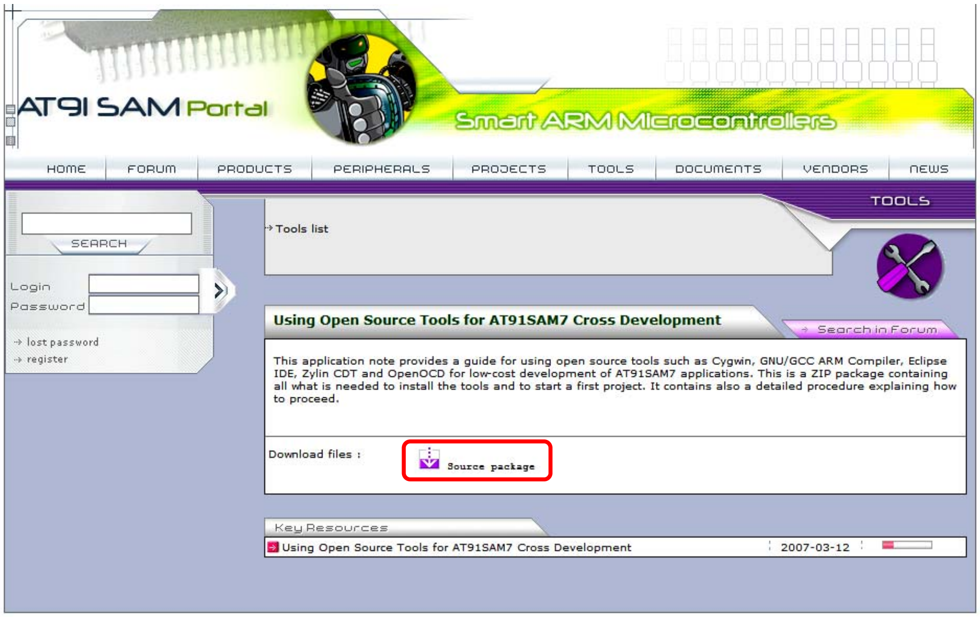
Using Open Source Tools for AT91SAM7 Cross Development

Guides the reader through the steps necessary for developing embedded software using open source tools GCC and Eclipse.

Download files : AN

Using Open Source Tools for AT91SAM7 Cross Development 2007-03-12

Now click on “**Source package**” as shown below to start the download.



→ Tools list

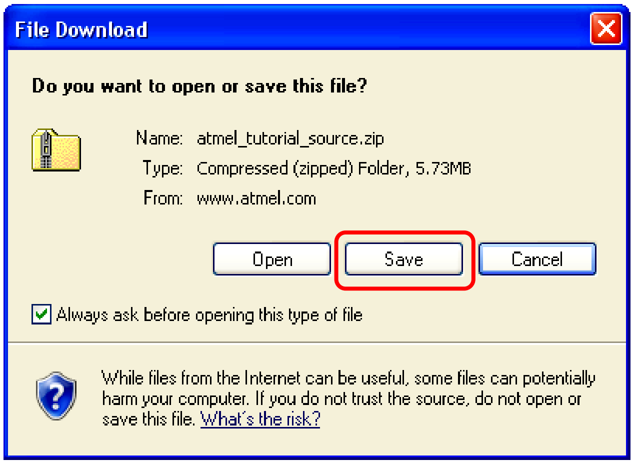
Using Open Source Tools for AT91SAM7 Cross Development

This application note provides a guide for using open source tools such as Cygwin, GNU/GCC ARM Compiler, Eclipse IDE, Zylin CDT and OpenOCD for low-cost development of AT91SAM7 applications. This is a ZIP package containing all what is needed to install the tools and to start a first project. It contains also a detailed procedure explaining how to proceed.

Download files : Source package

Using Open Source Tools for AT91SAM7 Cross Development 2007-03-12

Click on “**Save**” as shown below left and then select the “**c:\download**” folder as the destination in the “Save As” screen below right. Click “**Save**” to start the download of the sample code and configuration files.

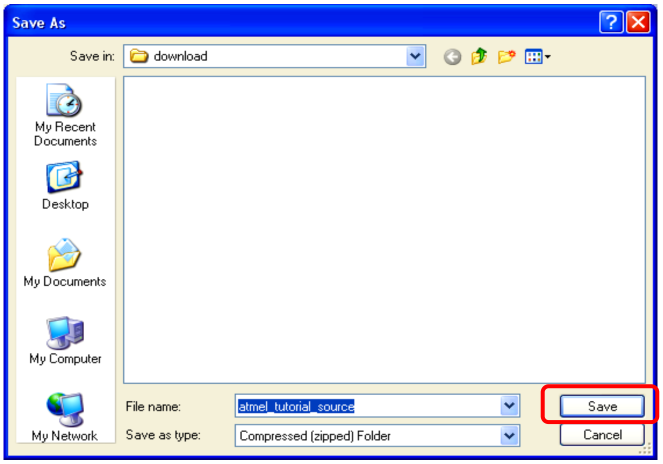

File Download

Do you want to open or save this file?

Name: atmel_tutorial_source.zip
Type: Compressed (zipped) Folder, 5.73MB
From: www.atmel.com

☑ Always ask before opening this type of file

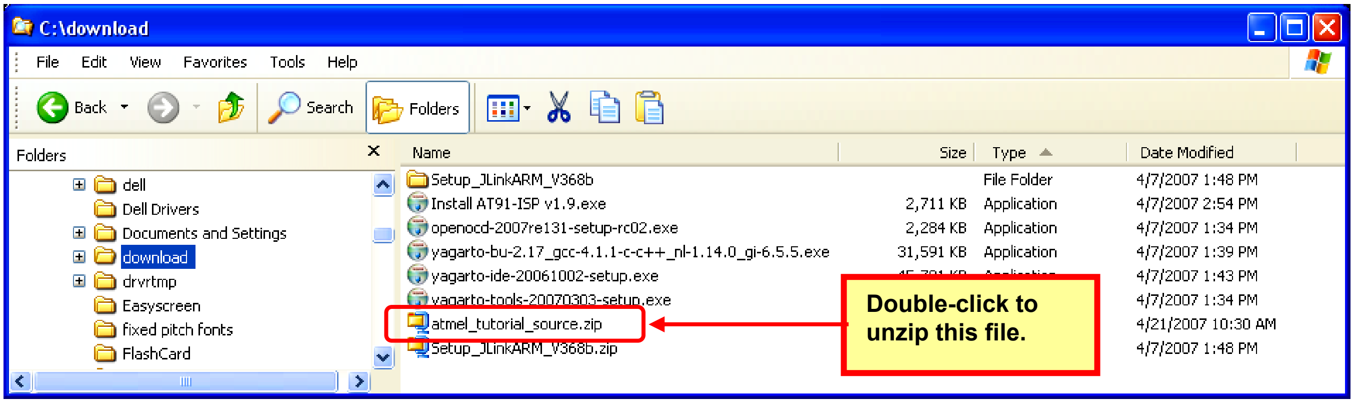
While files from the Internet can be useful, some files can potentially harm your computer. If you do not trust the source, do not open or save this file. What's the risk?


Save As

Save in: download

File name: atmel_tutorial_source
Save as type: Compressed (zipped) Folder

Now the c:\download folder shows the file “**c:\download\atmel_tutorial_source.zip**” as shown below.

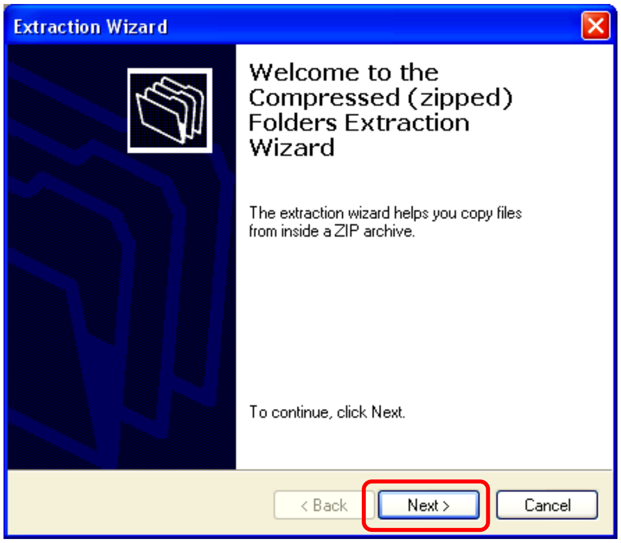

C:\download

Name	Size	Type	Date Modified
Setup_JLinkARM_V368b		File Folder	4/7/2007 1:48 PM
Install AT91-ISP v1.9.exe	2,711 KB	Application	4/7/2007 2:54 PM
openocd-2007re131-setup-rc02.exe	2,284 KB	Application	4/7/2007 1:34 PM
yagarto-bu-2.17_gcc-4.1.1-c-c++_nl-1.14.0_gi-6.5.5.exe	31,591 KB	Application	4/7/2007 1:39 PM
yagarto-ide-20061002-setup.exe	[illegible]	[illegible]	4/7/2007 1:43 PM
yagarto-tools-20070303-setup.exe	[illegible]	[illegible]	4/7/2007 1:34 PM
atmel_tutorial_source.zip	[illegible]	[illegible]	4/21/2007 10:30 AM
Setup_JLinkARM_V368b.zip	[illegible]	[illegible]	4/7/2007 1:48 PM

Double-click to unzip this file.

Double-click on the file “**c:\download\atmel_tutorial_source.zip**” to start the Windows file decompression facility.

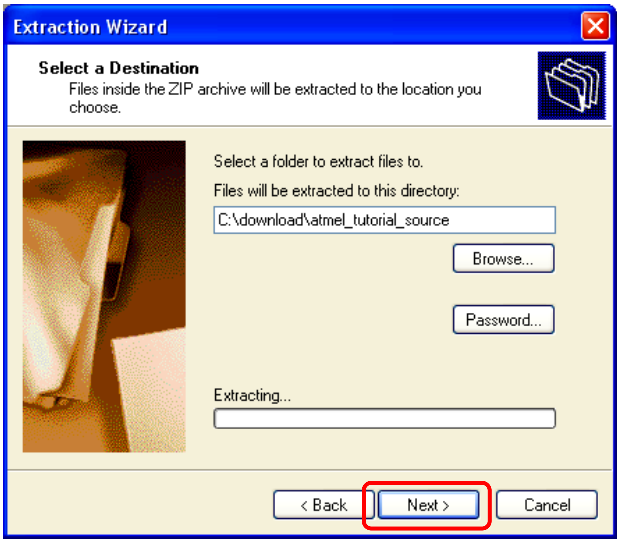
The Windows Compressed Folders Extraction Wizard will start as shown below on the left. Click on “**Next**” to start the wizard. Take the default destination folder as shown below right and click “**Next** to proceed.


Extraction Wizard

Welcome to the Compressed (zipped) Folders Extraction Wizard

The extraction wizard helps you copy files from inside a ZIP archive.

To continue, click Next.


Extraction Wizard

Select a Destination

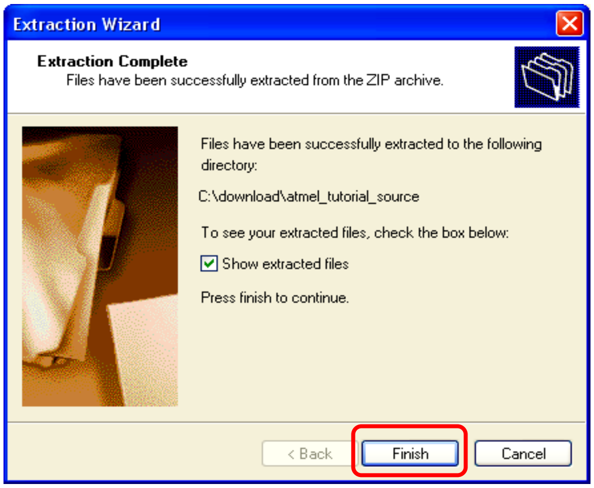
Files inside the ZIP archive will be extracted to the location you choose.

Select a folder to extract files to.

Files will be extracted to this directory:
C:\download\atmel_tutorial_source

Extracting...

The file decompression will finish in a few seconds; click “**Finish**” to complete the unzipping of the tutorial components as shown below.



Extraction Wizard

Extraction Complete

Files have been successfully extracted from the ZIP archive.

Files have been successfully extracted to the following directory:

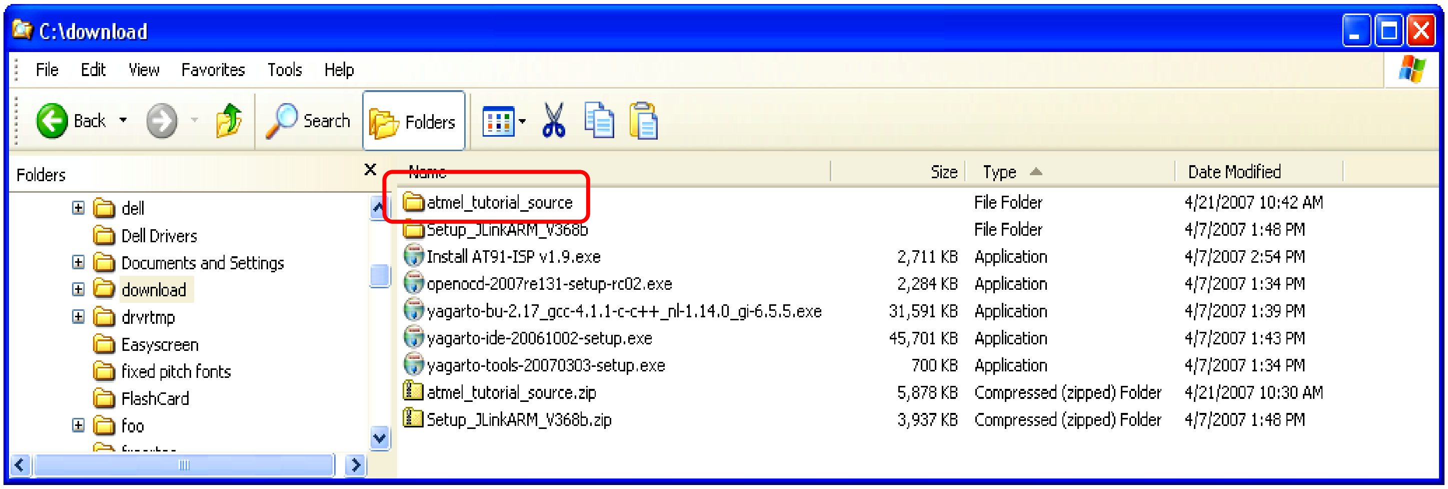
C:\download\atmel_tutorial_source

To see your extracted files, check the box below:

☑ Show extracted files

Press finish to continue.

Inspecting the “c:\download” folder, we see a sub folder “**c:/download/atmel_tutorial_source/”.**



C:\download

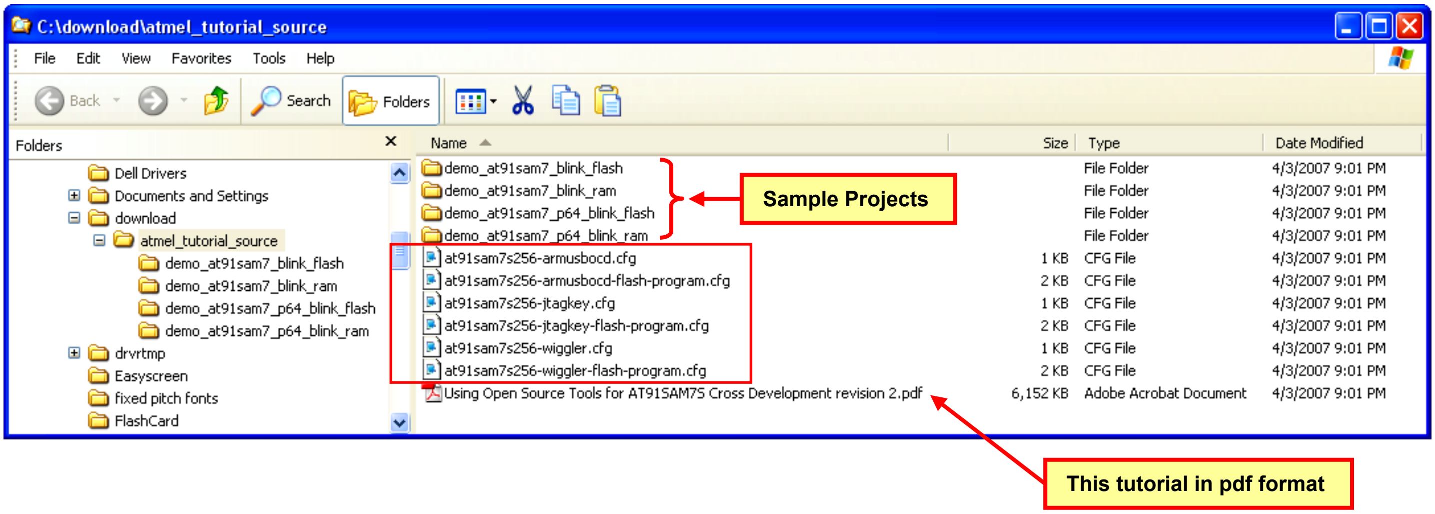
- ⊞ dell
- Dell Drivers
- ⊞ Documents and Settings
- ⊞ download
- ⊞ drvrtmp
- Easyscreen
- fixed pitch fonts
- FlashCard
- ⊞ foo

Name	Size	Type	Date Modified
atmel_tutorial_source		File Folder	4/21/2007 10:42 AM
Setup_JLinkARM_V368b		File Folder	4/7/2007 1:48 PM
Install AT91-ISP v1.9.exe	2,711 KB	Application	4/7/2007 2:54 PM
openocd-2007re131-setup-rc02.exe	2,284 KB	Application	4/7/2007 1:34 PM
yagarto-bu-2.17_gcc-4.1.1-c-c++_nl-1.14.0_gi-6.5.5.exe	31,591 KB	Application	4/7/2007 1:39 PM
yagarto-ide-20061002-setup.exe	45,701 KB	Application	4/7/2007 1:43 PM
yagarto-tools-20070303-setup.exe	700 KB	Application	4/7/2007 1:34 PM
atmel_tutorial_source.zip	5,878 KB	Compressed (zipped) Folder	4/21/2007 10:30 AM
Setup_JLinkARM_V368b.zip	3,937 KB	Compressed (zipped) Folder	4/7/2007 1:48 PM

There are four sample projects. Two are for the Atmel AT91SAM7-EK evaluation board and two are for the Olimex SAM7-P64 board. We will be “importing” these projects into Eclipse very shortly, so make a mental note of the folder where you stored them.

In the sample folder below, there are six OpenOCD configuration files with the extension “**.cfg**”. There are two configuration files for the wiggler, two for the Amontec JTAGKey, and two for the Olimex ARM-USB-OCD device. With respect to each hardware device, one configuration file is for debugging whilst one is for on chip flash memory programming. We will be copying the configuration files into the OpenOCD bin folder shortly so that OpenOCD can access them easily.

Finally the sample folder contains the tutorial itself in pdf format.



Name	Size	Type	Date Modified
demo_at91sam7_blink_flash		File Folder	4/3/2007 9:01 PM
demo_at91sam7_blink_ram		File Folder	4/3/2007 9:01 PM
demo_at91sam7_p64_blink_flash		File Folder	4/3/2007 9:01 PM
demo_at91sam7_p64_blink_ram		File Folder	4/3/2007 9:01 PM
at91sam7s256-armusbocd.cfg	1 KB	CFG File	4/3/2007 9:01 PM
at91sam7s256-armusbocd-flash-program.cfg	2 KB	CFG File	4/3/2007 9:01 PM
at91sam7s256-jtagkey.cfg	1 KB	CFG File	4/3/2007 9:01 PM
at91sam7s256-jtagkey-flash-program.cfg	2 KB	CFG File	4/3/2007 9:01 PM
at91sam7s256-wiggler.cfg	1 KB	CFG File	4/3/2007 9:01 PM
at91sam7s256-wiggler-flash-program.cfg	2 KB	CFG File	4/3/2007 9:01 PM
Using Open Source Tools for AT91SAM7S Cross Development revision 2.pdf	6,152 KB	Adobe Acrobat Document	4/3/2007 9:01 PM

Sample Projects

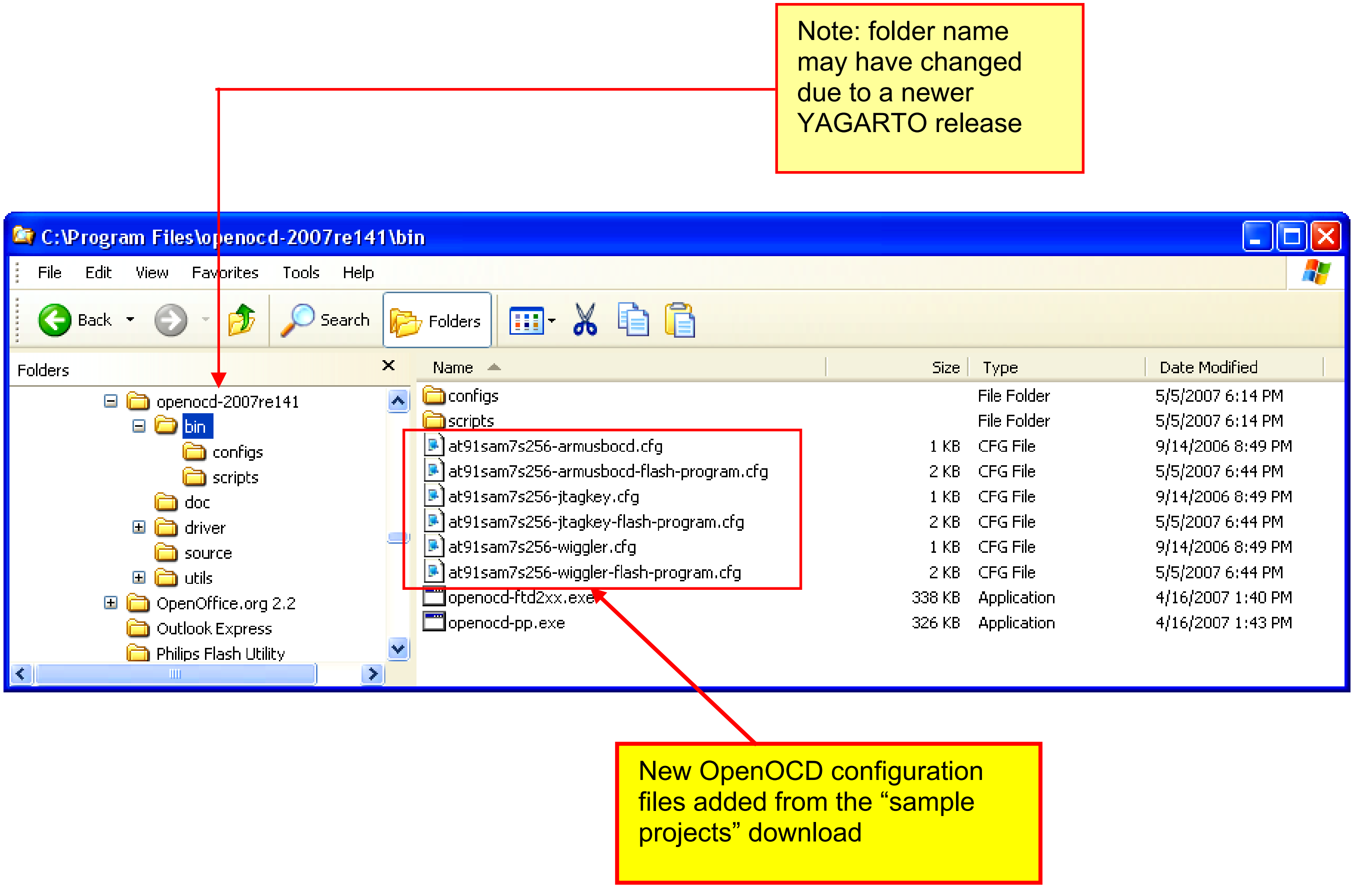
This tutorial in pdf format

Move the OpenOCD Configuration Files

Using Windows Explorer, select and move the six OpenOCD configuration files shown above into the "**c:\Program Files\openocd-2007re141\bin"** folder. These configuration files will be used by the sample projects later in the tutorial. Additionally, this destination folder already has a Windows path defined for it and thus simplifies setting up the OpenOCD as an external tool.

If you have downloaded a newer revision of YAGARTO, the destination folder will change. Make sure that you take this into account!

The OpenOCD folder should now look as shown below.



Name	Size	Type	Date Modified
configs		File Folder	5/5/2007 6:14 PM
scripts		File Folder	5/5/2007 6:14 PM
at91sam7s256-armusbocd.cfg	1 KB	CFG File	9/14/2006 8:49 PM
at91sam7s256-armusbocd-flash-program.cfg	2 KB	CFG File	5/5/2007 6:44 PM
at91sam7s256-jtagkey.cfg	1 KB	CFG File	9/14/2006 8:49 PM
at91sam7s256-jtagkey-flash-program.cfg	2 KB	CFG File	5/5/2007 6:44 PM
at91sam7s256-wiggler.cfg	1 KB	CFG File	9/14/2006 8:49 PM
at91sam7s256-wiggler-flash-program.cfg	2 KB	CFG File	5/5/2007 6:44 PM
openocd-ftd2xx.exe	338 KB	Application	4/16/2007 1:40 PM
openocd-pp.exe	326 KB	Application	4/16/2007 1:43 PM

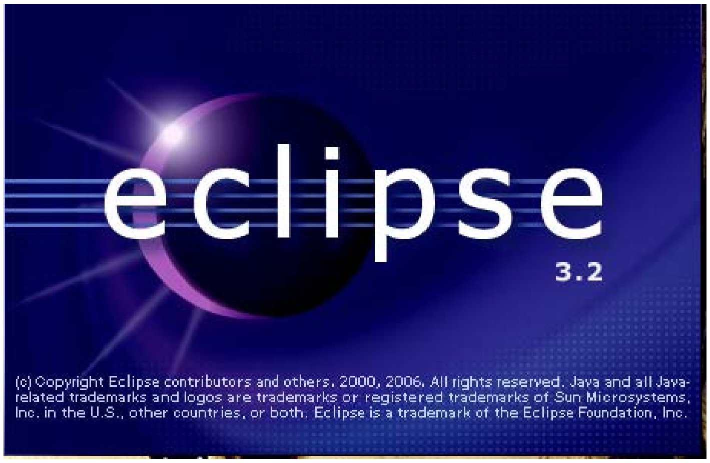
Note: folder name may have changed due to a newer YAGARTO release

New OpenOCD configuration files added from the "sample projects" download

Running Eclipse for the First Time

The Yagarto installer creates a desktop icon for starting Eclipse, as shown below. Click on this icon to start the Eclipse IDE.

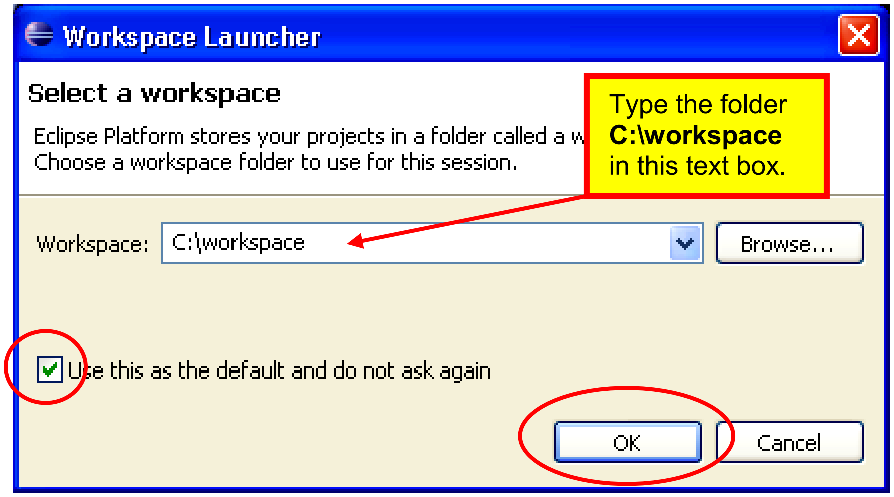
Now the Eclipse splash screen will open up, as shown below.



(c) Copyright Eclipse contributors and others. 2000, 2006. All rights reserved. Java and all Java-related trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S., other countries, or both. Eclipse is a trademark of the Eclipse Foundation, Inc.

At this point, Eclipse will present a "Workspace Launcher" dialog, shown below. This is where you specify the location of the "workspace" that will hold your Eclipse/CDT projects. You may place the workspace anywhere you wish but for this tutorial I placed it in the root folder as "C:\workspace".

Click the check box so the folder "**C:\workspace**" can be assigned to be the default anytime you enter Eclipse. Click "**OK**" to accept the workspace assignment and continue with Eclipse start-up.



Workspace Launcher

Select a workspace

Eclipse Platform stores your projects in a folder called a w

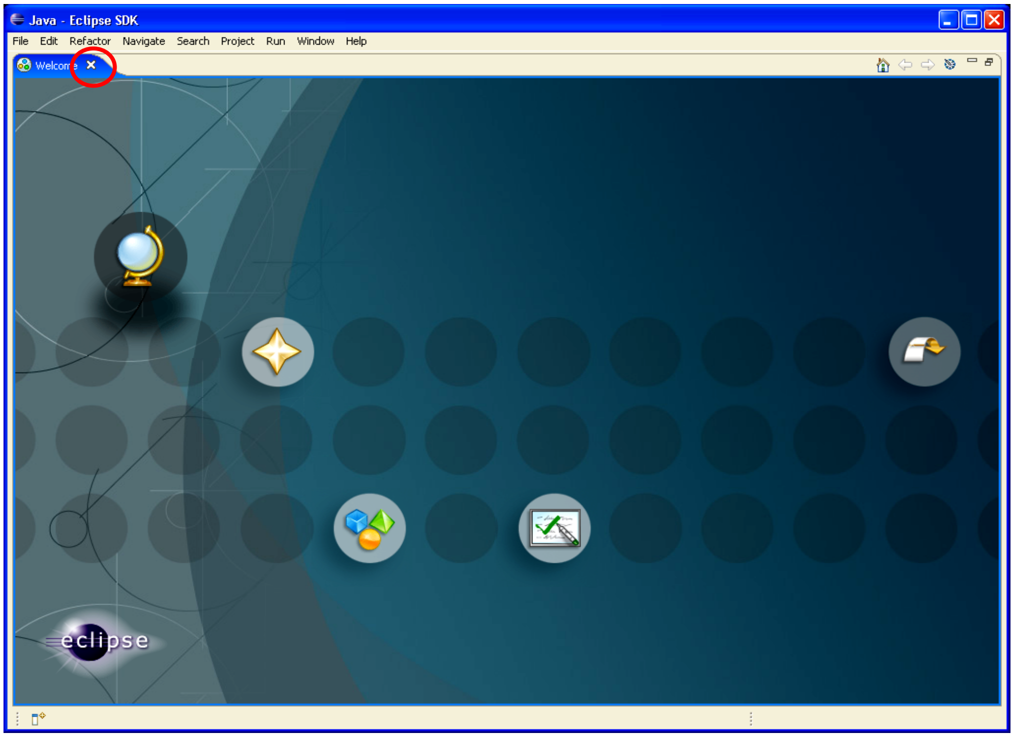
Choose a workspace folder to use for this session.

Workspace: C:\workspace

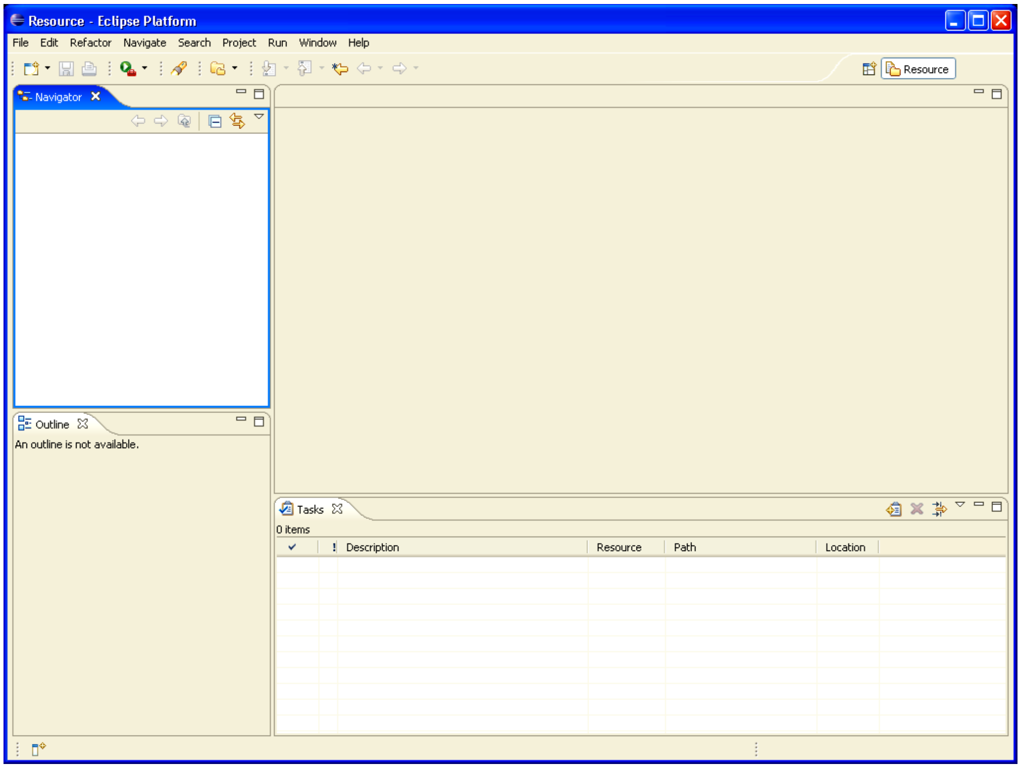
☑ Use this as the default and do not ask again

Type the folder **C:\workspace** in this text box.

Now Eclipse will officially start and show the “Welcome” page. Since most of the informational icons refer to the JAVA aspects of Eclipse, discard the “welcome” screen by clicking on the “**X**” as shown below.



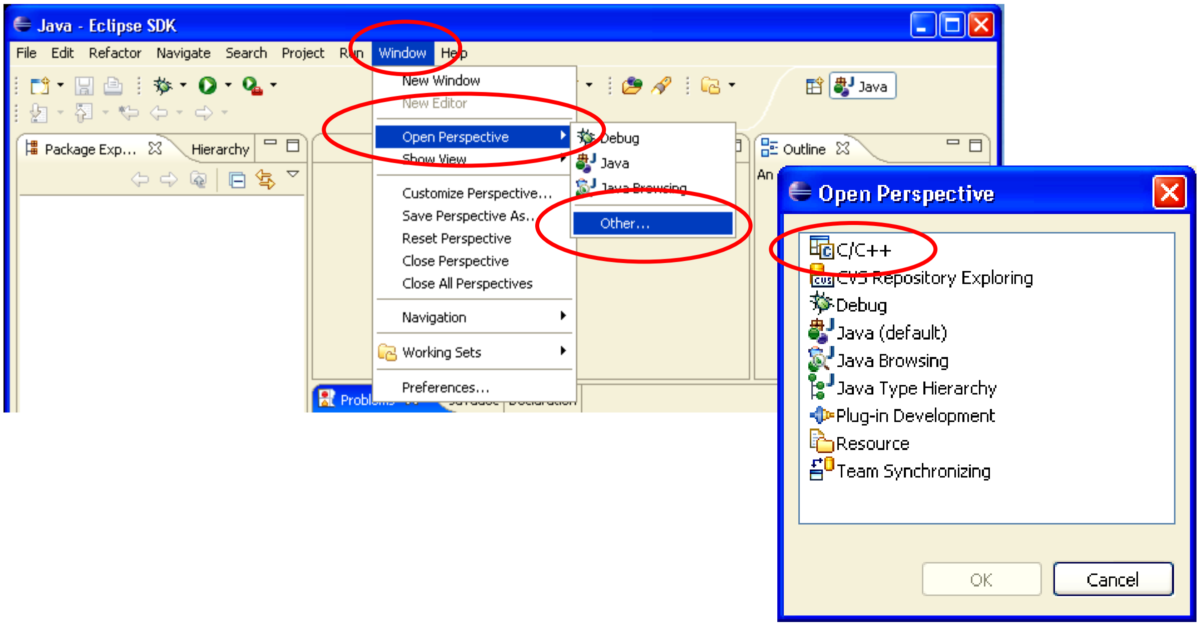
What follows is the “Resource” perspective. A perspective is simply a layout of “views” on the display surface (the Resource perspective includes “Navigator”, “Editor”, “Outline” and “Tasks” views.


Resource - Eclipse Platform
File Edit Refactor Navigate Search Project Run Window Help
Resource
Navigator
Outline
Tasks
Description
Resource
Path
Location

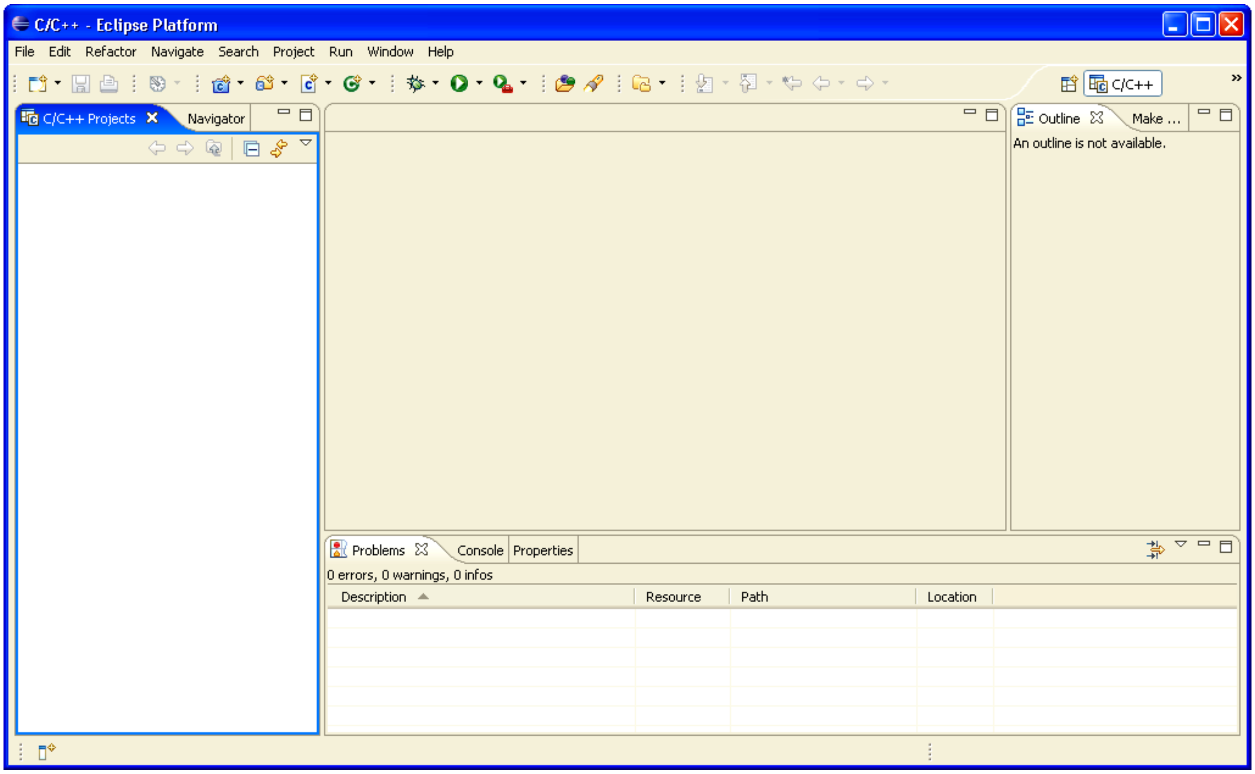
An outline is not available.

0 items

Let’s switch to the C/C++ perspective. Click on “**Window – Open Perspective – Other…**”, then click on “**C/C++**” to open Eclipse into the C/C++ perspective.



This is the C/C++ perspective. We will be learning more about the various component parts later in this tutorial.



An outline is not available.

0 errors, 0 warnings, 0 infos

Set Up Eclipse External Tools

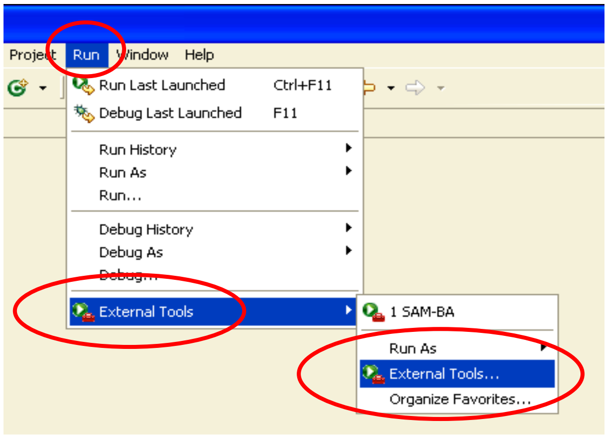
We have installed on our desktop PC several tools; such as the OpenOCD or the J-Link GDB Server and the SAM-BA flash programming utility. We would like to have a convenient way to start these tools from the Eclipse screen. Eclipse has just such a facility – it's called Eclipse "External Tools". The tools installed this way can be conveniently started from the "Run" pull-down menu or via a toolbar button.

Set Up OpenOCD as an Eclipse External Tool (wiggler)

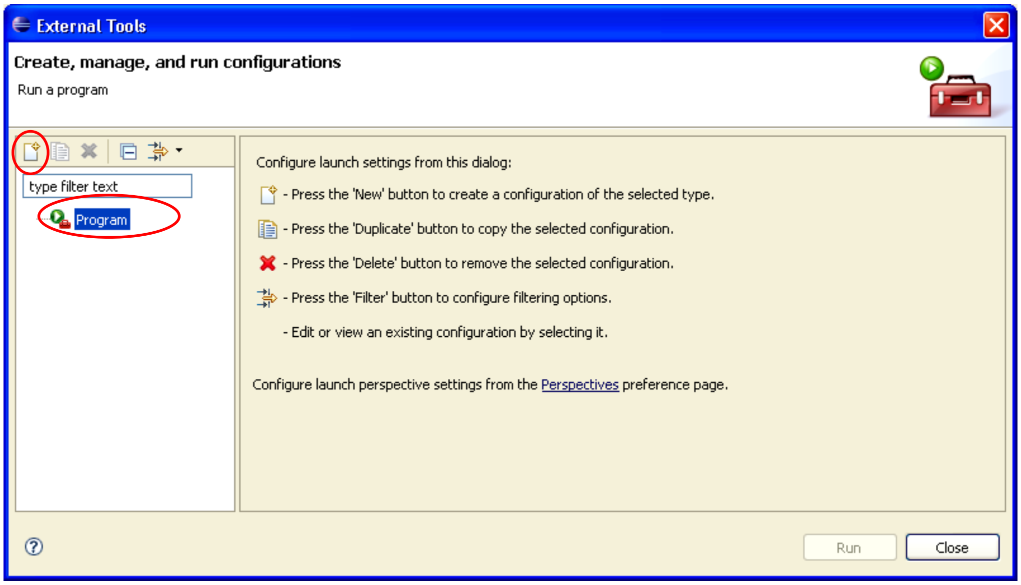
If you have purchased an Olimex ARM-JTAG (wiggler), you need to set up OpenOCD as an external tool and tailor it specifically for operation with the "wiggler".

When it's time to debug an application, we must be able to conveniently start the OpenOCD debugger. OpenOCD runs as a daemon; a program that runs in the background waiting for commands to be submitted to it. Eclipse has a very nice "external tool" feature that allows us to add OpenOCD to the RUN pull-down menu.

Click on "**Run – External Tools – External Tools…**"



The "**External Tools**" window will appear. Click on "**Program**" and then "**New**" button to establish a new External Tool.



Create, manage, and run configurations

Run a program

Configure launch settings from this dialog:

- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.

Configure launch perspective settings from the Perspectives preference page.

Fill out the “**External Tools**” form exactly as shown below.

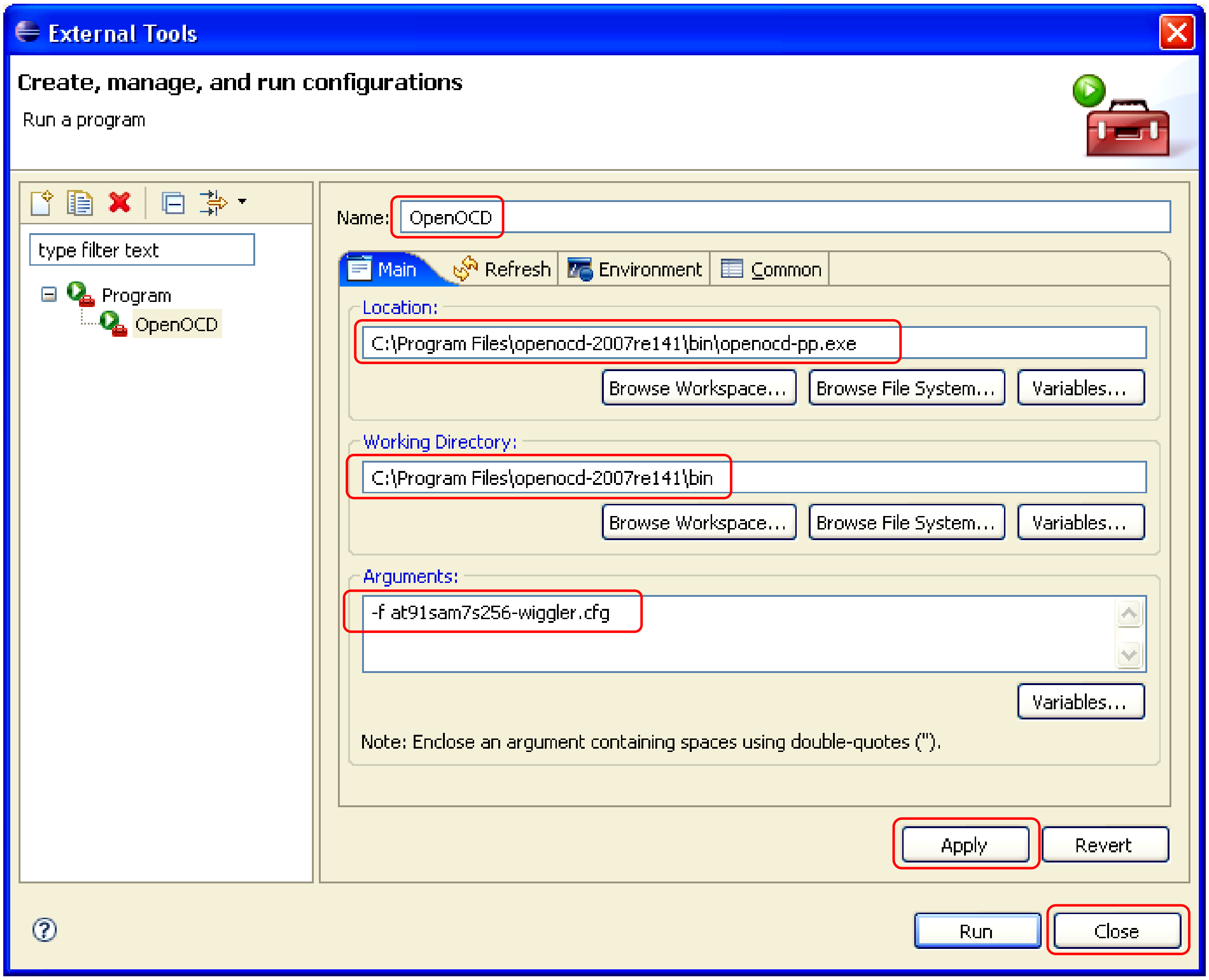
In the “Name” text box, call this external tool “**OpenOCD**”

There are two versions of OpenOCD; **openocd-pp.exe** supports the parallel-port “wiggler” device (ARM-JTAG from Olimex) while **openocd-ftd2xx.exe** supports the USB-based devices from Amontec and Olimex. In this section we are installing the “wiggler” version of OpenOCD as an Eclipse external tool.

In the “Location:” pane, use the “**Browse File System…**” button to search for the OpenOCD executable; it will be in this folder: **c:\Program Files\openocd-2007re141\bin\openocd-pp.exe**”.

In the “Working Directory” pane, use the “**Browse File System…**” button to specify “**c:\Program Files\openocd-2007re131\bin\**” as the working directory.

In the “Arguments” pane, enter the argument “**-f at91sam7s256-wiggler.cfg**” to specify the OpenOCD configuration file designed for the wiggler. Remember that we copied the six OpenOCD configuration files into the “c:\Program Files\openocd-2007re131\bin\” earlier. In this case, we need the “wiggler” version.



Name: OpenOCD

Location: C:\Program Files\openocd-2007re141\bin\openocd-pp.exe

Working Directory: C:\Program Files\openocd-2007re141\bin

Arguments: -f at91sam7s256-wiggler.cfg

Note: Enclose an argument containing spaces using double-quotes (").

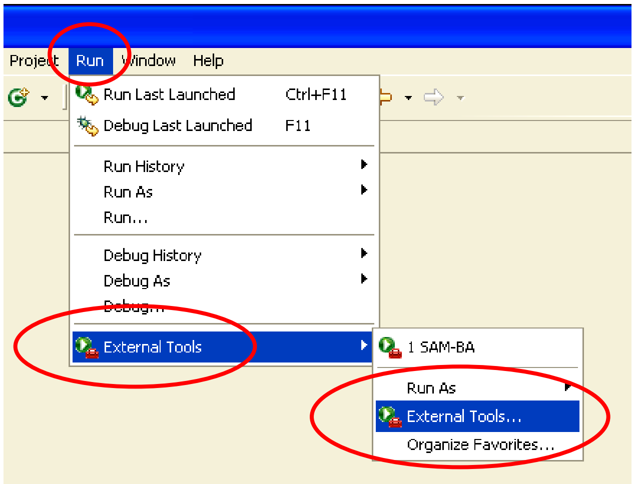
No changes are required to the other tabs in the form (Refresh, Environment, and Common).
Click on “**Apply**” and “**Close**” to register **OpenOCD** as an external tool.

Set Up OpenOCD as an Eclipse External Tool (ARM-USB-OCD)

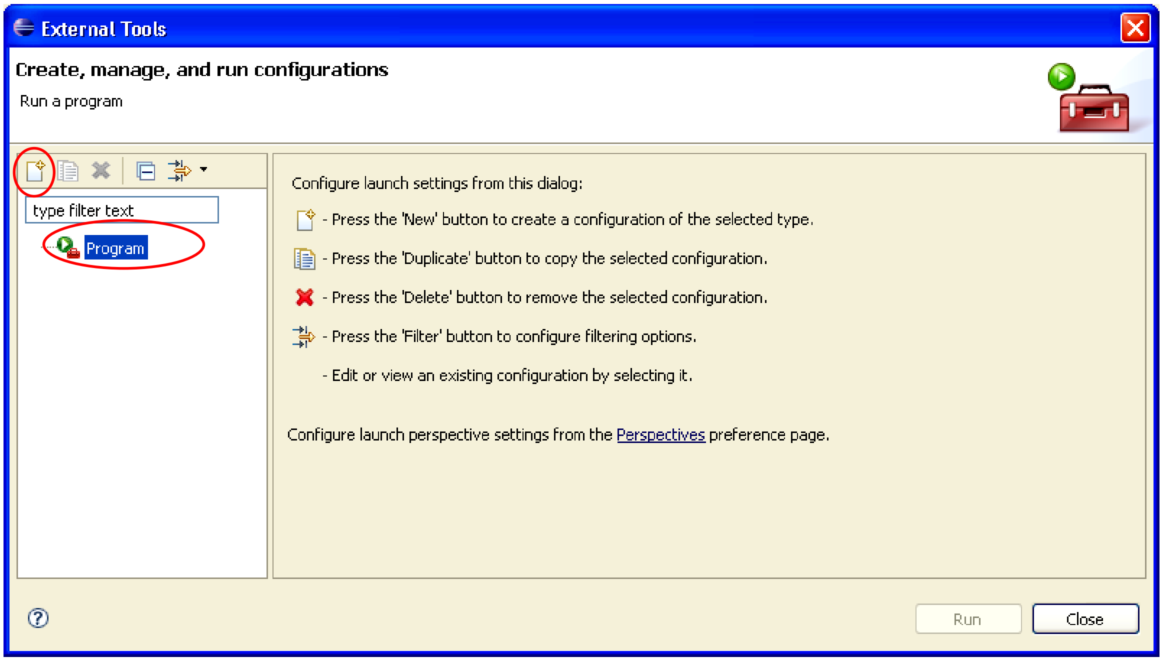
If you have purchased an Olimex ARM-USB-JTAG, you need to set up OpenOCD as an external tool and tailor it specifically for operation with the Olimex ARM-USB-OCD JTAG interface.

When it's time to debug an application, we must be able to conveniently start the OpenOCD debugger. OpenOCD runs as a daemon; a program that runs in the background waiting for commands to be submitted to it. Eclipse has a very nice "external tool" feature that allows us to add OpenOCD to the RUN pull-down menu.

Click on "**Run – External Tools – External Tools…**"



The "**External Tools**" window will appear. Click on "**Program**" and then "**New**" button to establish a new External Tool.



Configure launch settings from this dialog:

- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.

Configure launch perspective settings from the Perspectives preference page.

Fill out the “**External Tools**” form exactly as shown below.

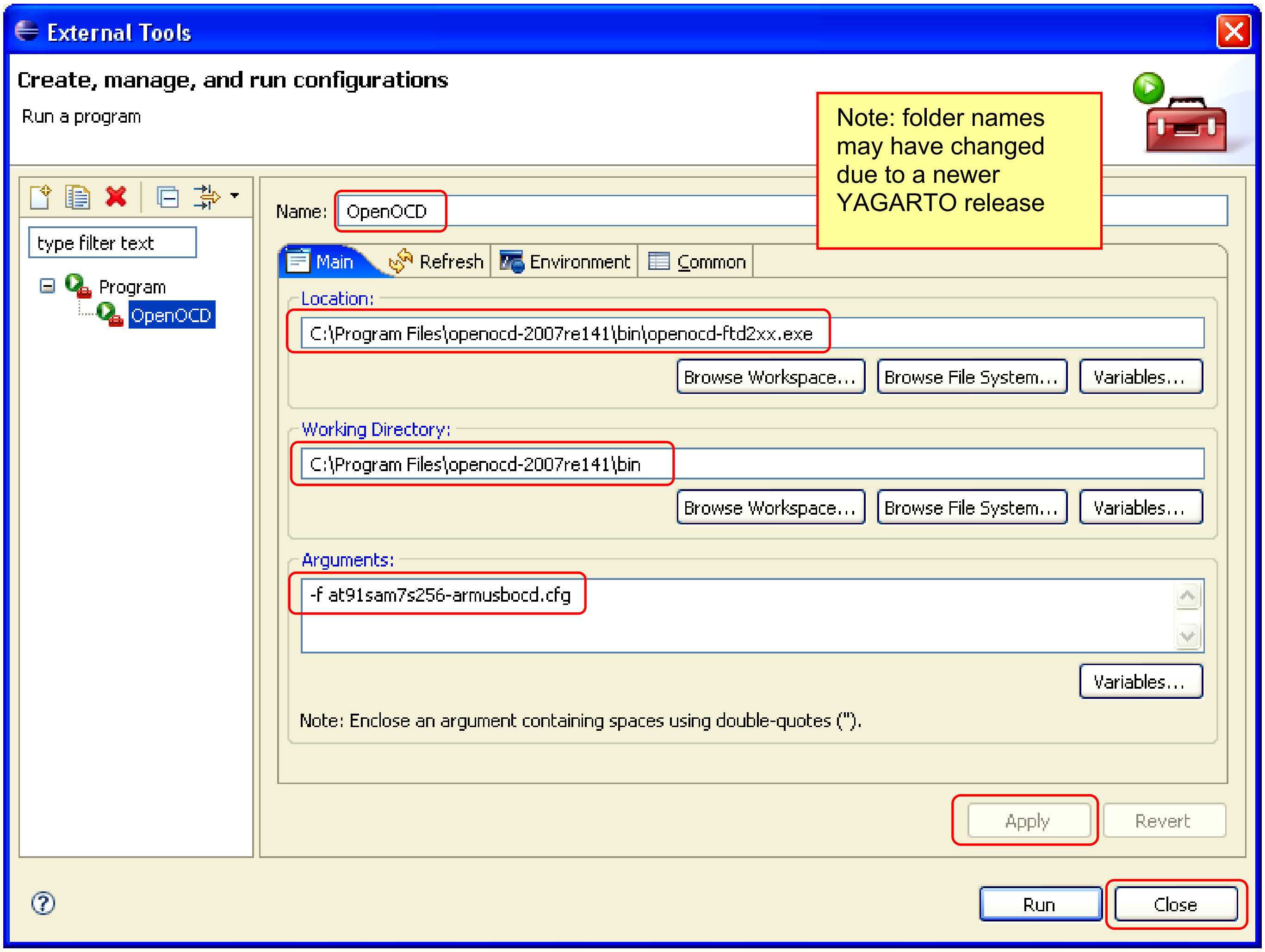
In the “Name” text box, call this external tool “**OpenOCD**”

There are two versions of OpenOCD; **openocd-pp.exe** supports the parallel-port “wiggler” device (ARM-JTAG from Olimex) while **openocd-ftd2xx.exe** supports the USB-based devices from Amontec and Olimex. In this section we are installing the “USB” version of OpenOCD as an Eclipse external tool.

In the “Location:” pane, use the “**Browse File System…**” button to search for the OpenOCD executable; it will be in this folder: **c:\Program Files\openocd-2007re131\bin\openocd-ftd2xx.exe**”.

In the “Working Directory” pane, use the “**Browse File System…**” button to specify “**c:\Program Files\openocd-2007re131\bin\**” as the working directory.

In the “Arguments” pane, enter the argument “**-f at91sam7s256-armusbocd.cfg**” to specify the OpenOCD configuration file designed for the Olimex ARM-USB-OCD.



External Tools

Create, manage, and run configurations

Run a program

Note: folder names may have changed due to a newer YAGARTO release

Name: OpenOCD

Location: C:\Program Files\openocd-2007re141\bin\openocd-ftd2xx.exe

Working Directory: C:\Program Files\openocd-2007re141\bin

Arguments: -f at91sam7s256-armusbocd.cfg

Note: Enclose an argument containing spaces using double-quotes (").

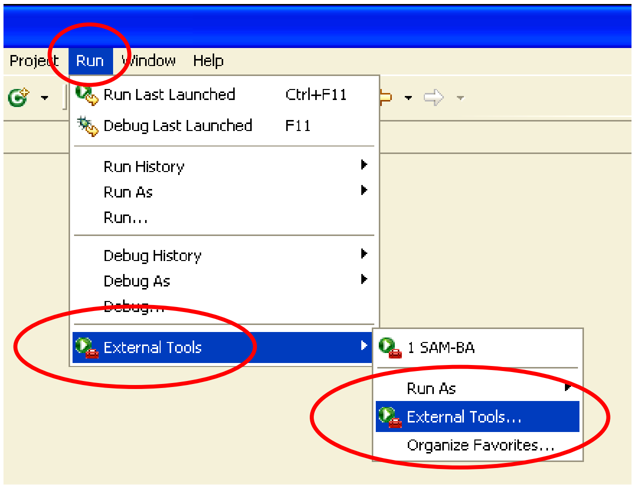
No changes are required to the other tabs in the form (Refresh, Environment, and Common).
Click on “**Apply**” and “**Close**” to register **OpenOCD** as an external tool.

Set Up OpenOCD as an Eclipse External Tool (JTAGkey)

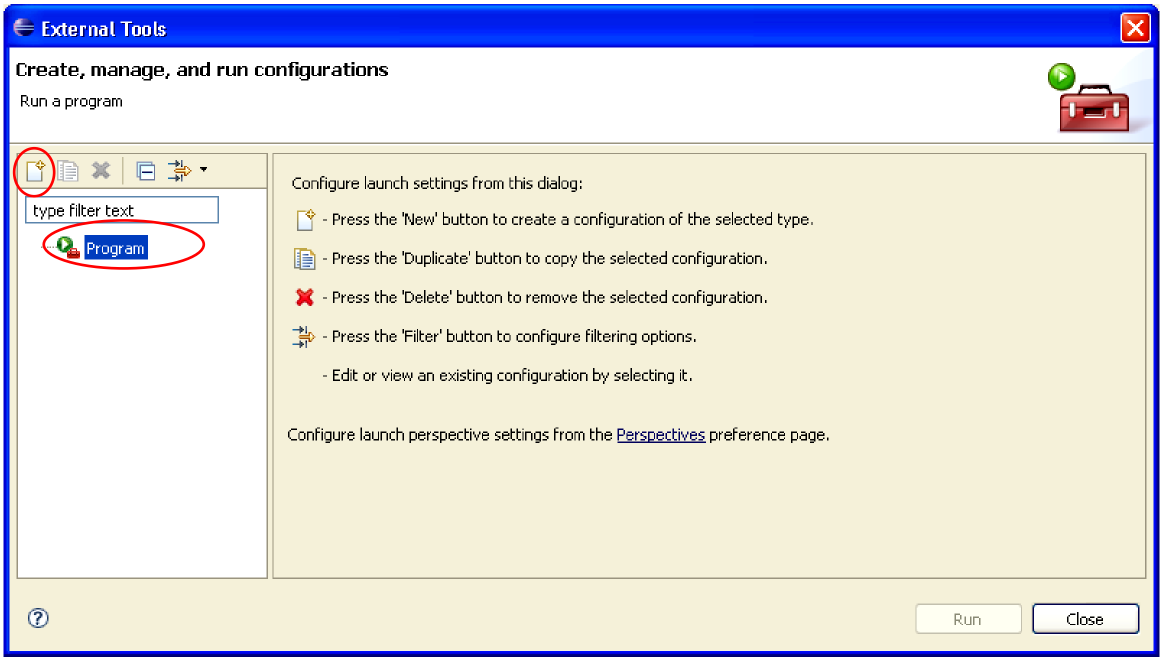
If you have purchased an Amontec JTAGKey, you need to set up OpenOCD as an external tool and tailor it specifically for operation with the JTAGKey.

When it's time to debug an application, we must be able to conveniently start the OpenOCD debugger. OpenOCD runs as a daemon; a program that runs in the background waiting for commands to be submitted to it. Eclipse has a very nice "external tool" feature that allows us to add OpenOCD to the RUN pull-down menu.

Click on "**Run – External Tools – External Tools…**"



The "**External Tools**" window will appear. Click on "**Program**" and then "**New**" button to establish a new External Tool.



Create, manage, and run configurations

Run a program

Configure launch settings from this dialog:

- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.

Configure launch perspective settings from the Perspectives preference page.

Fill out the “**External Tools**” form exactly as shown below.

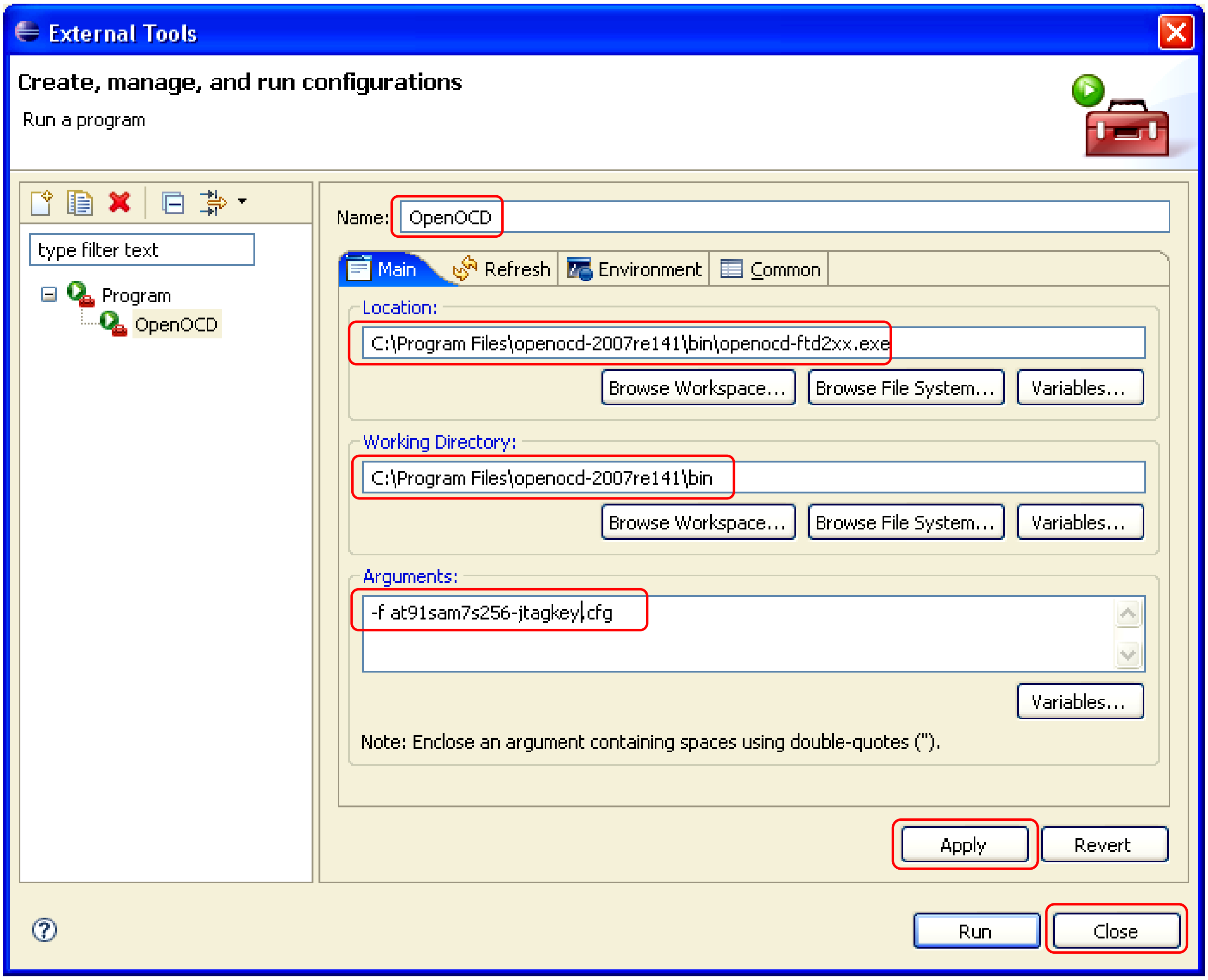
In the “Name” text box, call this external tool “**OpenOCD**”.

There are two versions of OpenOCD; **openocd-pp.exe** supports the parallel-port “wiggler” device (ARM-JTAG from Olimex) while **openocd-ftd2xx.exe** supports the USB-based devices from Amontec and Olimex. In this section we are installing the “USB” version of OpenOCD as an Eclipse external tool.

In the “Location:” pane, use the “**Browse File System…**” button to search for the OpenOCD executable; it will be in this folder: **c:\Program Files\openocd-2007re131\bin\openocd-ftd2xx.exe**”.

In the “Working Directory” pane, use the “**Browse File System…**” button to specify “**c:\Program Files\openocd-2007re131\bin\**” as the working directory.

In the “Arguments” pane, enter the argument “**-f at91sam7s256-jtagkey.cfg**” to specify the OpenOCD configuration file designed for the Amontec JTAGKey and its little brother, the JTAGKey-Tiny.


External Tools

Create, manage, and run configurations

Run a program

Name: OpenOCD

Location: C:\Program Files\openocd-2007re141\bin\openocd-ftd2xx.exe

Working Directory: C:\Program Files\openocd-2007re141\bin

Arguments: -f at91sam7s256-jtagkey.cfg

Note: Enclose an argument containing spaces using double-quotes (").

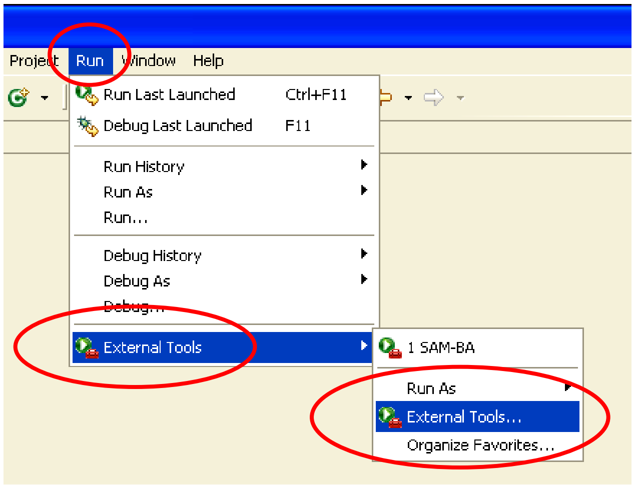
No changes are required to the other tabs in the form (Refresh, Environment, and Common).
Click on “**Apply**” and “**Close**” to register **OpenOCD** as an external tool.

Set Up J-Link GDB Server as an Eclipse External Tool (SAM-ICE)

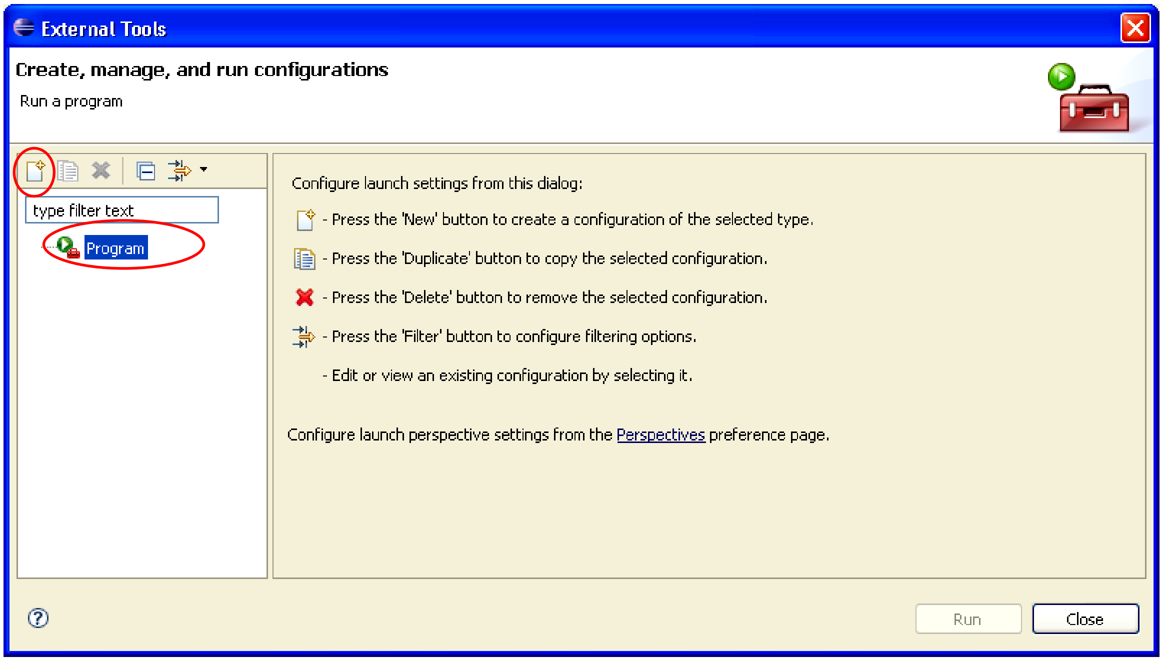
If you have purchased an Atmel SAM-ICE, you need to set up the J-Link GDB Server as an external tool and tailor it specifically for operation with the SAM-ICE.

When it's time to debug an application, we must be able to conveniently start the J-Link GDB Server. J-Link GDB Server runs as a daemon; a program that runs in the background waiting for commands to be submitted to it. Eclipse has a very nice "external tool" feature that allows us to add J-Link GDB Server to the RUN pull-down menu.

Click on "**Run – External Tools – External Tools…**"



The "**External Tools**" window will appear. Click on "**Program**" and then "**New**" button to establish a new External Tool.



Create, manage, and run configurations

Run a program

Configure launch settings from this dialog:

- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.

Configure launch perspective settings from the Perspectives preference page.

Fill out the "**External Tools**" form exactly as shown below.

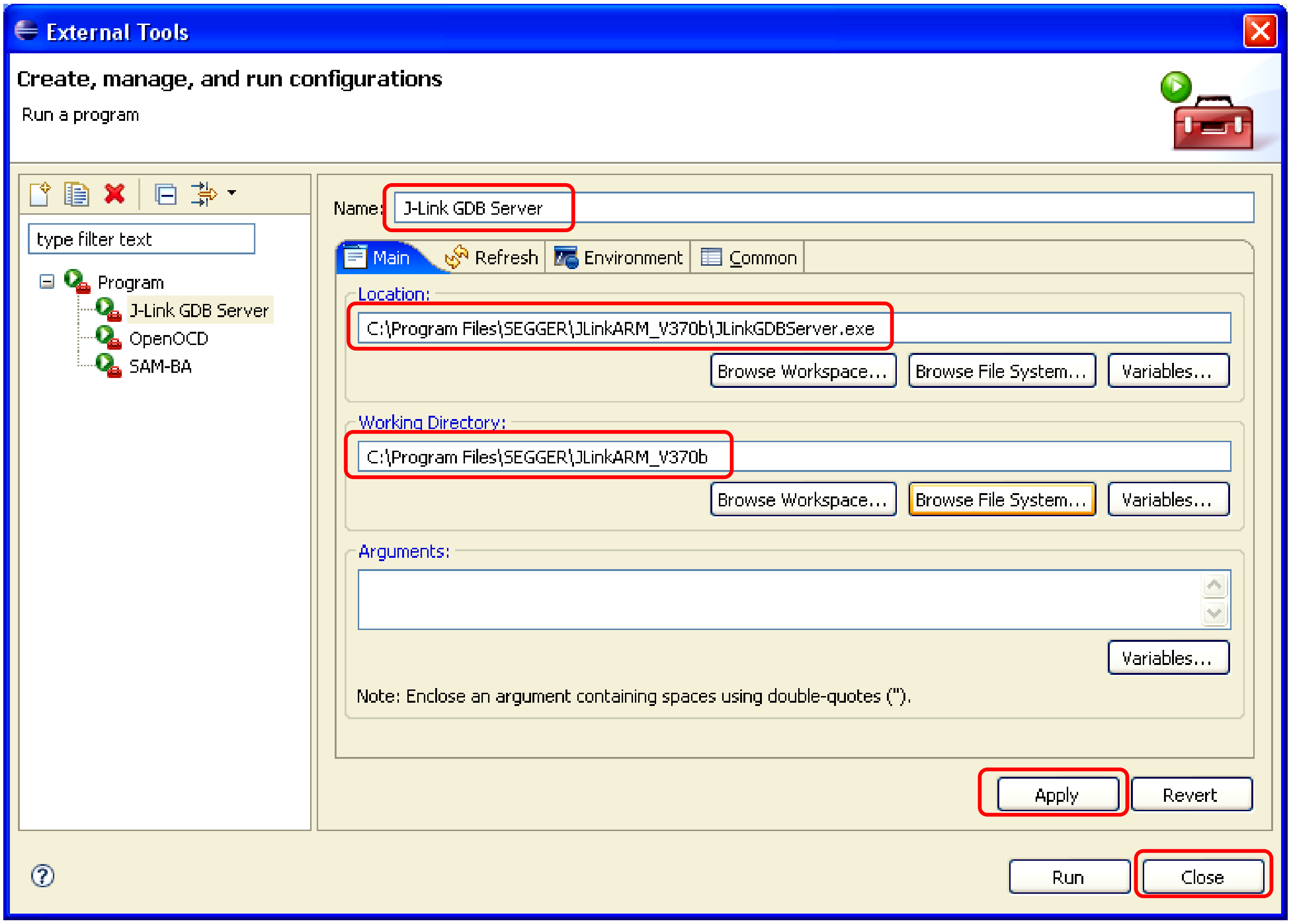
In the "Name" text box, call this external tool "**J-Link GDB Server**

In the "Location:" pane, use the "**Browse File System…**" button to search for the J-Link GDB Server executable; it will be in this folder: **c:\Program Files\SEGGER\JLinkARM_V370b\JLinkGDBServer.exe**".

In the "Working Directory" pane, use the "**Browse File System…**" button to specify "**c:\Program Files\SEGGER\JLinkARM_V370b\**" as the working directory.

The "Arguments" pane may be left empty

No changes are required to the other tabs in the form (Refresh, Environment, and Common).
Click on "**Apply**" and "**Close**" to register **J-Link** as an external tool.



External Tools

Create, manage, and run configurations

Run a program

Name: J-Link GDB Server

Location:
C:\Program Files\SEGGER\JLinkARM_V370b\JLinkGDBServer.exe

Working Directory:
C:\Program Files\SEGGER\JLinkARM_V370b

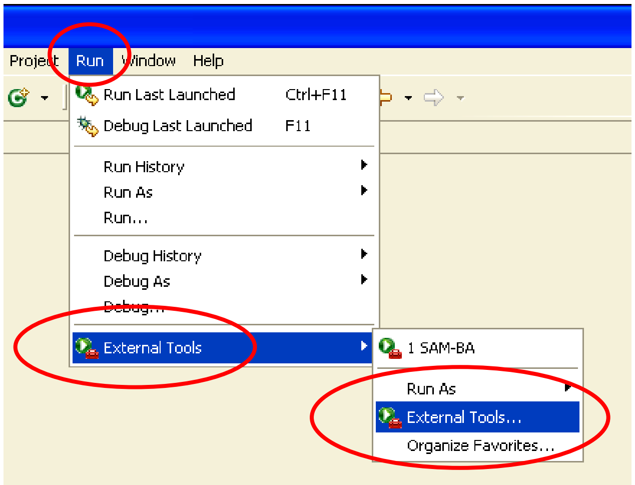
Arguments:

Note: Enclose an argument containing spaces using double-quotes (").

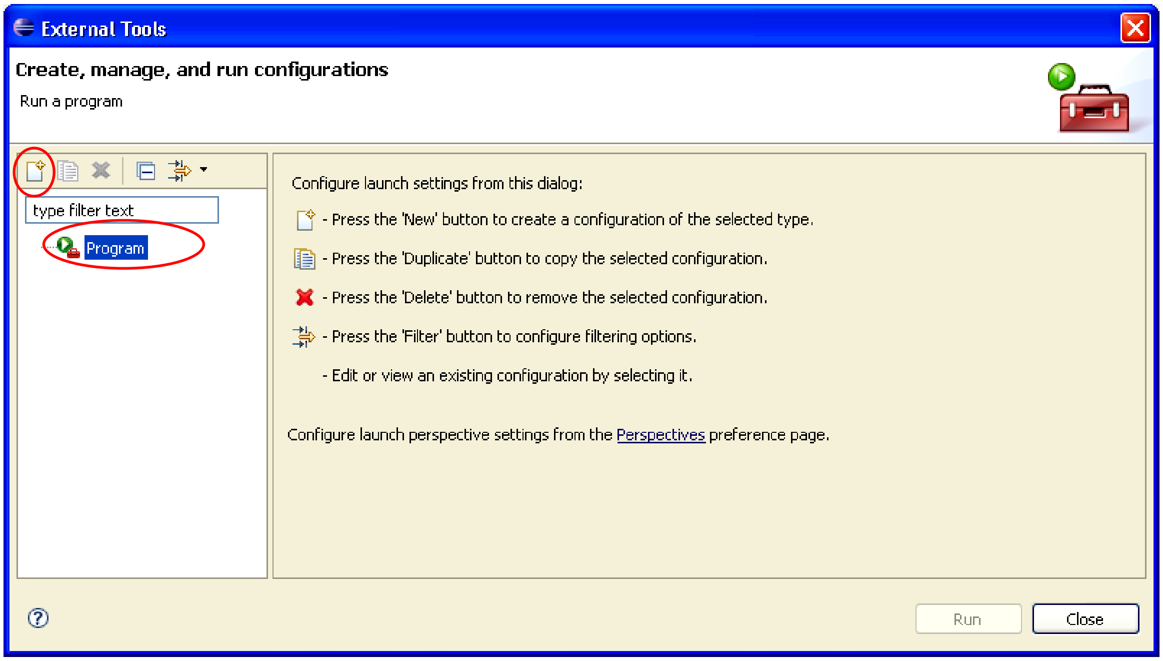
Set Up SAM-BA as an Eclipse External Tool

In any case, you should have the Atmel SAM-BA Flash Programming utility in your Eclipse toolbox. Use the following instructions to set up the SAM-BA utility as an Eclipse external tool.

Click on “**Run – External Tools – External Tools…**”



The “**External Tools**” window will appear. Click on “**Program**” and then “**New**” button to establish a new External Tool.



Create, manage, and run configurations

Run a program

Configure launch settings from this dialog:

- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.

Configure launch perspective settings from the Perspectives preference page.

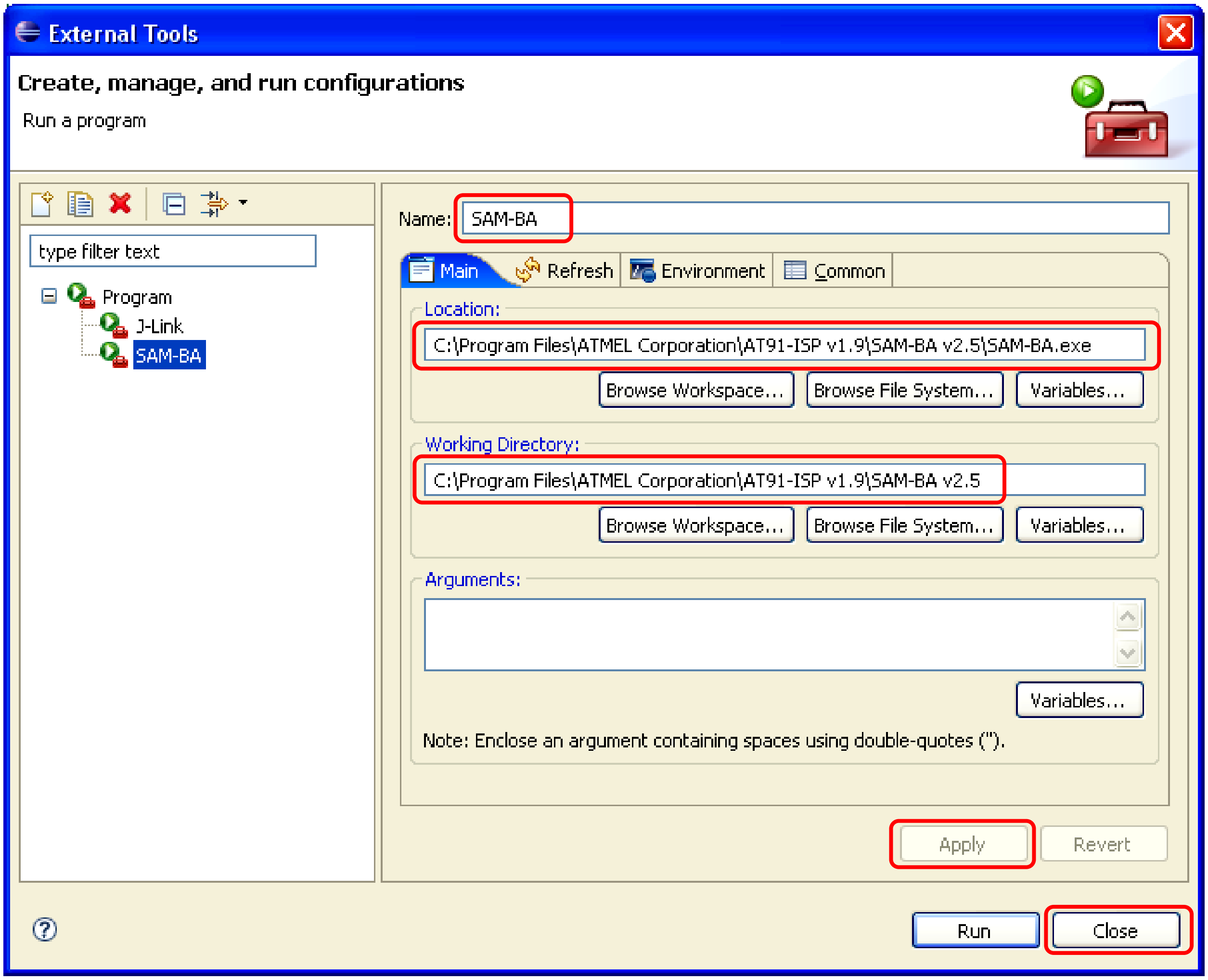
Fill out the “**External Tools**” form exactly as shown below.

In the “Name” text box, call this external tool “SAM-BA”.

In the “Location:” pane, use the “**Browse File System…**” button to search for the SAM-BA executable; it will be in this folder: “**c:\Program Files\ATMEL Corporation\AT91-ISP v1.9\SAM-BA.exe**”.

In the “Working Directory” pane, use the “**Browse File System…**” button to specify “**c:\Program Files\ATMEL Corporation\AT91-ISP v1.9**” as the working directory.

The “Arguments” pane may be left empty



External Tools

Create, manage, and run configurations

Run a program

type filter text

- Program
  - J-Link
  - SAM-BA

Name: SAM-BA

Location:
C:\Program Files\ATMEL Corporation\AT91-ISP v1.9\SAM-BA v2.5\SAM-BA.exe

Working Directory:
C:\Program Files\ATMEL Corporation\AT91-ISP v1.9\SAM-BA v2.5

Arguments:

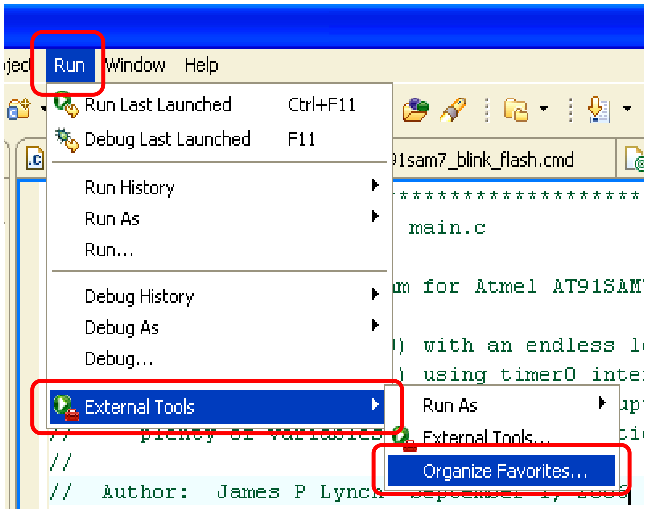
Note: Enclose an argument containing spaces using double-quotes (").

No changes are required to the other tabs in the form (Refresh, Environment, and Common).
Click on “**Apply**” and “**Close**” to register SAM-BA as an external tool.

Adding Your JTAG Tools into the "Favorites" List

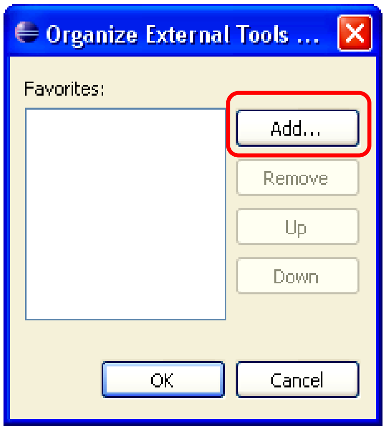
We have just installed the JTAG debugger daemon and the Atmel SAM-BA flash programming utility as Eclipse external tools. Just one more operation is needed to actually place them at the top of the "Run" pull-down menu; that is to add them to the "favorites" list.

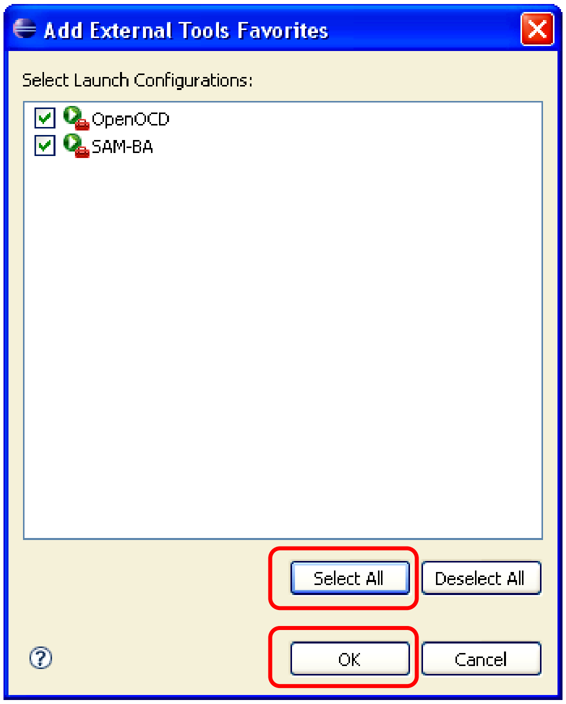
Click on "**Run – External Tools – Organize Favorites …**" as shown below.

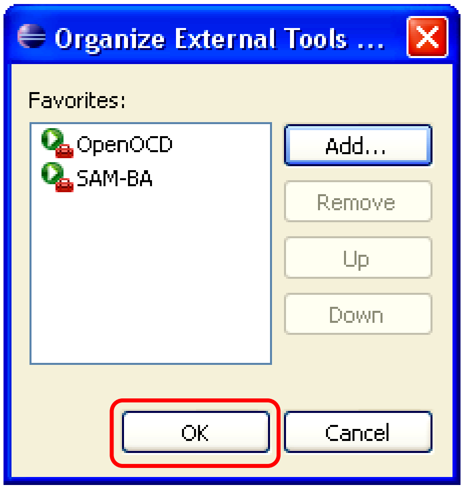


In the "Organize External Tools …" window below left, click on "**Add …**". This brings up the "Add External Tools Favorites" window in the middle below. Click on "**Select All**" followed by "**OK**".

The "Organize External Tools window reappears as shown below right. Click on "**OK**" to register OpenOCD or J-Link GDB Server and SAM-BA as "favorites". Note in the example below, we installed OpenOCD and SAM-BA as "favorites". If you have the SAM-ICE JTAG debugger, then you would install J-Link and SAM-BA as your favorites.

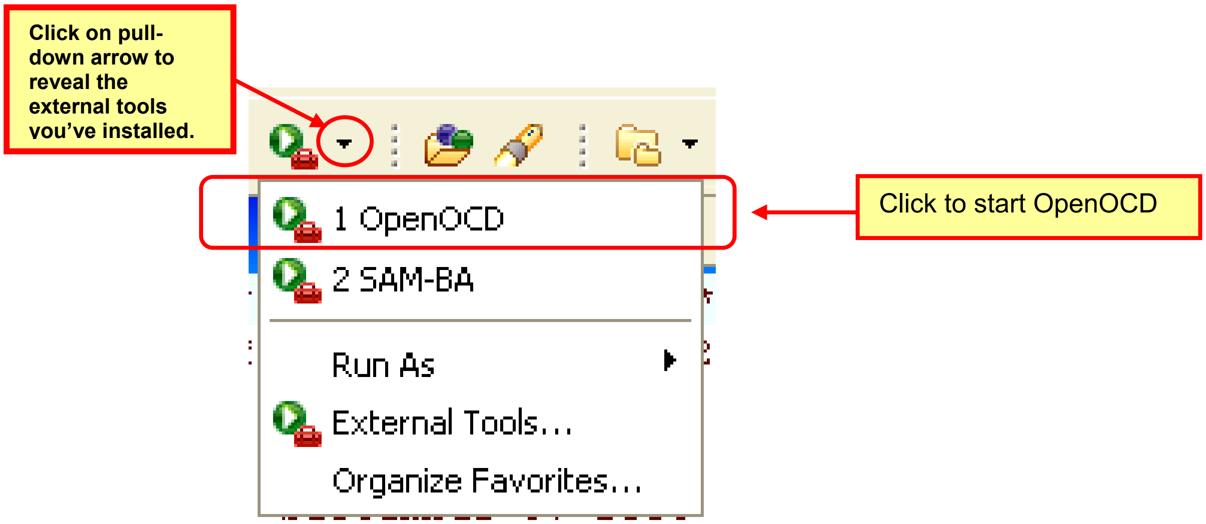






There are two convenient ways to start the JTAG software daemon; the RUN menu or the External Tools toolbar button.

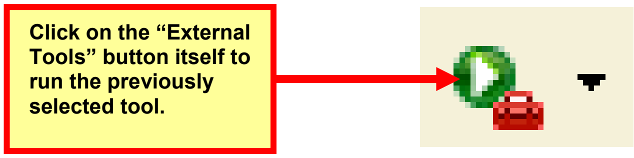
The toolbar button is the most convenient. Click on the little pull-down arrow on the External Tools button. The JTAG executable appears at the top of the list, just click on it to start the OpenOCD JTAG daemon.



Click on pull-down arrow to reveal the external tools you've installed.

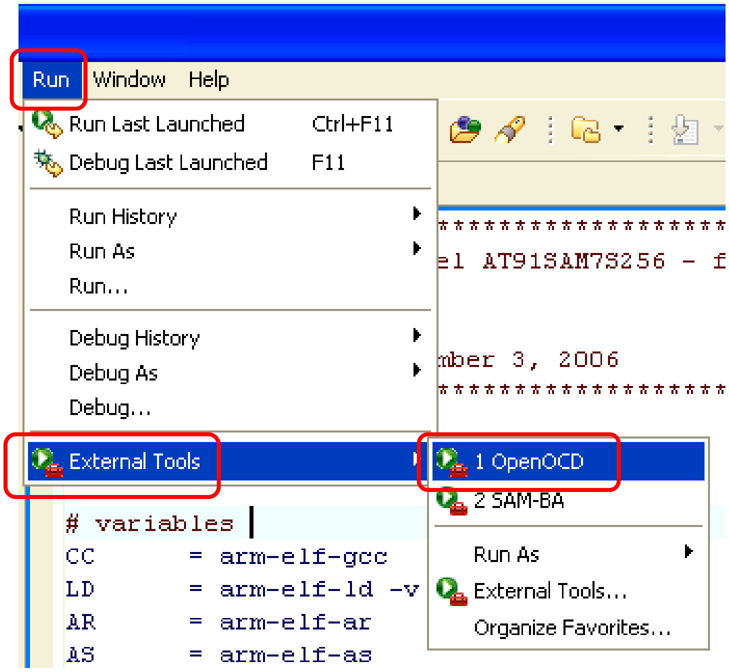
Click to start OpenOCD

Eclipse always remembers the last external tool you selected. Therefore, the next time just clicking on the External Tool toolbar button itself will start the previously selected tool.



Click on the "External Tools" button itself to run the previously selected tool.

Finally, you can also start the JTAG software daemon from the "Run" pull-down menu itself, as shown below. Click on "**Run**" followed by "**External Tools**" and then the tool itself (**OpenOCD** in this example). There will typically be multiple tools installed; for example the Atmel SAM-BA boot assistant utility can be conveniently started the same way.



```
*******************
el AT91SAM7S256 - f

mber 3, 2006
*******************

# variables
CC       = arm-elf-gcc
LD       = arm-elf-ld -v
AR       = arm-elf-ar
AS       = arm-elf-as
```

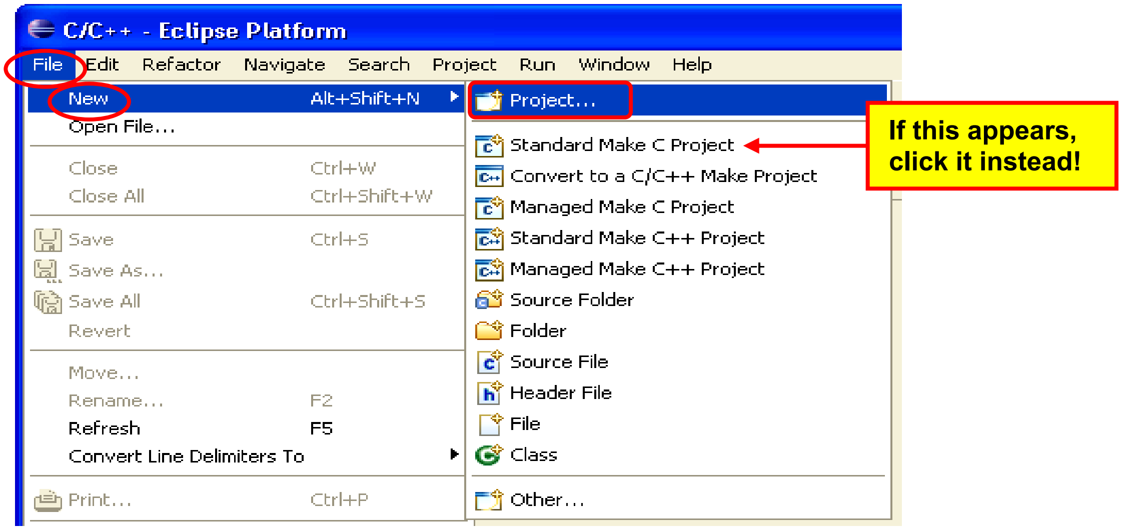
Create an Eclipse Project

Now all our hard work preparing an open source Eclipse tool set will pay off. We can now actually create a bona fide Atmel AT91SAM7 application using the Eclipse IDE and the open source compilers and debuggers.

Click on the desktop Eclipse icon to start Eclipse.

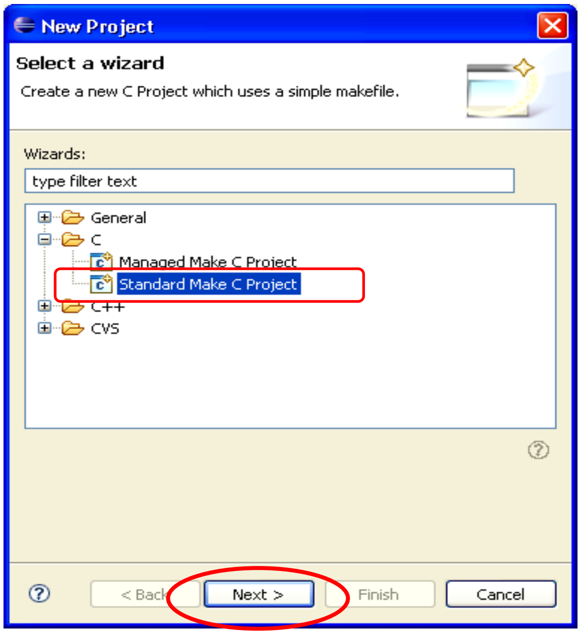
Let's jump right in and create an Eclipse C/C++ project. This project will run out of FLASH memory. Specifically the project will blink LED1 in a main program background loop. It will blink LED2 on an IRQ interrupt from onboard Timer1. Finally, if you push switch SW1 it will assert a FIQ interrupt that flashes LED3 and increments a counter. There are also plenty of variables defined for debug practice.

In the **File** pull-down menu, click on "**File – New – Project…**" to get started, as shown below.



If this appears, click it instead!

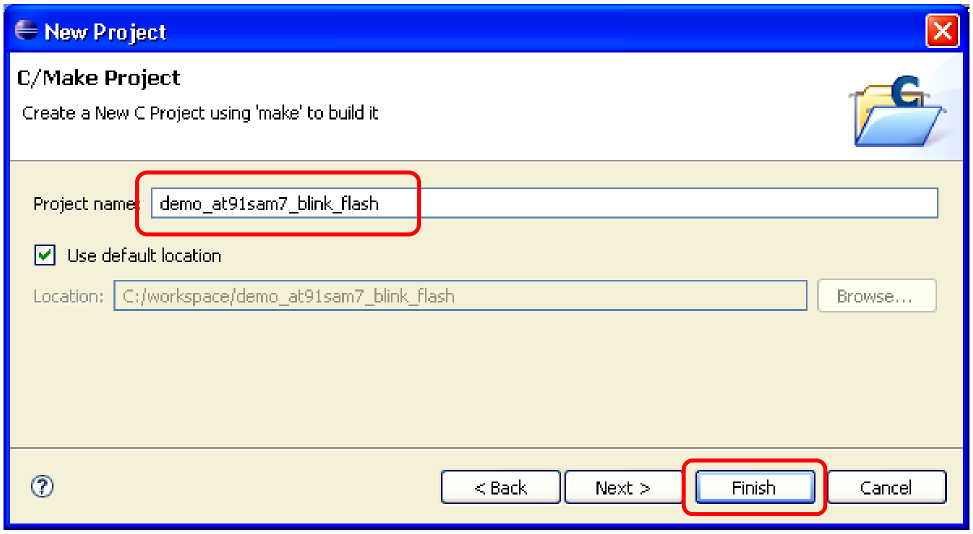
In the "New Project" wizard shown below, expand the C type by clicking on the "**+**" sign and then select "**Standard Make C Project**". Click "**Next**" to continue.



Select a wizard

Create a new C Project which uses a simple makefile.

Enter the sample project name “**demo_at91sam7_blink_flash**” into the text window below. Click “**Finish**” to continue.



New Project

C/Make Project

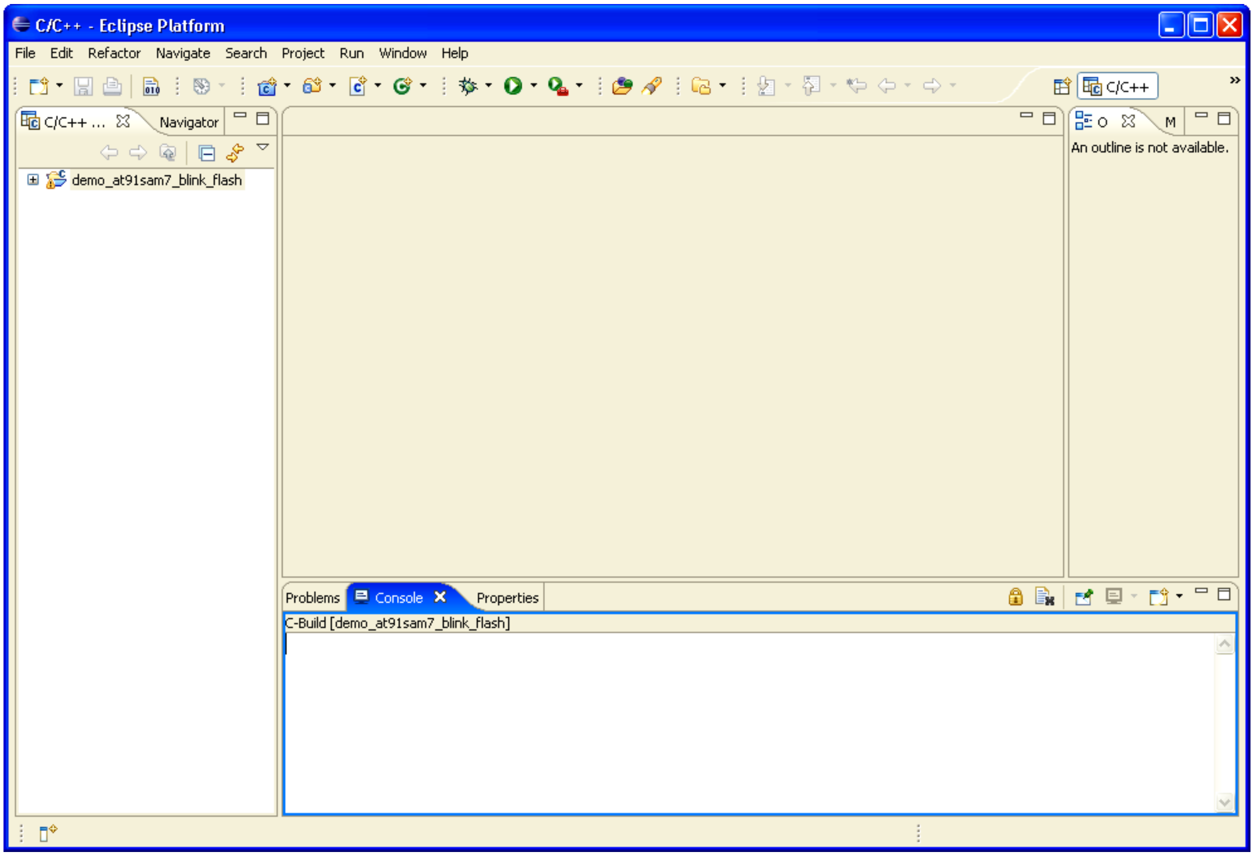
Create a New C Project using 'make' to build it

Project name: demo_at91sam7_blink_flash

☑ Use default location

Location: C:/workspace/demo_at91sam7_blink_flash

Now the C/C++ perspective shows a valid project, as shown below in the C/C++ Projects view on the left, but there are no source files in that project. Normally you would select “**File – New – Source File**” and enter a file name and start typing. This time, however, we will be importing source files already prepared by the author to demonstrate Eclipse’s features.



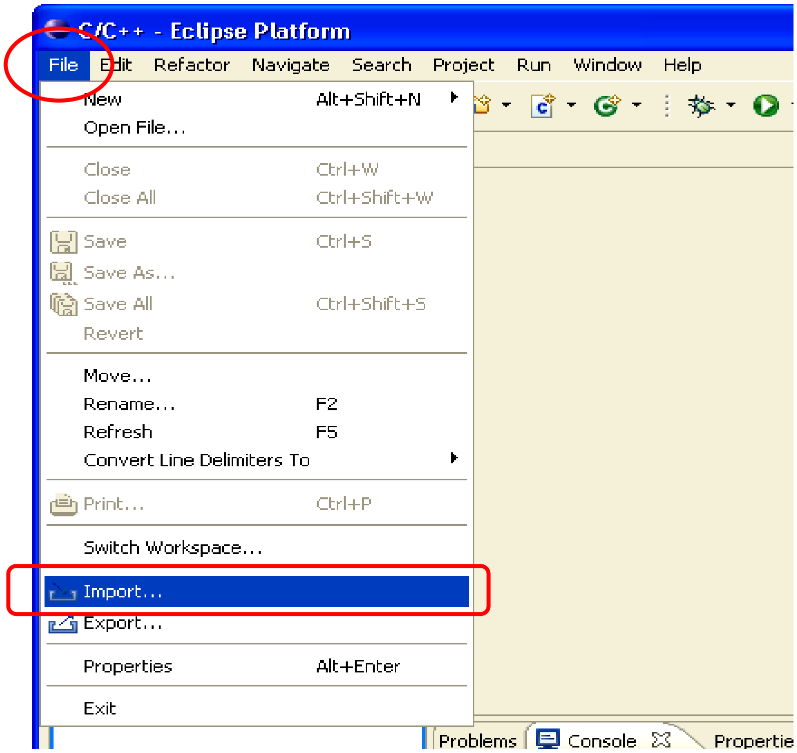
C/C++ - Eclipse Platform

demo_at91sam7_blink_flash

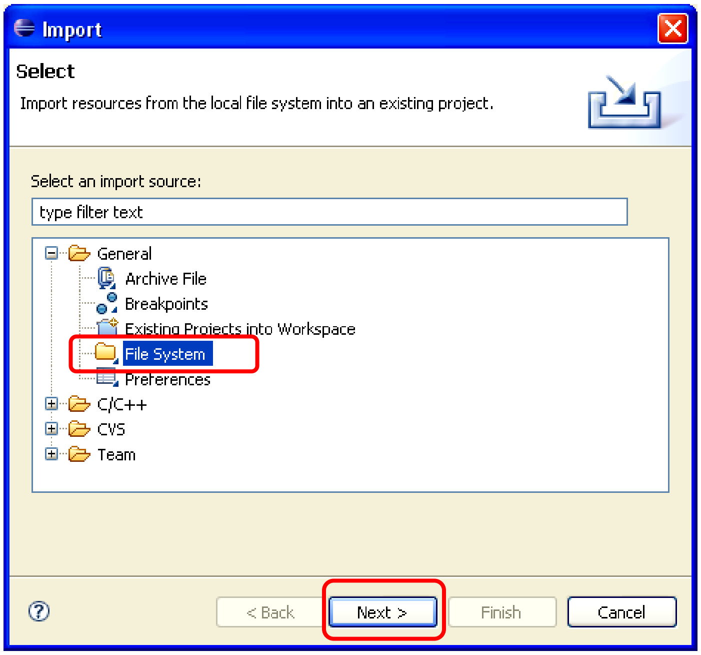
An outline is not available.

C-Build [demo_at91sam7_blink_flash]

In the Eclipse screen below, click on “**File – Import…**”; this will bring up the file import dialog.



In the “Import” screen below, click on “**File System**” and then click “**Next**” to continue.


Import

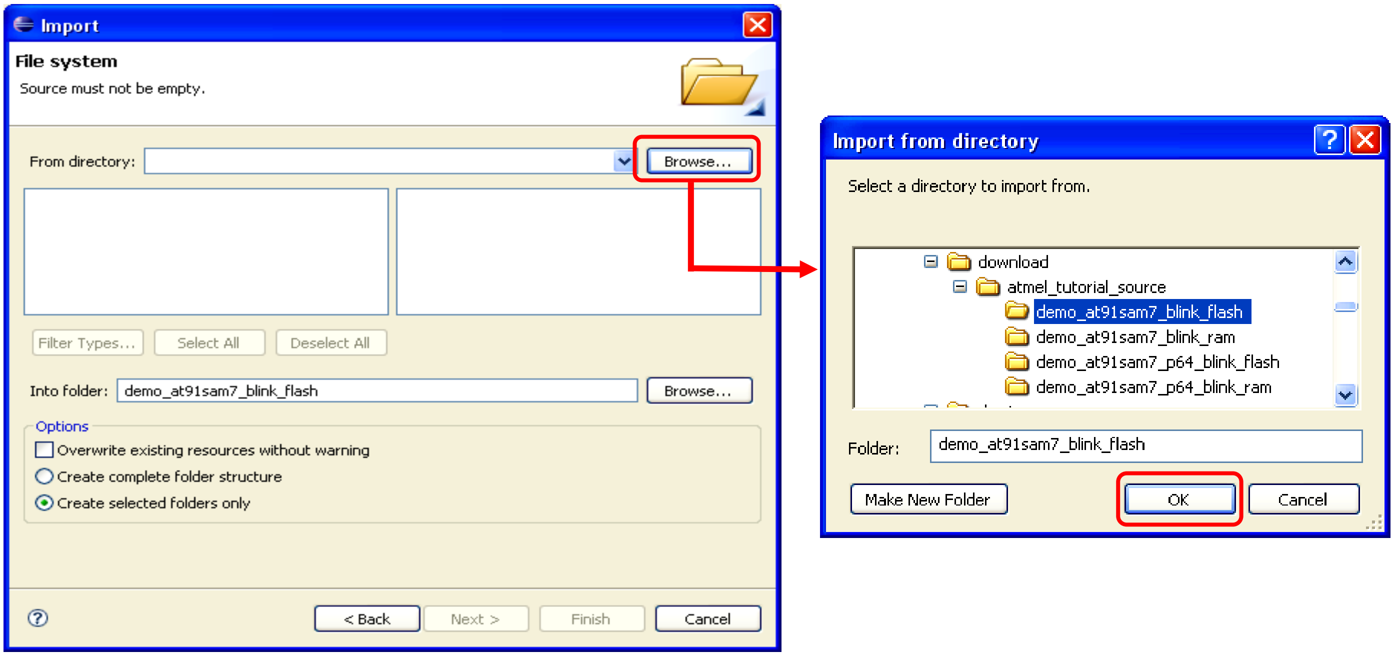
Select

Import resources from the local file system into an existing project.

Select an import source:

In the “Import – File system” screen below, use the “**Browse**” button associated with the “From directory” text box to search for the sample project to be imported. In this case, it resides in the folder you created earlier: **c:\download\atmel_tutorial_source\demo_at91sam7_blink_flash.**

By the way, you will use this procedure many times in the future to create a new Eclipse project from the components of a previous project.



File system
Source must not be empty.

From directory:

Into folder: demo_at91sam7_blink_flash

Options
☐ Overwrite existing resources without warning
○ Create complete folder structure
● Create selected folders only

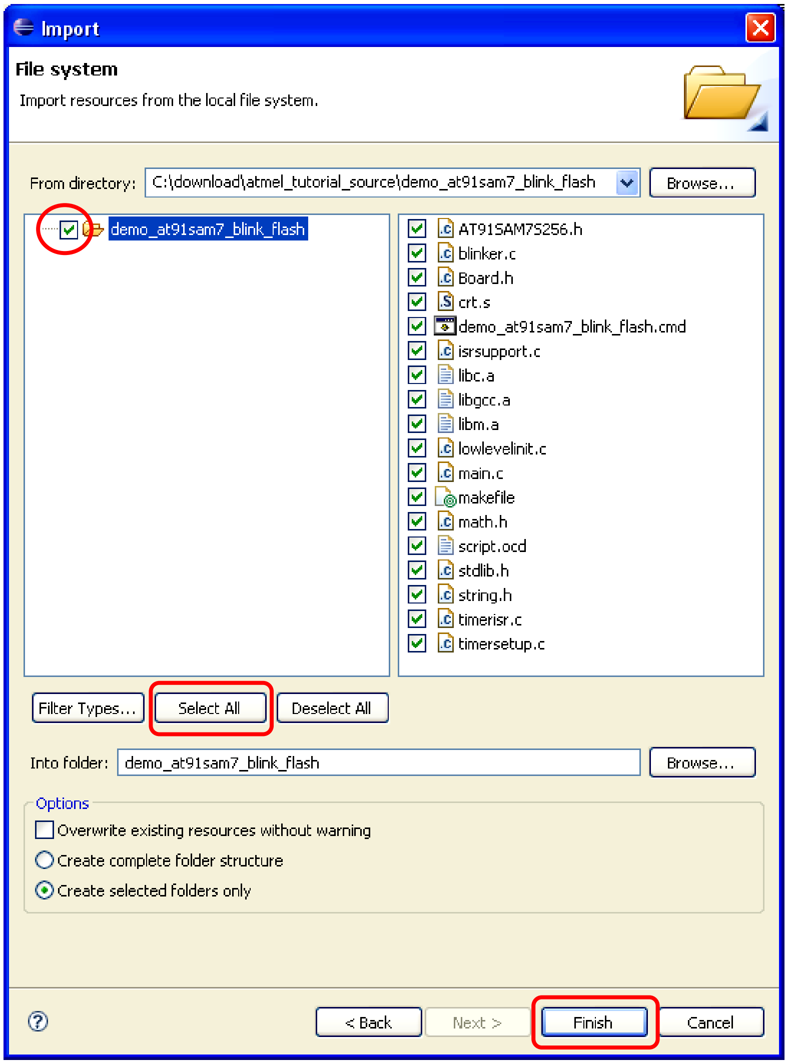
Select a directory to import from.

⊟ download
⊟ atmel_tutorial_source
demo_at91sam7_blink_flash
demo_at91sam7_blink_ram
demo_at91sam7_p64_blink_flash
demo_at91sam7_p64_blink_ram

Folder: demo_at91sam7_blink_flash

Check the box for the folder “demo_at91sam7_blink_flash” and then click the “**Select All**” button below because we want to import every one of these files.

The “Into folder:” text box should already be filled in properly; if not, click the “**Browse**” button to specify the project folder “**demo_at91sam7_blink_flash**”. Click “**Finish**” to start the File Import operation.



File system
Import resources from the local file system.

From directory: C:\download\atmel_tutorial_source\demo_at91sam7_blink_flash

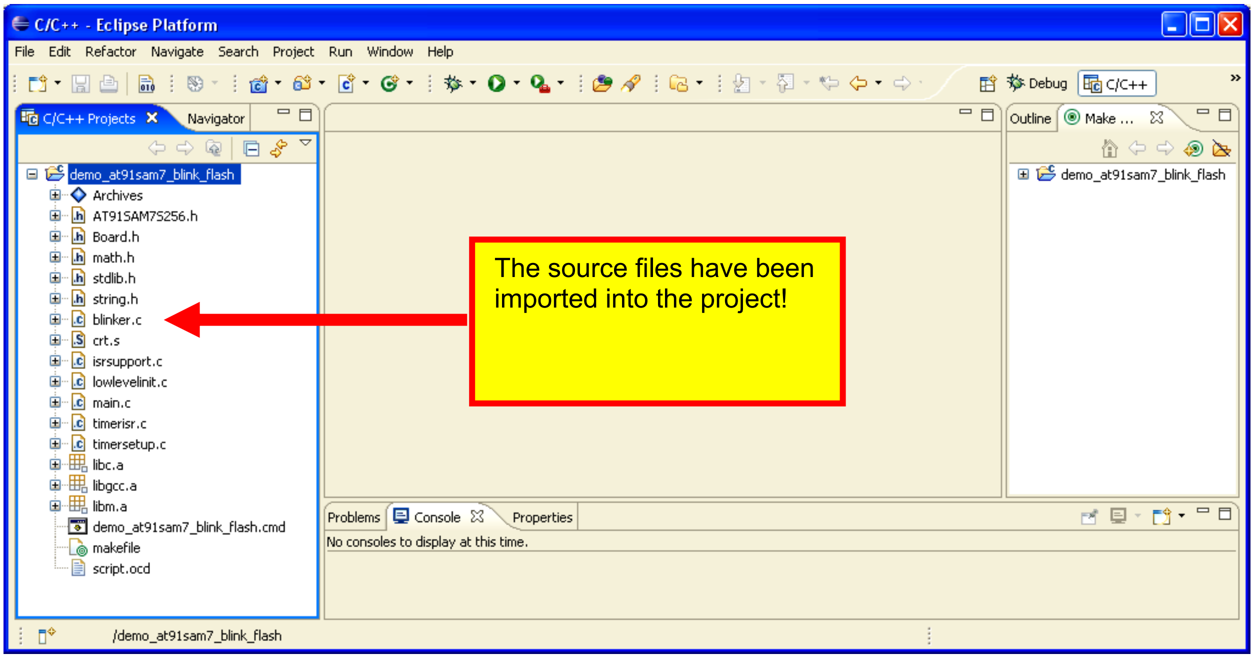
☑ demo_at91sam7_blink_flash

☑ AT91SAM7S256.h
☑ blinker.c
☑ Board.h
☑ crt.s
☑ demo_at91sam7_blink_flash.cmd
☑ isrsupport.c
☑ libc.a
☑ libgcc.a
☑ libm.a
☑ lowlevelinit.c
☑ main.c
☑ makefile
☑ math.h
☑ script.ocd
☑ stdlib.h
☑ string.h
☑ timerisr.c
☑ timersetup.c

Into folder: demo_at91sam7_blink_flash

Options
☐ Overwrite existing resources without warning
○ Create complete folder structure
● Create selected folders only

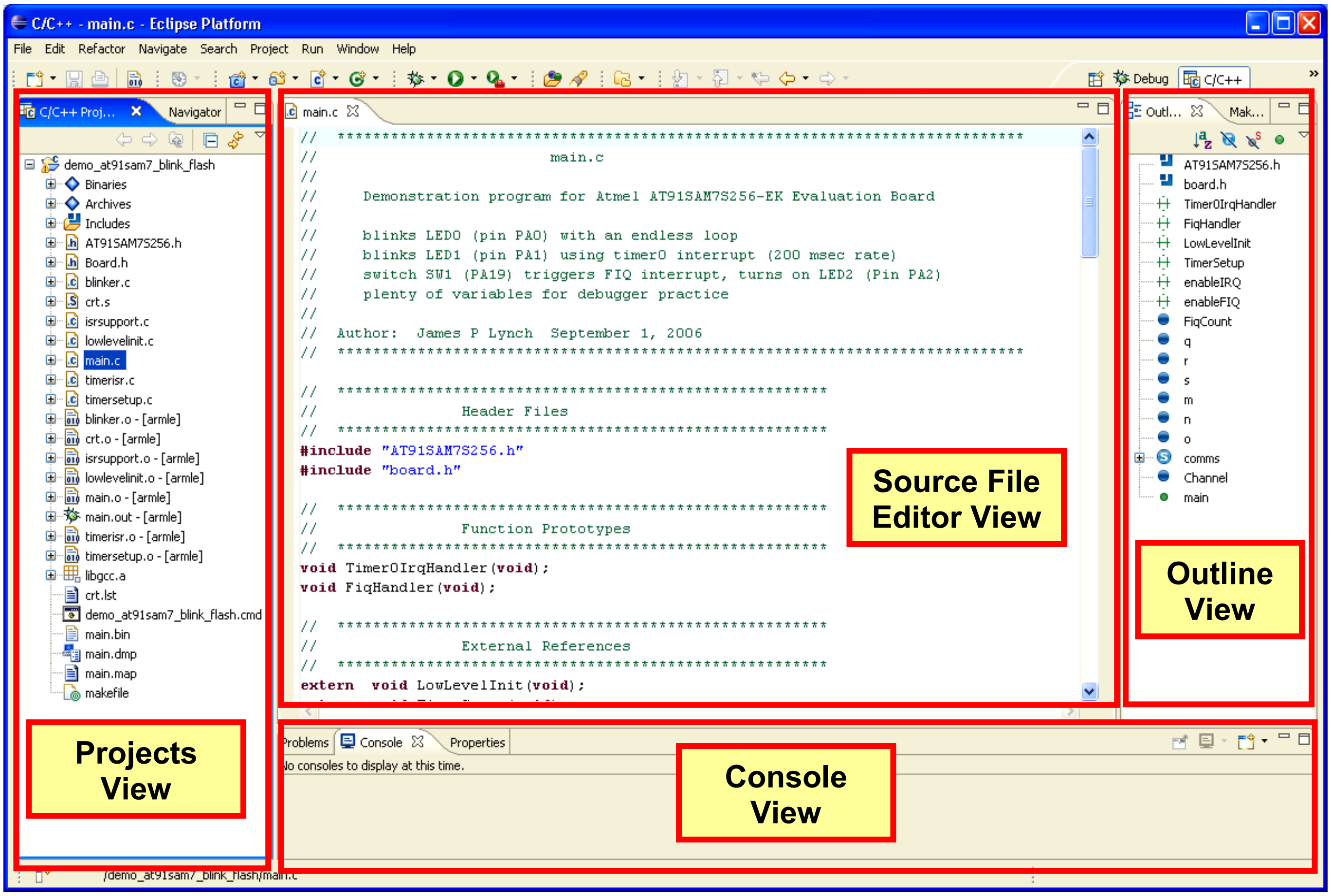
Now if you expand the **demo_at91sam7_blink_flash** project in the C/C++ Projects view below, you will see that all the source files have been imported into our project. By clicking on the "**+**" sign on the project name in the C/C++ Projects panel on the left, the imported files are revealed.



No consoles to display at this time.

The source files have been imported into the project!

In the Eclipse window below, the **main.c** file has been selected by clicking on it and it thus displays in the source file editor view in the center.



```
//  *****************************************************************
//                          main.c
//
//     Demonstration program for Atmel AT91SAM7S256-EK Evaluation Board
//
//     blinks LED0 (pin PA0) with an endless loop
//     blinks LED1 (pin PA1) using timer0 interrupt (200 msec rate)
//     switch SW1 (PA19) triggers FIQ interrupt, turns on LED2 (Pin PA2)
//     plenty of variables for debugger practice
//
//  Author:  James P Lynch  September 1, 2006
//  *****************************************************************

//  **********************************************
//                 Header Files
//  **********************************************
#include "AT91SAM7S256.h"
#include "board.h"

//  **********************************************
//                 Function Prototypes
//  **********************************************
void Timer0IrqHandler(void);
void FiqHandler(void);

//  **********************************************
//                 External References
//  **********************************************
extern  void LowLevelInit(void);
```

No consoles to display at this time.

Source File Editor View

Outline View

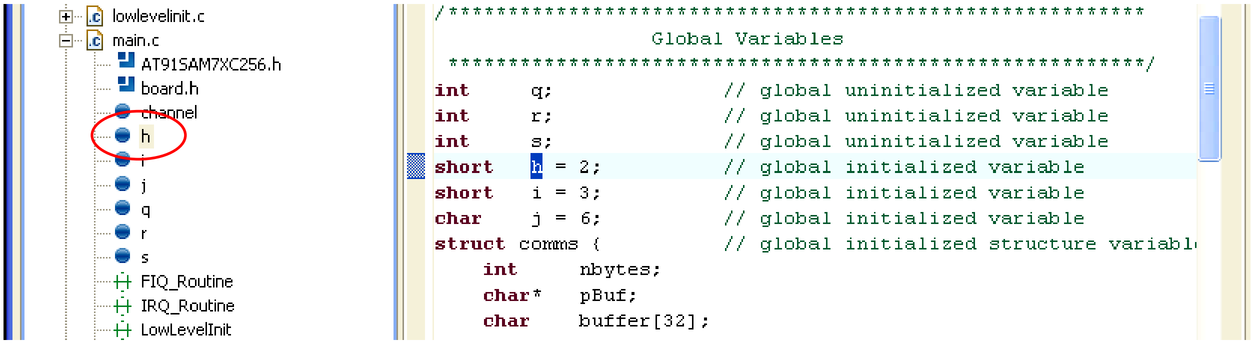
Projects View

Console View

In the “**C/C++ Projects”** view on the left, you can click on any source file and the Source Window will jump to that file.

Source modules can be expanded (by clicking on the “+” expander icon) to reveal the variables and functions contained therein. This allows a very quick way to find the definition of a variable in the file.

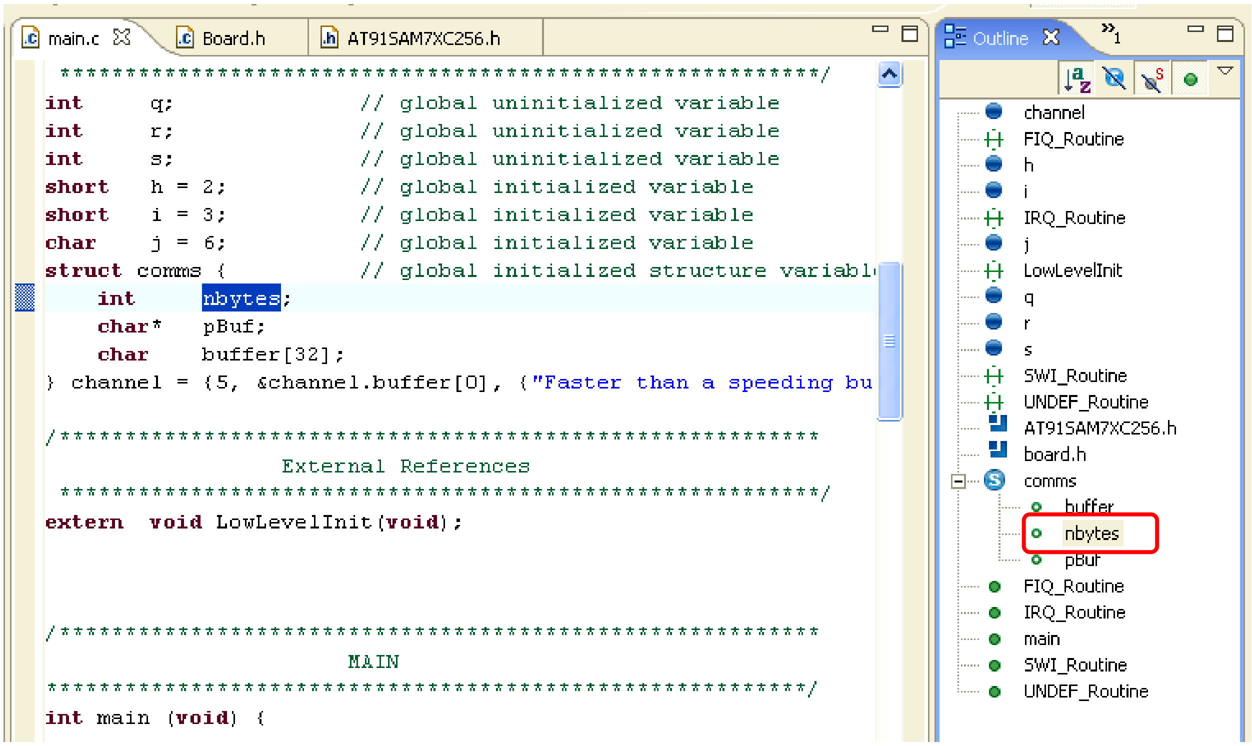
In the sample directly below, we expanded the main.c source file to reveal the variables and functions. By clicking on the variable “**h**” in the C/C++ Projects view on the left, the source window jumps to the definition of that variable. This feature is more dramatic when you have a very large source file and it’s tedious to scroll through all of it looking for a particular variable or function.



```
/*************************************************************
                         Global Variables
 *************************************************************/
int     q;              // global uninitialized variable
int     r;              // global uninitialized variable
int     s;              // global uninitialized variable
short   h = 2;          // global initialized variable
short   i = 3;          // global initialized variable
char    j = 6;          // global initialized variable
struct comms {          // global initialized structure variabl
    int     nbytes;
    char*   pBuf;
    char    buffer[32];
```

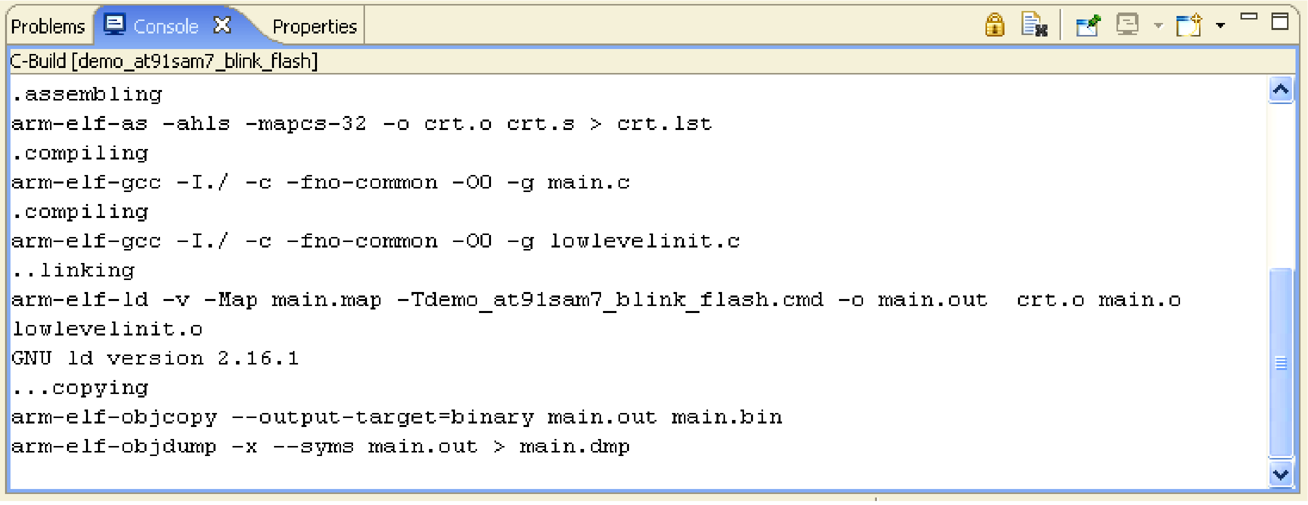
In the “**Outline**” view on the right, any C/C++ file being displayed in the source window in the center will have a tabular list of all important C/C++ elements (such as enumerations, structures, typedefs, variables, etc) to allow quick location of those elements in the source file.

In the example below, clicking on “nbytes” in the comms structural variable will cause the source file to jump to the definition of the “nbytes” element.



```
 ************************************************************/
int     q;              // global uninitialized variable
int     r;              // global uninitialized variable
int     s;              // global uninitialized variable
short   h = 2;          // global initialized variable
short   i = 3;          // global initialized variable
char    j = 6;          // global initialized variable
struct comms {          // global initialized structure variabl
    int     nbytes;
    char*   pBuf;
    char    buffer[32];
} channel = {5, &channel.buffer[0], {"Faster than a speeding bu

/************************************************************
                    External References
 ************************************************************/
extern  void LowLevelInit(void);



/************************************************************
                         MAIN
*************************************************************/
int main (void) {
```

At the bottom of the Eclipse screen is the "**Console**" view. This shows, for example, the execution of the Make utility. In the example shown below, you can see the GNU assembler, compiler and linker steps being executed. If there are problems, you can select the "**Problems**" tab to see more information pertaining to any problems that occur.



```
.assembling
arm-elf-as -ahls -mapcs-32 -o crt.o crt.s > crt.lst
.compiling
arm-elf-gcc -I./ -c -fno-common -O0 -g main.c
.compiling
arm-elf-gcc -I./ -c -fno-common -O0 -g lowlevelinit.c
..linking
arm-elf-ld -v -Map main.map -Tdemo_at91sam7_blink_flash.cmd -o main.out  crt.o main.o
lowlevelinit.o
GNU ld version 2.16.1
...copying
arm-elf-objcopy --output-target=binary main.out main.bin
arm-elf-objdump -x --syms main.out > main.dmp
```

Eclipse CDT has a fairly comprehensive User's Guide that can be downloaded from here:

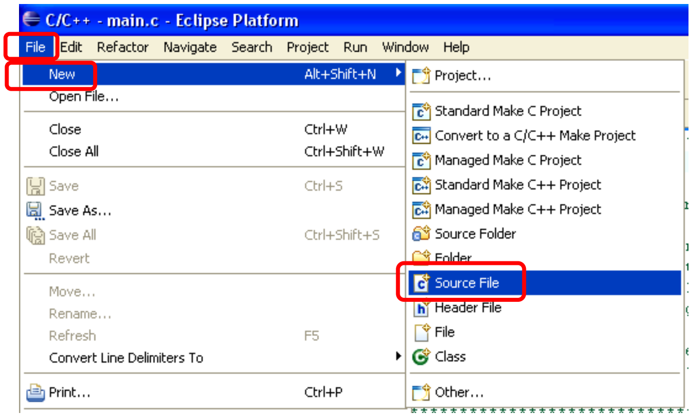
http://dev.eclipse.org/viewcvs/index.cgi/%7Echeckout%7E/cdt-home/user/C_C++_Development_Toolkit_User_Guide.pdf?cvsroot=Tools_Project

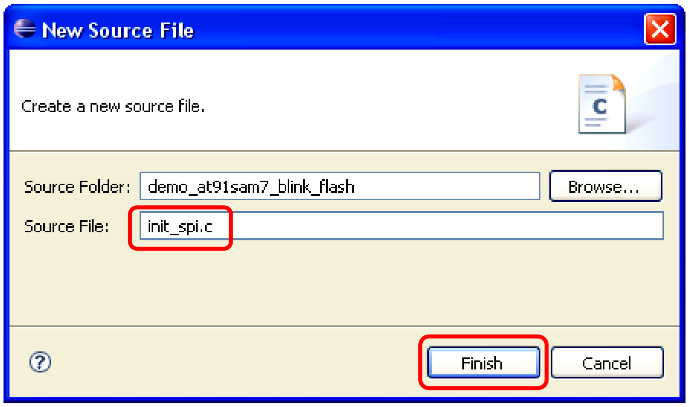
Using the Eclipse Editor

The Eclipse editor works like most editors you have used. Since it's a "software" editor, the fonts are fixed-pitch and that makes indented code line up very nicely.

Creating a New Source File

To create a new file from scratch, just click "**File – New – Source File**" as shown below left. You will be asked for a file name, enter the name and extension as shown below right.

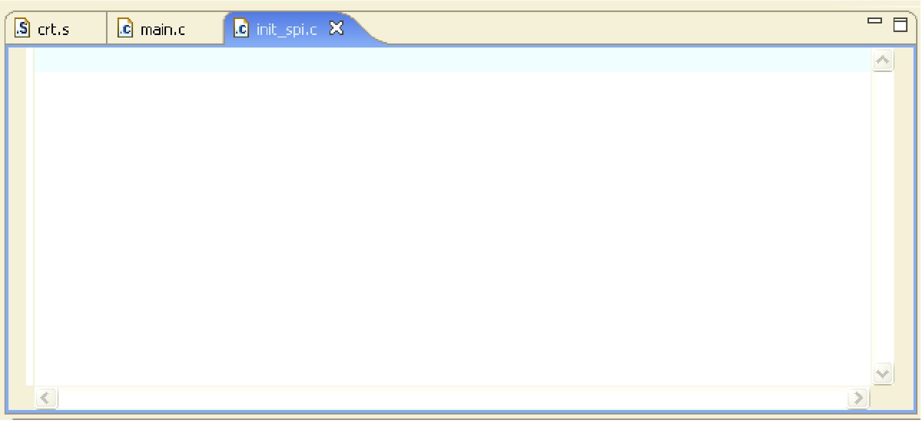




Source Folder: demo_at91sam7_blink_flash

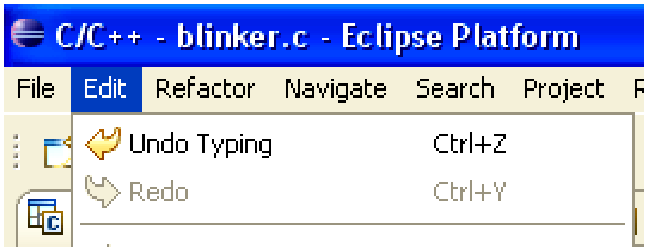
Source File: init_spi.c

Click "**Finish**" above right to create a new editing window, as shown below. The new file name appears in your Eclipse project view on the far left. Now you can type in your new file!



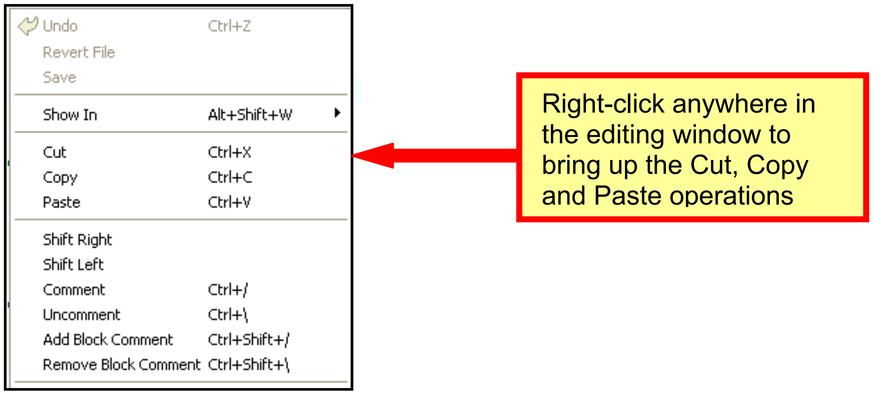
Undo / Redo

Eclipse has a full "**Undo**" facility; it's found in the Edit pull-down menu as shown below.



Cut, Copy and Paste Operations

Cut, Copy and Paste operations are in the “Edit” pull-down menu, but right-clicking anywhere in the editing window will bring up the “right-click” menu wherein you can select the Cut, Copy or Paste operation, as shown below. There is currently no “column copy and paste” operation available, but the thousands who complained have been promised this feature in the summer 2007 release of Eclipse.

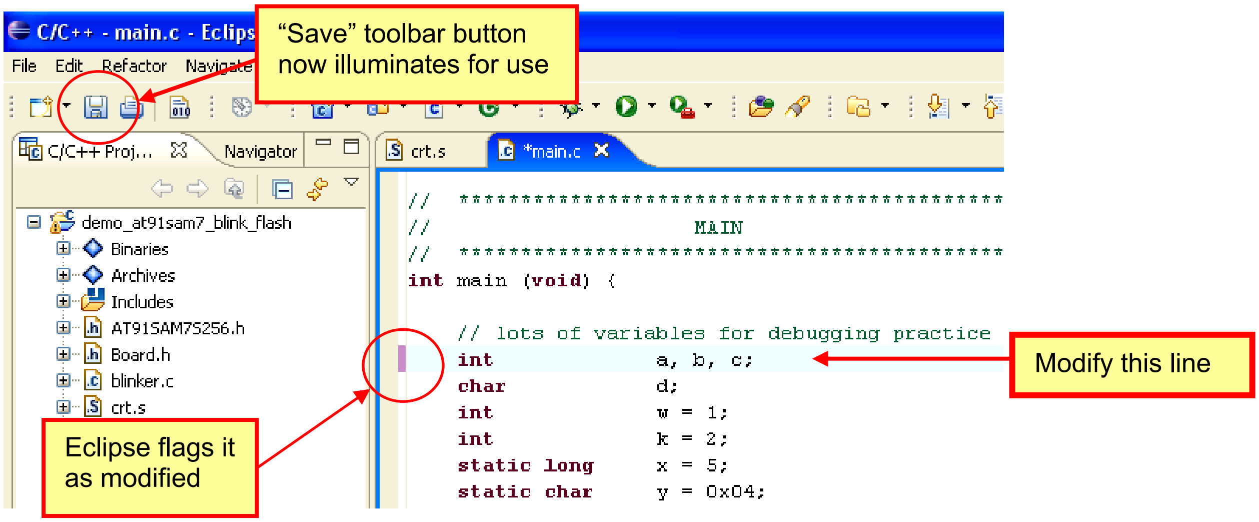


Right-click anywhere in the editing window to bring up the Cut, Copy and Paste operations

Saving Your Code

If you modify a source line as shown below, Eclipse tags the line as modified by a notation in the left margin and illuminates the “**Save**” button in the toolbar. Clicking the toolbar “**Save**” button updates the file copy with your changes and removes the “modified” notation.

In the “**Windows – Preferences – General – Workspace**” pull-down menu, you can set up Eclipse to automatically save before a build and automatically save every few minutes.



```
//  ***********************************************
//                      MAIN
//  ***********************************************
int main (void) {

    // lots of variables for debugging practice
    int                 a, b, c;
    char                d;
    int                 w = 1;
    int                 k = 2;
    static long         x = 5;
    static char         y = 0x04;
```

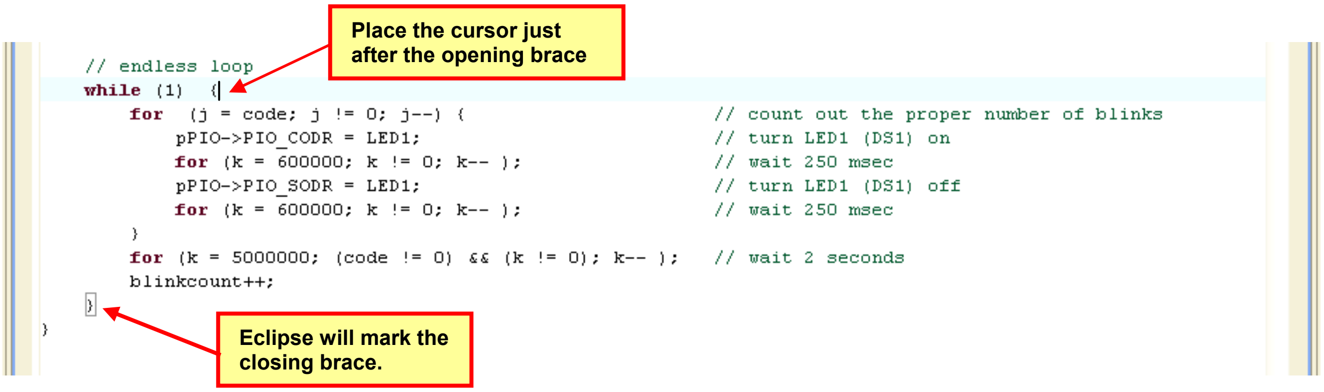
“Save” toolbar button now illuminates for use

Modify this line

Eclipse flags it as modified

Brace Checking

Locating the closing brace is quite easy; just position the cursor just after the opening brace and the closing brace will be immediately identified by Eclipse with a little box as shown below. This works in reverse at the closing brace. The same trick also works for parentheses.



```
    // endless loop
    while (1)  {
        for  (j = code; j != 0; j--) {                     // count out the proper number of blinks
            pPIO->PIO_CODR = LED1;                          // turn LED1 (DS1) on
            for (k = 600000; k != 0; k-- );                 // wait 250 msec
            pPIO->PIO_SODR = LED1;                          // turn LED1 (DS1) off
            for (k = 600000; k != 0; k-- );                 // wait 250 msec
        }
        for (k = 5000000; (code != 0) && (k != 0); k-- );   // wait 2 seconds
        blinkcount++;
    }
}
```

Place the cursor just after the opening brace

Eclipse will mark the closing brace.

Searching

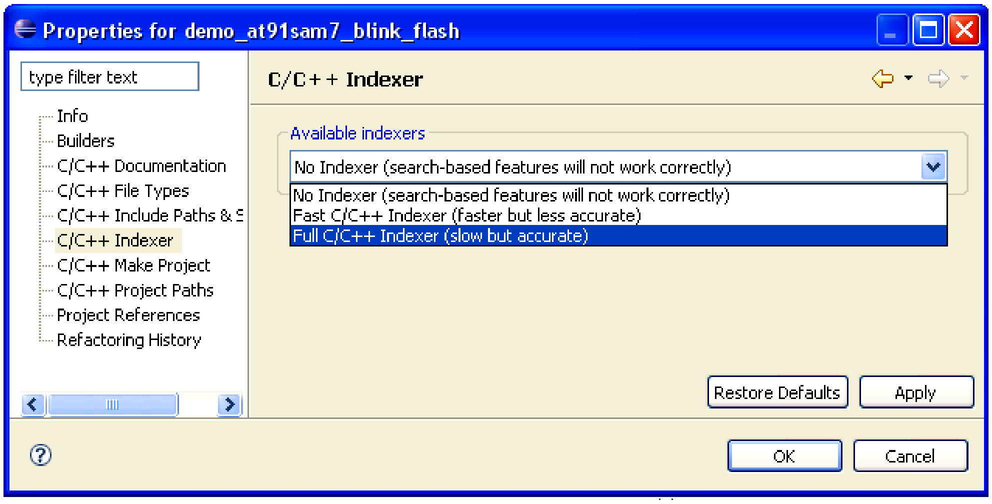
Eclipse has extremely sophisticated search/replace capabilities. To simplify things a bit, the novice user is probably interested in just two search features:

- Show me the definition of the variable I've selected

- Show me every place in the project where I've used it

First, we have to make sure the Eclipse Indexer is turned on. In the Projects pull-down menu, click on "**Project – Properties – C/C++ Indexer**". This brings up the C/C++ Indexer window, as shown below.

Select the "**Full C/C++ Indexer**" even though it has been slandered as "slow but accurate". If you've built a huge project, then you may prefer the faster but less accurate indexer.



Properties for demo_at91sam7_blink_flash

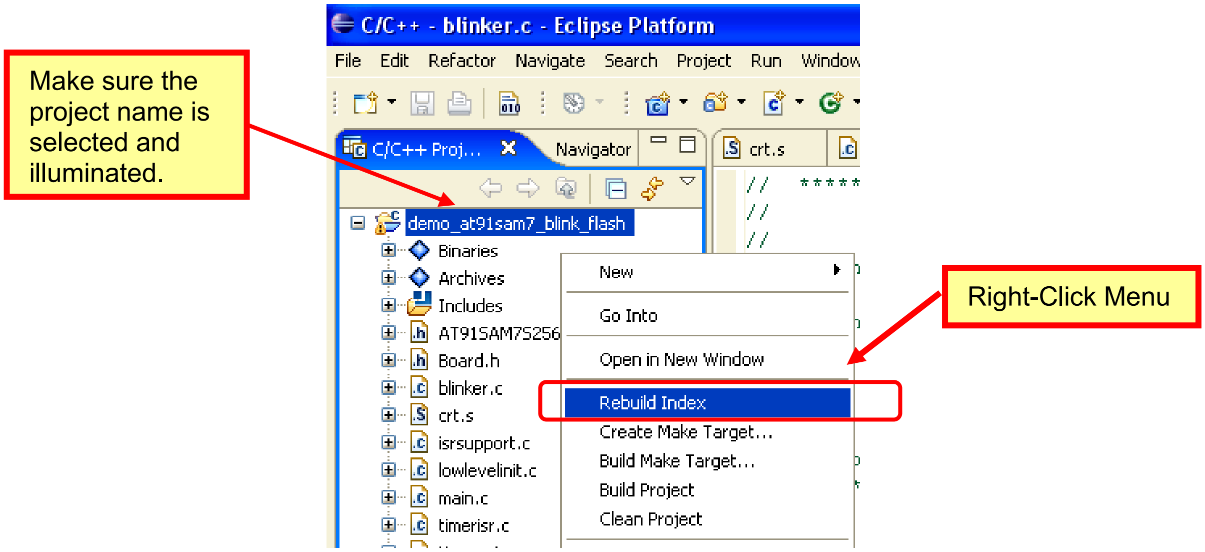
C/C++ Indexer

Available indexers

No Indexer (search-based features will not work correctly)

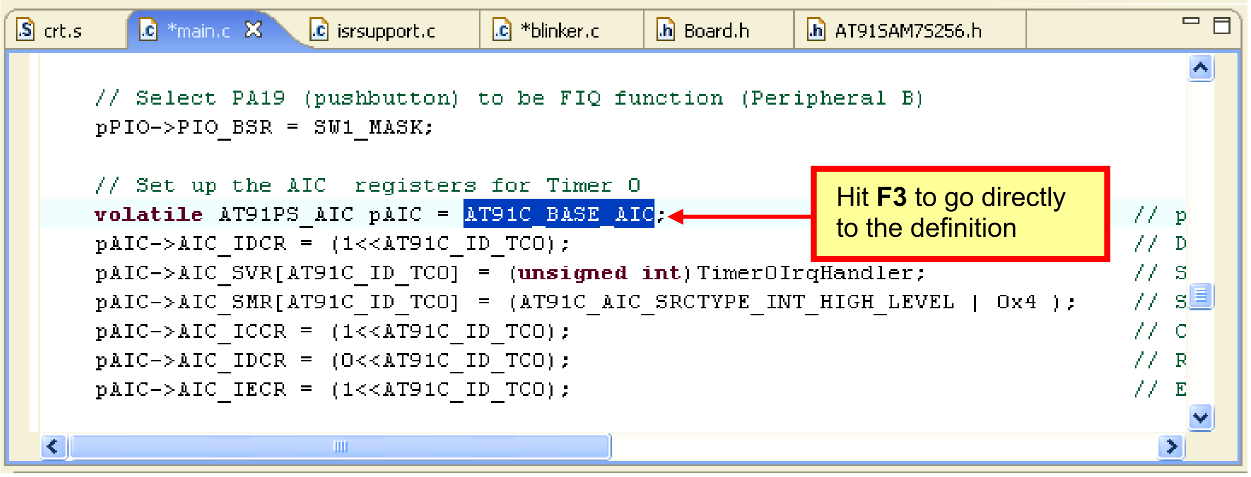
No Indexer (search-based features will not work correctly)
Fast C/C++ Indexer (faster but less accurate)
Full C/C++ Indexer (slow but accurate)

Before doing any heavy duty searching, it behooves you to command Eclipse to rebuild the index. In the C/C++ Projects view on the far left, click in the project name (make sure it is selected). Then use the "right-click" menu to select "**Rebuild Index**" as shown below.


Make sure the project name is selected and illuminated.

Right-Click Menu

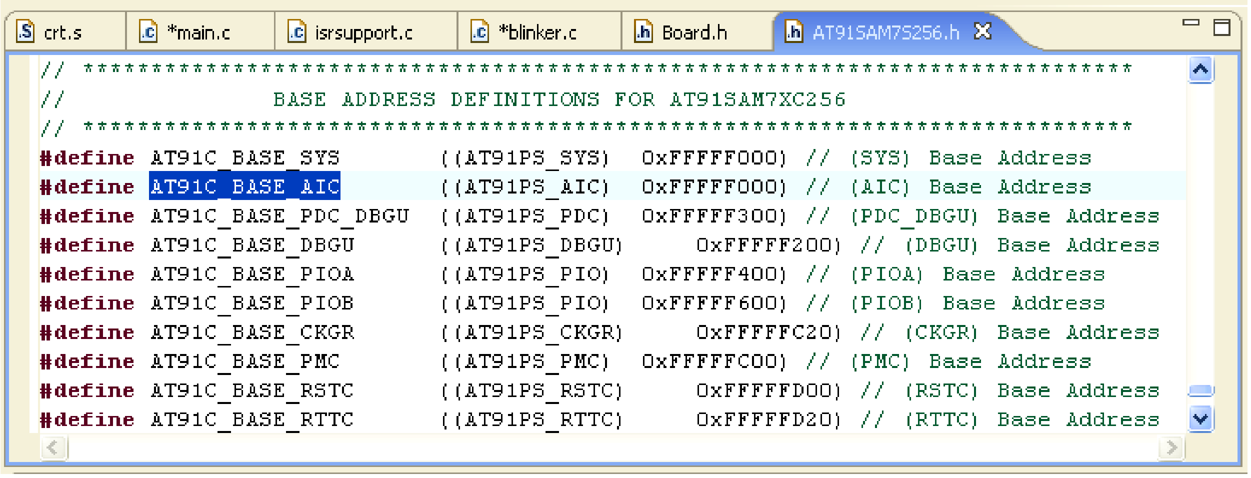
To find the definition of a variable, just select and highlight it and hit the **F3** button on your keyboard.



```
    // Select PA19 (pushbutton) to be FIQ function (Peripheral B)
    pPIO->PIO_BSR = SW1_MASK;

    // Set up the AIC  registers for Timer 0
    volatile AT91PS_AIC pAIC  = AT91C_BASE_AIC;                          // p
    pAIC->AIC_IDCR = (1<<AT91C_ID_TC0);                                  // D
    pAIC->AIC_SVR[AT91C_ID_TC0] = (unsigned int)Timer0IrqHandler;        // S
    pAIC->AIC_SMR[AT91C_ID_TC0] = (AT91C_AIC_SRCTYPE_INT_HIGH_LEVEL | 0x4 );  // S
    pAIC->AIC_ICCR = (1<<AT91C_ID_TC0);                                  // C
    pAIC->AIC_IDCR = (0<<AT91C_ID_TC0);                                  // R
    pAIC->AIC_IECR = (1<<AT91C_ID_TC0);                                  // E
```

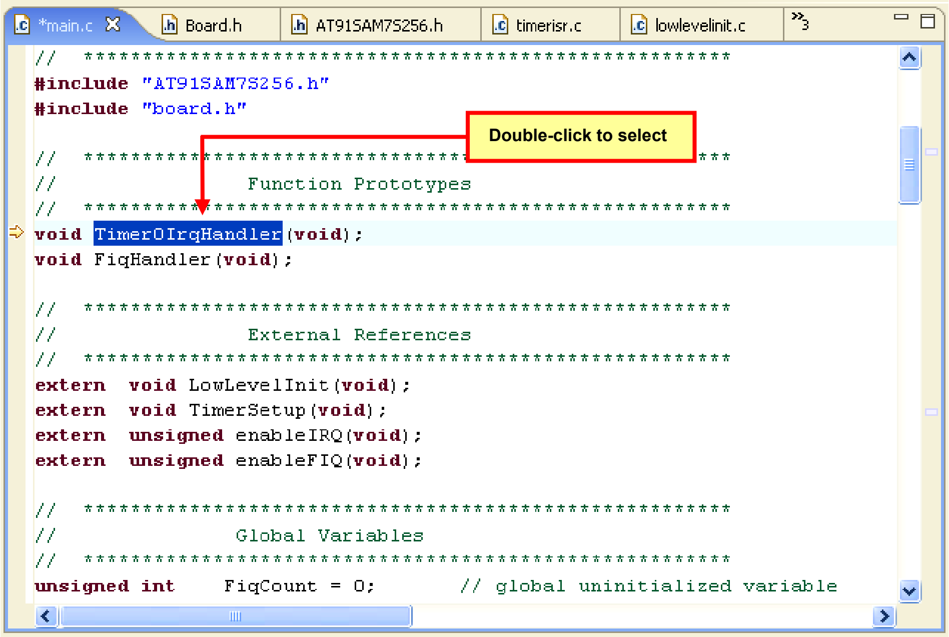
Hit **F3** to go directly to the definition

In a flash, Eclipse will jump to the definition of the constant AT91C_BASE_AIC; note that it's in a different file as shown below.



```
// ****************************************************************************
//              BASE ADDRESS DEFINITIONS FOR AT91SAM7XC256
// ****************************************************************************
#define AT91C_BASE_SYS       ((AT91PS_SYS)  0xFFFFF000) // (SYS) Base Address
#define AT91C_BASE_AIC       ((AT91PS_AIC)  0xFFFFF000) // (AIC) Base Address
#define AT91C_BASE_PDC_DBGU  ((AT91PS_PDC)  0xFFFFF300) // (PDC_DBGU) Base Address
#define AT91C_BASE_DBGU      ((AT91PS_DBGU)     0xFFFFF200) // (DBGU) Base Address
#define AT91C_BASE_PIOA      ((AT91PS_PIO)  0xFFFFF400) // (PIOA) Base Address
#define AT91C_BASE_PIOB      ((AT91PS_PIO)  0xFFFFF600) // (PIOB) Base Address
#define AT91C_BASE_CKGR      ((AT91PS_CKGR)     0xFFFFFC20) // (CKGR) Base Address
#define AT91C_BASE_PMC       ((AT91PS_PMC)  0xFFFFFC00) // (PMC) Base Address
#define AT91C_BASE_RSTC      ((AT91PS_RSTC)     0xFFFFFD00) // (RSTC) Base Address
#define AT91C_BASE_RTTC      ((AT91PS_RTTC)     0xFFFFFD20) // (RTTC) Base Address
```

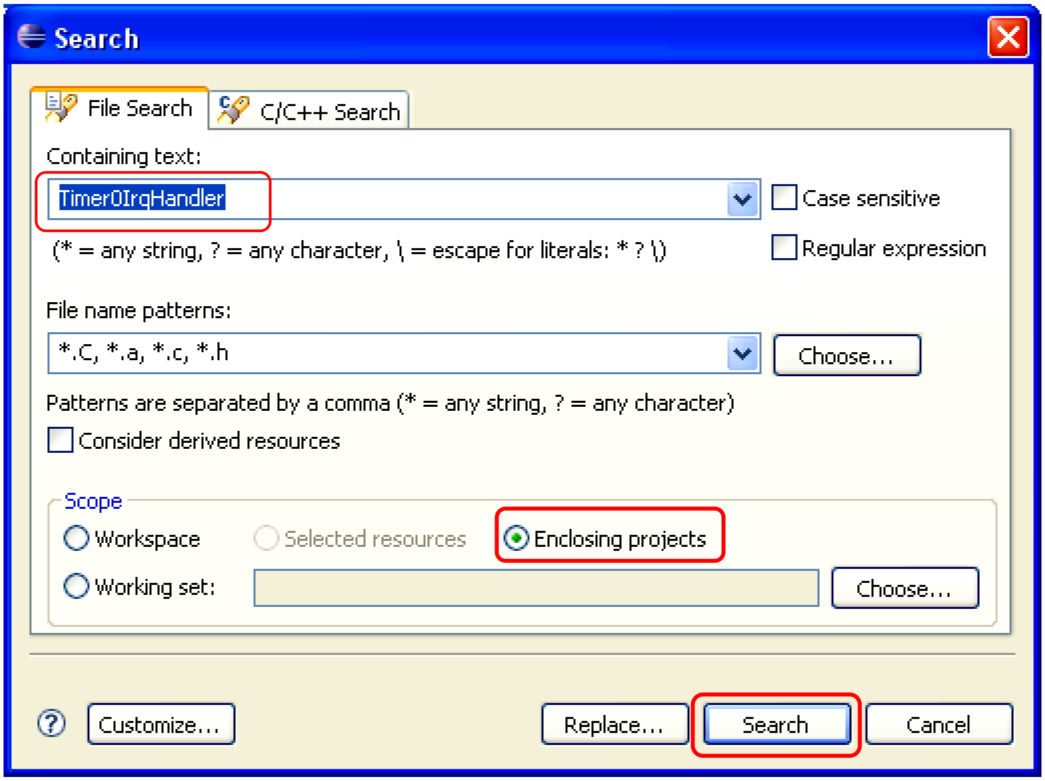
To find all occurrences of a variable, function, constant or any string, select the target text as shown below. Here we'd like to see every occurrence of the function **Timer0IrqHandler** in the entire project.



```
//   *******************************************************
#include "AT91SAM7S256.h"
#include "board.h"

//   *******************************************************
//                Function Prototypes
//   *******************************************************
void TimerOIrqHandler(void);
void FiqHandler(void);

//   *******************************************************
//                External References
//   *******************************************************
extern  void LowLevelInit(void);
extern  void TimerSetup(void);
extern  unsigned enableIRQ(void);
extern  unsigned enableFIQ(void);

//   *******************************************************
//               Global Variables
//   *******************************************************
unsigned int    FiqCount = 0;       // global uninitialized variable
```

Click on the "**Search**" toolbar button.

This will bring up the "Search" window, click on the "**File Search**" tab. By previous selection of the text, the target search text should already appear in the window. Set the scope of the search to "**Enclosing Projects**" and click "**Search**" to command Eclipse to find all occurrences.



Containing text: TimerOIrqHandler

(* = any string, ? = any character, \ = escape for literals: * ? \)

☐ Case sensitive

☐ Regular expression

File name patterns: *.C, *.a, *.c, *.h

Patterns are separated by a comma (* = any string, ? = any character)

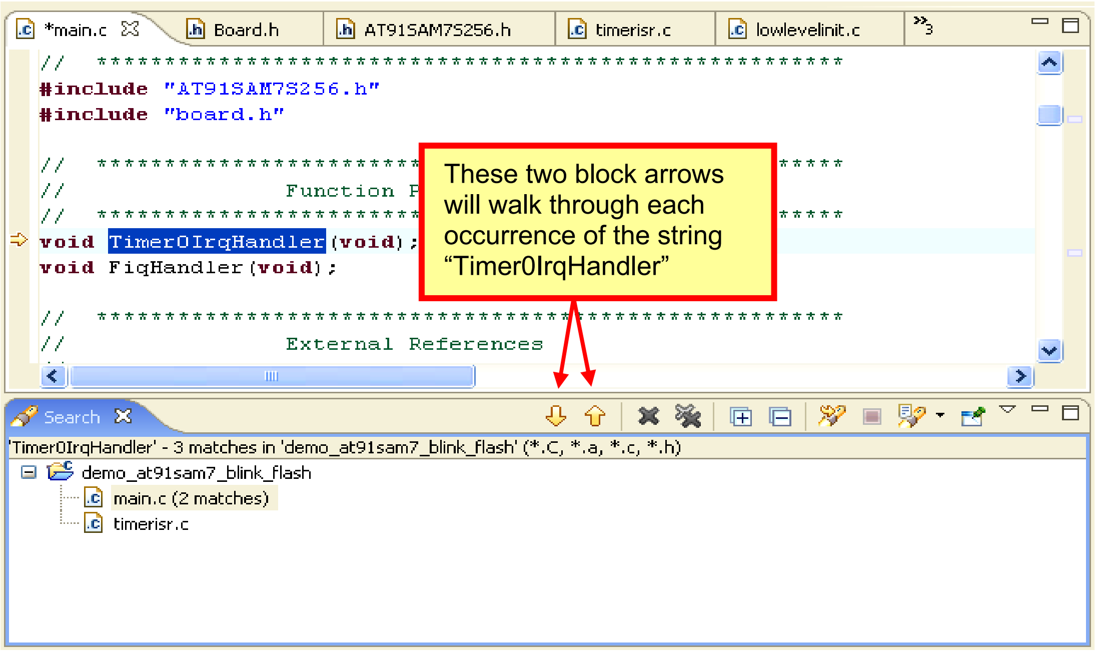
☐ Consider derived resources

Scope

○ Workspace ○ Selected resources ● Enclosing projects

○ Working set:

Now Eclipse will pop-up the “Search” view right below the editing window and it will show 3 occurrences as shown below.



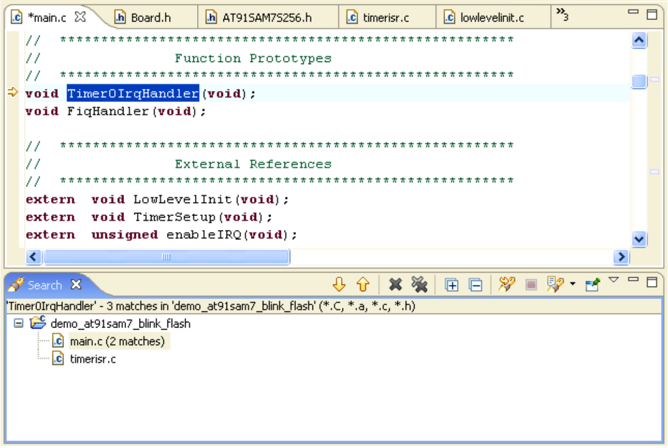
```
//   ************************************************************
#include "AT91SAM7S256.h"
#include "board.h"

//   ************************
//                      Function P
//   ************************
void TimerOIrqHandler(void);
void FiqHandler(void);

//   ************************************************************
//                      External References
```

These two block arrows will walk through each occurrence of the string “Timer0IrqHandler”

'Timer0IrqHandler' - 3 matches in 'demo_at91sam7_blink_flash' (*.C, *.a, *.c, *.h)

```
⊟ demo_at91sam7_blink_flash
    main.c (2 matches)
    timerisr.c
```

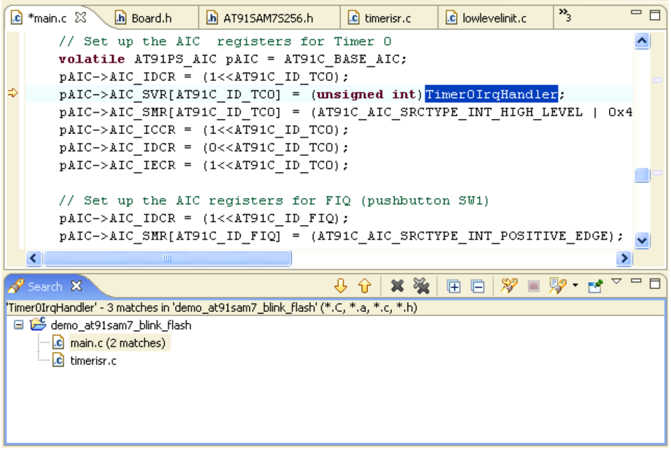
Successive clicks of the yellow block arrows in the Search view will walk through each of the three occurrences of the target string. Note in the sequence directly below, the string appears in two different files.



```
//   ************************************************************
//                      Function Prototypes
//   ************************************************************
void TimerOIrqHandler(void);
void FiqHandler(void);

//   ************************************************************
//                      External References
//   ************************************************************
extern  void LowLevelInit(void);
extern  void TimerSetup(void);
extern  unsigned enableIRQ(void);
```

'Timer0IrqHandler' - 3 matches in 'demo_at91sam7_blink_flash' (*.C, *.a, *.c, *.h)

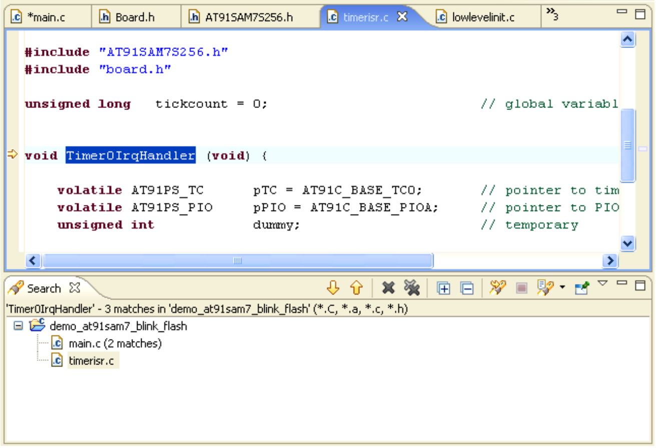
```
⊟ demo_at91sam7_blink_flash
    main.c (2 matches)
    timerisr.c
```



```
    // Set up the AIC  registers for Timer 0
    volatile AT91PS_AIC pAIC = AT91C_BASE_AIC;
    pAIC->AIC_IDCR = (1<<AT91C_ID_TC0);
    pAIC->AIC_SVR[AT91C_ID_TC0] = (unsigned int)TimerOIrqHandler;
    pAIC->AIC_SMR[AT91C_ID_TC0] = (AT91C_AIC_SRCTYPE_INT_HIGH_LEVEL | 0x4
    pAIC->AIC_ICCR = (1<<AT91C_ID_TC0);
    pAIC->AIC_IDCR = (0<<AT91C_ID_TC0);
    pAIC->AIC_IECR = (1<<AT91C_ID_TC0);

    // Set up the AIC registers for FIQ (pushbutton SW1)
    pAIC->AIC_IDCR = (1<<AT91C_ID_FIQ);
    pAIC->AIC_SMR[AT91C_ID_FIQ] = (AT91C_AIC_SRCTYPE_INT_POSITIVE_EDGE);
```

'Timer0IrqHandler' - 3 matches in 'demo_at91sam7_blink_flash' (*.C, *.a, *.c, *.h)

```
⊟ demo_at91sam7_blink_flash
    main.c (2 matches)
    timerisr.c
```



```
#include "AT91SAM7S256.h"
#include "board.h"

unsigned long   tickcount = 0;                          // global variabl


void TimerOIrqHandler (void) {

    volatile AT91PS_TC      pTC = AT91C_BASE_TC0;       // pointer to tim
    volatile AT91PS_PIO     pPIO = AT91C_BASE_PIOA;     // pointer to PIO
    unsigned int            dummy;                      // temporary
```

'Timer0IrqHandler' - 3 matches in 'demo_at91sam7_blink_flash' (*.C, *.a, *.c, *.h)

```
⊟ demo_at91sam7_blink_flash
    main.c (2 matches)
    timerisr.c
```

Discussion of the Source Files – FLASH Version

We will not describe every source file in detail. Most of these files are derived from other Atmel documentation and are simply modified to be compatible with the GNU tools. The source files designed by the author are heavily annotated and you shouldn't have too much trouble understanding them.

AT91SAM7S256.H

This is the standard H file for the Atmel AT91SAM7S256 microprocessor.

```
//  ----------------------------------------------------------------------------
//          ATMEL Microcontroller Software Support  -  ROUSSET  -
//  ----------------------------------------------------------------------------
//  DISCLAIMER:  THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR
//  IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
//  MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE
//  DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT,
//  INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
//  LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,
//  OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
//  LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
//  NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
//  EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
//  ----------------------------------------------------------------------------
// File Name:           AT91SAM7S256.h
// Object:              AT91SAM7S256 definitions
// Generated:           AT91 SW Application Group  11/02/2005 (17:07:34)
//
// CVS Reference:       /AT91SAM7XC256.pl/1.1/Wed Nov  2 13:59:10 2005//

#ifndef       AT91SAM7XC256_H
#define       AT91SAM7XC256_H

typedef volatile unsigned int AT91_REG;              // Hardware register definition

// *****************************************************************************
//              SOFTWARE API DEFINITION  FOR System Peripherals
// *****************************************************************************
typedef struct _AT91S_SYS {
      AT91_REG    AIC_SMR[32];        // Source Mode Register
      AT91_REG    AIC_SVR[32];        // Source Vector Register
      AT91_REG    AIC_IVR;            // IRQ Vector Register
      AT91_REG    AIC_FVR;            // FIQ Vector Register
      AT91_REG    AIC_ISR;            // Interrupt Status Register
      AT91_REG    AIC_IPR;            // Interrupt Pending Register
      AT91_REG    AIC_IMR;            // Interrupt Mask Register
      AT91_REG    AIC_CISR;           // Core Interrupt Status Register
      AT91_REG    Reserved0[2];       //
      AT91_REG    AIC_IECR;           // Interrupt Enable Command Register
      AT91_REG    AIC_IDCR;           // Interrupt Disable Command Register
      AT91_REG    AIC_ICCR;           // Interrupt Clear Command Register
      AT91_REG    AIC_ISCR;           // Interrupt Set Command Register
      AT91_REG    AIC_EOICR;          // End of Interrupt Command Register
      AT91_REG    AIC_SPU;            // Spurious Vector Register
      AT91_REG    AIC_DCR;            // Debug Control Register (Protect)
      AT91_REG    Reserved1[1];       //
      AT91_REG    AIC_FFER;           // Fast Forcing Enable Register
      AT91_REG    AIC_FFDR;           // Fast Forcing Disable Register
      AT91_REG    AIC_FFSR;           // Fast Forcing Status Register
      AT91_REG    Reserved2[45];      //
      AT91_REG    DBGU_CR;            // Control Register
      AT91_REG    DBGU_MR;            // Mode Register
      AT91_REG    DBGU_IER;           // Interrupt Enable Register
      AT91_REG    DBGU_IDR;           // Interrupt Disable Register
      AT91_REG    DBGU_IMR;           // Interrupt Mask Register
      AT91_REG    DBGU_CSR;           // Channel Status Register
      AT91_REG    DBGU_RHR;           // Receiver Holding Register
      AT91_REG    DBGU_THR;           // Transmitter Holding Register
      AT91_REG    DBGU_BRGR;          // Baud Rate Generator Register
      AT91_REG    Reserved3[7];       //
```

```
    AT91_REG    DBGU_CIDR;          // Chip ID Register
    AT91_REG    DBGU_EXID;          // Chip ID Extension Register
    AT91_REG    DBGU_FNTR;          // Force NTRST Register
    AT91_REG    Reserved4[45];      //
    AT91_REG    DBGU_RPR;           // Receive Pointer Register
    AT91_REG    DBGU_RCR;           // Receive Counter Register
    AT91_REG    DBGU_TPR;           // Transmit Pointer Register
    AT91_REG    DBGU_TCR;           // Transmit Counter Register
    AT91_REG    DBGU_RNPR;          // Receive Next Pointer Register
    AT91_REG    DBGU_RNCR;          // Receive Next Counter Register
    AT91_REG    DBGU_TNPR;          // Transmit Next Pointer Register
    AT91_REG    DBGU_TNCR;          // Transmit Next Counter Register
    AT91_REG    DBGU_PTCR;          / PDC Transfer Control Register
    AT91_REG    DBGU_PTSR;          // PDC Transfer Status Register
    AT91_REG    Reserved5[54];      //
    AT91_REG    PIOA_PER;           // PIO Enable Register
    AT91_REG    PIOA_PDR;           // PIO Disable Register
               :
               :
               :
    This is a very large file !
```

BOARD.H

This is the standard Atmel board definition file for the AT91SAM7S-EK Evaluation Board.

```
//----------------------------------------------------------------------------------------------------
//        ATMEL Microcontroller Software Support  -  ROUSSET  -
//----------------------------------------------------------------------------------------------------
// The software is delivered "AS IS" without warranty or condition of any
// kind, either express, implied or statutory. This includes without
// limitation any warranty or condition with respect to merchantability or
// fitness for any particular purpose, or against the infringements of
// intellectual property rights of others.
//----------------------------------------------------------------------------------------------------
// File Name:      Board.h
// Object:         AT91SAM7S Evaluation Board Features Definition File.
//
// Creation:  JPP   16/June/2004
//----------------------------------------------------------------------------------------------------
#ifndef Board_h
#define Board_h

#include "AT91SAM7S256.h"
#define __inline inline

#define true      -1
#define false     0

//-----------------------------------------------
// SAM7Board Memories Definition
//-----------------------------------------------
// The AT91SAM7S64 embeds a 16-Kbyte SRAM bank, and 64 K-Byte Flash

#define  INT_SARM        0x00200000
#define  INT_SARM_REMAP   0x00000000

#define  INT_FLASH 0x00000000
#define  INT_FLASH_REMAP  0x01000000

#define  FLASH_PAGE_NB    512
#define  FLASH_PAGE_SIZE  128

//-----------------------
// Leds Definition
//-----------------------
#define LED1             (1<<0)                     // PA0
#define LED2             (1<<1)                     // PA1
#define LED3             (1<<2)                     // PA2
#define LED4             (1<<3)                     // PA3
#define NB_LEB           4
#define LED_MASK         (LED1|LED2|LED3|LED4)
```

```
//----------------------------------
// Push Buttons Definition
//-----------------------------------
#define SW1_MASK (1<<19)                         // PA19
#define SW2_MASK (1<<20)                         // PA20
#define SW3_MASK (1<<15)                         // PA15
#define SW4_MASK (1<<14)                         // PA14
#define SW_MASK         (SW1_MASK|SW2_MASK|SW3_MASK|SW4_MASK)

#define SW1             (1<<19)                  // PA19
#define SW2             (1<<20)                  // PA20
#define SW3             (1<<15)                  // PA15
#define SW4             (1<<14)                  // PA14

//-------------------------
// USART Definition
//-------------------------
// SUB-D 9 points J3 DBGU*/
#define DBGU_RXD AT91C_PA9_DRXD                  // JP11 must be close
#define DBGU_TXD        AT91C_PA10_DTXD          // JP12 must be close
#define AT91C_DBGU_BAUD115200                    // Baud rate
#define US_RXD_PIN      AT91C_PA5_RXD0           // JP9 must be close
#define US_TXD_PIN      AT91C_PA6_TXD0           // JP7 must be close
#define US_RTS_PINAT91C_PA7_RTS0                 // JP8 must be close
#define US_CTS_PINAT91C_PA8_CTS0                 // JP6 must be close

//--------------
// Master Clock
//--------------
#define EXT_OC      18432000                     // Exetrnal ocilator MAINCK
#define MCK         47923200                     // MCK (PLLRC div by 2)
#define MCKKHz      (MCK/1000)                   //

#endif   // Board_h
```

BLINKER.C

The blinker routine is entered if the application code crashes due to a prefetch abort interrupt, a data abort interrupt or an undefined instruction abort interrupt. The function enters an endless loop and emits an LED blink code identifying the source of the abort. The system must be RESET to recover.

```
//  *****************************************************************************
//                      blinker.c
//
//    Endless loop blinks a code for crash analysis
//
//    Inputs:  Code  -  blink code to display
//             1 = undefined instruction (one blink  ......... long pause)
//             2 = prefetch abort           (two blinks ........ long pause)
//             3 = data abort               (three blinks ...... long pause)
//
//  Author:  James P Lynch  May 12, 2007
//  *****************************************************************************
#include "AT91SAM7S256.h"
#include "board.h"

unsigned long    blinkcount;                                    // global variable

void  blinker(unsigned char   code) {
    volatile AT91PS_PIO       pPIO = AT91C_BASE_PIOA;           // pointer to PIO register structure
    volatile unsigned int     j,k;                              // loop counters

    // endless loop
    while (1) {
        for  (j = code; j != 0; j--) {                          // count out the proper number of blinks
            pPIO->PIO_CODR = LED1;                              // turn LED1 (DS1) on
            for (k = 600000; k != 0; k-- );                     // wait 250 msec
            pPIO->PIO_SODR = LED1;                              // turn LED1 (DS1) off
            for (k = 600000; k != 0; k-- );                     // wait 250 msec
        }
        for (k = 5000000; (code != 0) && (k != 0); k-- );       // wait 2 seconds
        blinkcount++;
    }
}
```

CRT.S

This assembly language startup file includes parts of the standard Atmel startup file with a few changes by the author to conform to the GNU assembler.

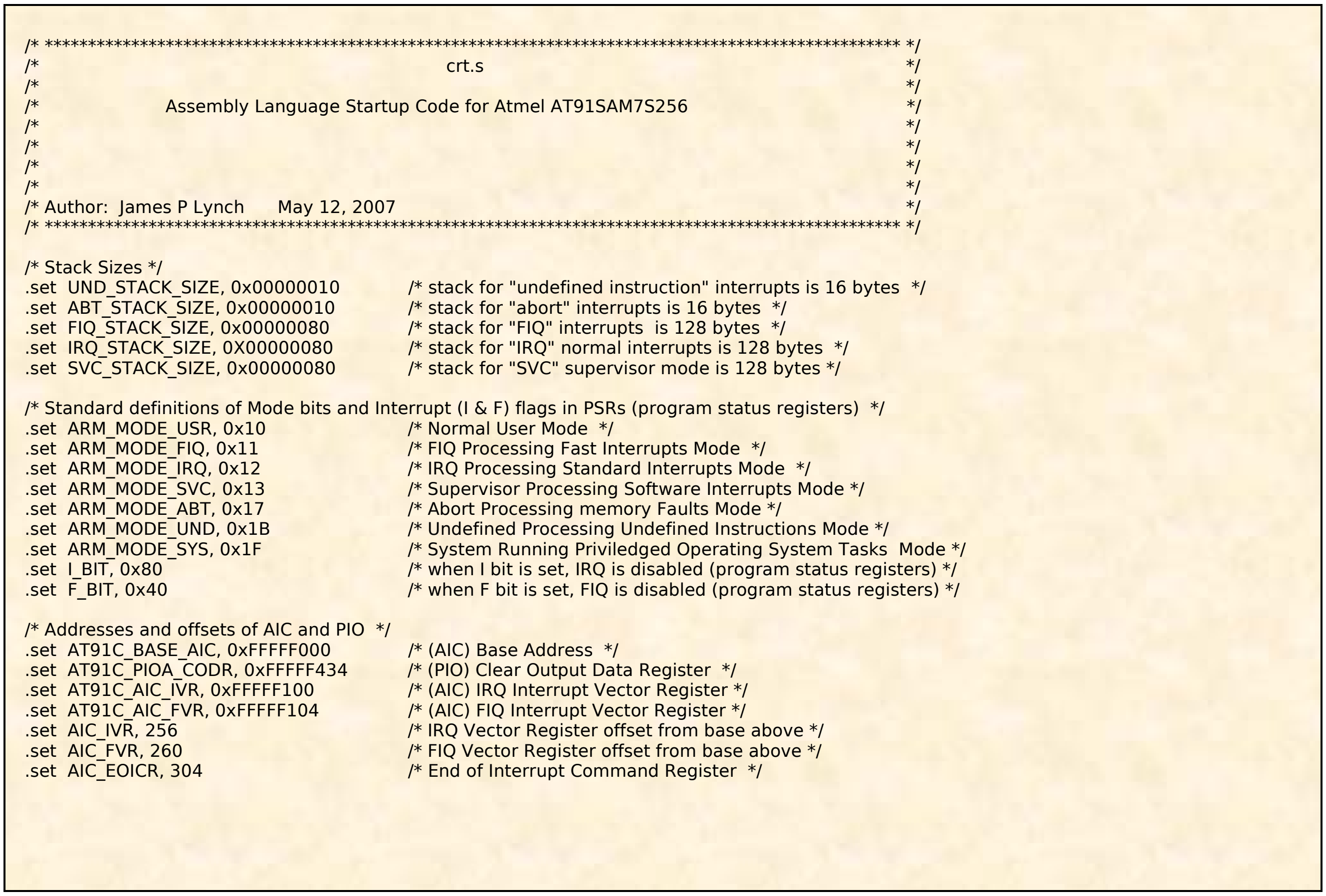
The interrupt vector table is implemented as branch instructions with one interesting difference; the FIQ interrupt service routine is completely implemented right after the vector table. The designers of the ARM microprocessor purposely placed the FIQ vector last in the vector table for this very purpose. This is the most efficient implementation of a FIQ interrupt. The AT91F_Fiq_Handler routine, coded completely in assembler, turns on LED3 and increments a global variable.

The AT91F_Irq_Handler routine is derived from Atmel documentation and supports nested IRQ interrupts. For a detailed technical discussion of this topic, consult pages 336 – 342 in the book “ARM System Developer’s Guide” by Andrew Sloss et. al. Another great advantage of this technique is that the assembly language nested interrupt handler calls a standard C Language function to do most of the work servicing the IRQ interrupt. You don’t have to deal with the GNU C extensions that support ARM interrupt processing.

The start-up code called by the RESET vector sets up 128 byte stacks for the IRQ and FIQ interrupt modes and finally places the CPU in “System” mode with the FIQ and IRQ interrupts disabled. System mode operation allows the main( ) program to enable the IRQ and FIQ interrupts after all peripherals have been properly initialized.

The start-up code also initializes all variables that require it and clears all uninitialized variables to zero before branching to the C Language main( ) routine.

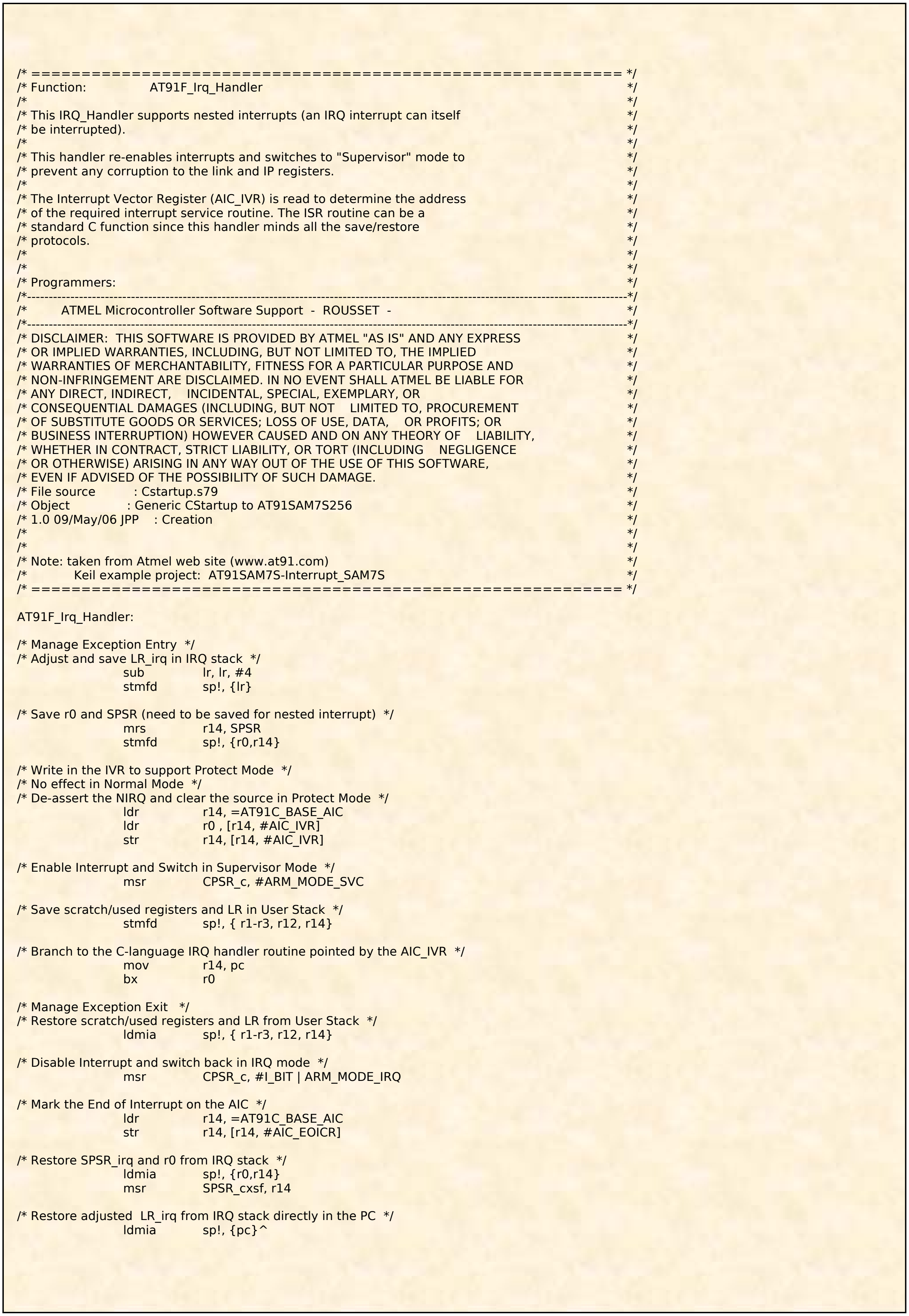
The author would like to thank Eric Pasquier for noting deficiencies in the Revision B version of the IRQ handler. As per Eric’s suggestions, the standard Atmel IRQ code is used in this revision.



```
/* ******************************************************************************************************** */
/*                                                    crt.s                                                  */
/*                                                                                                           */
/*                  Assembly Language Startup Code for Atmel AT91SAM7S256                                    */
/*                                                                                                           */
/*                                                                                                           */
/*                                                                                                           */
/*                                                                                                           */
/* Author:  James P Lynch      May 12, 2007                                                                  */
/* ******************************************************************************************************** */

/* Stack Sizes */
.set  UND_STACK_SIZE, 0x00000010          /* stack for "undefined instruction" interrupts is 16 bytes  */
.set  ABT_STACK_SIZE, 0x00000010          /* stack for "abort" interrupts is 16 bytes  */
.set  FIQ_STACK_SIZE, 0x00000080          /* stack for "FIQ" interrupts  is 128 bytes  */
.set  IRQ_STACK_SIZE, 0X00000080          /* stack for "IRQ" normal interrupts is 128 bytes  */
.set  SVC_STACK_SIZE, 0x00000080          /* stack for "SVC" supervisor mode is 128 bytes */

/* Standard definitions of Mode bits and Interrupt (I & F) flags in PSRs (program status registers)  */
.set  ARM_MODE_USR, 0x10                  /* Normal User Mode  */
.set  ARM_MODE_FIQ, 0x11                  /* FIQ Processing Fast Interrupts Mode  */
.set  ARM_MODE_IRQ, 0x12                  /* IRQ Processing Standard Interrupts Mode  */
.set  ARM_MODE_SVC, 0x13                  /* Supervisor Processing Software Interrupts Mode */
.set  ARM_MODE_ABT, 0x17                  /* Abort Processing memory Faults Mode */
.set  ARM_MODE_UND, 0x1B                  /* Undefined Processing Undefined Instructions Mode */
.set  ARM_MODE_SYS, 0x1F                  /* System Running Priviledged Operating System Tasks  Mode */
.set  I_BIT, 0x80                         /* when I bit is set, IRQ is disabled (program status registers) */
.set  F_BIT, 0x40                         /* when F bit is set, FIQ is disabled (program status registers) */

/* Addresses and offsets of AIC and PIO  */
.set  AT91C_BASE_AIC, 0xFFFFF000          /* (AIC) Base Address  */
.set  AT91C_PIOA_CODR, 0xFFFFF434         /* (PIO) Clear Output Data Register  */
.set  AT91C_AIC_IVR, 0xFFFFF100           /* (AIC) IRQ Interrupt Vector Register */
.set  AT91C_AIC_FVR, 0xFFFFF104           /* (AIC) FIQ Interrupt Vector Register */
.set  AIC_IVR, 256                        /* IRQ Vector Register offset from base above */
.set  AIC_FVR, 260                        /* FIQ Vector Register offset from base above */
.set  AIC_EOICR, 304                      /* End of Interrupt Command Register  */
```

```
/* identify all GLOBAL symbols  */
.global _vec_reset
.global _vec_undef
.global _vec_swi
.global _vec_pabt
.global _vec_dabt
.global _vec_rsv
.global _vec_irq
.global _vec_fiq
.global AT91F_Irq_Handler
.global   AT91F_Fiq_Handler
.global   AT91F_Default_FIQ_handler
.global   AT91F_Default_IRQ_handler
.global   AT91F_Spurious_handler
.global   AT91F_Dabt_Handler
.global   AT91F_Pabt_Handler
.global   AT91F_Undef_Handler

/* GNU assembler controls  */
.text                                   /* all assembler code that follows will go into .text section  */
.arm                                    /* compile for 32-bit ARM instruction set  */
.align                                  /* align section on 32-bit boundary  */

/* ============================================================  */
/*                         VECTOR TABLE                              */
/*                                                                   */
/*    Must be located in FLASH at address 0x00000000                 */
/*                                                                   */
/*    Easy to do if this file crt.s is first in the list             */
/*    for the linker step in the makefile, e.g.                      */
/*                                                                   */
/*        $(LD) $(LFLAGS) -o main.out  crt.o main.o                  */
/*                                                                   */
/* ============================================================  */

_vec_reset:     b       _init_reset             /* RESET vector - must be at 0x00000000  */
_vec_undef:     b       AT91F_Undef_Handler     /* Undefined Instruction vector  */
_vec_swi:       b       _vec_swi                /* Software Interrupt vector  */
_vec_pabt:      b       AT91F_Pabt_Handler      /* Prefetch abort vector  */
_vec_dabt:      b       AT91F_Dabt_Handler      /* Data abort vector  */
_vec_rsv:       nop                             /* Reserved vector  */
_vec_irq:       b       AT91F_Irq_Handler       /* Interrupt Request (IRQ) vector */
_vec_fiq:                                       /* Fast interrupt request (FIQ) vector */


/* ============================================================  */
/* Function:            AT91F_Fiq_Handler                            */
/*                                                                   */
/* The FIQ interrupt asserts when switch SW1 is pressed.             */
/*                                                                   */
/* This simple FIQ handler turns on LED3 (Port PA2). The LED3 will be */
/* turned off by the background loop in main() thus giving a visual  */
/* indication that the interrupt has occurred.                       */
/*                                                                   */
/* This FIQ_Handler supports non-nested FIQ interrupts (a FIQ interrupt */
/* cannot itself be interrupted).                                    */
/*                                                                   */
/* The Fast Interrupt Vector Register (AIC_FVR) is read to clear the interrupt */
/*                                                                   */
/* A global variable FiqCount is also incremented.                   */
/*                                                                   */
/* Remember that switch SW1 is not debounced, so the FIQ interrupt may */
/* occur more than once for a single button push.                    */
/*                                                                   */
/* Programmer: James P Lynch                                         */
/* ============================================================  */
AT91F_Fiq_Handler:

/* Adjust LR_irq */
                sub     lr, lr, #4

/* Read the AIC Fast Interrupt Vector register to clear the interrupt */
                ldr     r12, =AT91C_AIC_FVR
                ldr     r11, [r12]

/* Turn on LED3 (write 0x0008 to PIOA_CODR at 0xFFFFF434) */
                ldr     r12, =AT91C_PIOA_CODR
                mov     r11, #0x04
                str     r11, [r12]



/* Increment the _FiqCount variable */
                ldr     r12, =FiqCount
                ldr     r11, [r12]
                add     r11, r11, #1
                str     r11, [r12]

/* Return from Fiq interrupt */
                movs    pc, lr
```

```
/* ================================================================== */
/*                          _init_reset Handler                          */
/*                                                                    */
/*     RESET vector 0x00000000 branches to here.                      */
/*                                                                    */
/*     ARM microprocessor begins execution after RESET at address 0x00000000   */
/*     in Supervisor mode with interrupts disabled!                   */
/*                                                                    */
/*     _init_reset handler:   creates a stack for each ARM mode.      */
/*                            sets up a stack pointer for each ARM mode.   */
/*                            turns off interrupts in each mode.      */
/*                            leaves CPU in SYS (System) mode.        */
/*                                                                    */
/*                            block copies the initializers to .data section   */
/*                            clears the .bss section to zero         */
/*                                                                    */
/*                            branches to main( )                     */
/* ================================================================== */


.text                        /* all assembler code that follows will go into .text section   */
.align                       /* align section on 32-bit boundary  */


_init_reset:
              /* Setup a stack for each mode with interrupts initially disabled. */
              ldr   r0, =_stack_end                               /* r0 = top-of-stack  */

              msr   CPSR_c, #ARM_MODE_UND|I_BIT|F_BIT             /* switch to Undefined Instruction Mode  */
              mov   sp, r0                                        /* set stack pointer for UND mode  */
              sub   r0, r0, #UND_STACK_SIZE                       /* adjust r0 past UND stack  */

              msr   CPSR_c, #ARM_MODE_ABT|I_BIT|F_BIT             /* switch to Abort Mode */
              mov   sp, r0                                        /* set stack pointer for ABT mode  */
              sub   r0, r0, #ABT_STACK_SIZE                       /* adjust r0 past ABT stack  */

              msr   CPSR_c, #ARM_MODE_FIQ|I_BIT|F_BIT             /* switch to FIQ Mode */
              mov   sp, r0                                        /* set stack pointer for FIQ mode  */
              sub   r0, r0, #FIQ_STACK_SIZE                       /* adjust r0 past FIQ stack  */

              msr   CPSR_c, #ARM_MODE_IRQ|I_BIT|F_BIT             /* switch to IRQ Mode */
              mov   sp, r0                                        /* set stack pointer for IRQ mode  */
              sub   r0, r0, #IRQ_STACK_SIZE                       /* adjust r0 past IRQ stack  */

              msr   CPSR_c, #ARM_MODE_SVC|I_BIT|F_BIT             /* switch to Supervisor Mode */
              mov   sp, r0                                        /* set stack pointer for SVC mode  */
              sub   r0, r0, #SVC_STACK_SIZE                       /* adjust r0 past SVC stack  */

              msr   CPSR_c, #ARM_MODE_SYS|I_BIT|F_BIT             /* switch to System Mode */
              mov   sp, r0                                        /* set stack pointer for SYS mode  */
                                                                  /* we now start execution in SYSTEM mode */
                                                                  /* This is exactly like USER mode (same stack) */
                                                                  /* but SYSTEM mode has more privileges */

              /* copy initialized variables .data section  (Copy from ROM to RAM) */
              ldr     R1, =_etext
              ldr     R2, =_data
              ldr     R3, =_edata
1:            cmp   R2, R3
              ldrlo   R0, [R1], #4
              strlo   R0, [R2], #4
              blo     1b

              /* Clear uninitialized variables .bss section (Zero init)  */
              mov   R0, #0
              ldr     R1, =_bss_start
              ldr     R2, =_bss_end
2:            cmp    R1, R2
              strlo   R0, [R1], #4
              blo     2b

              /* Enter the C code  */
              b       main
```



```
/* ================================================================ */
/* Function:              AT91F_Irq_Handler                                      */
/*                                                                               */
/* This IRQ_Handler supports nested interrupts (an IRQ interrupt can itself      */
/* be interrupted).                                                              */
/*                                                                               */
/* This handler re-enables interrupts and switches to "Supervisor" mode to       */
/* prevent any corruption to the link and IP registers.                          */
/*                                                                               */
/* The Interrupt Vector Register (AIC_IVR) is read to determine the address      */
/* of the required interrupt service routine. The ISR routine can be a           */
/* standard C function since this handler minds all the save/restore             */
/* protocols.                                                                    */
/*                                                                               */
/*                                                                               */
/* Programmers:                                                                  */
/*-------------------------------------------------------------------------------*/
/*        ATMEL Microcontroller Software Support  -  ROUSSET  -                  */
/*-------------------------------------------------------------------------------*/
/* DISCLAIMER:  THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS       */
/* OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED             */
/* WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND           */
/* NON-INFRINGEMENT ARE DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR        */
/* ANY DIRECT, INDIRECT,    INCIDENTAL, SPECIAL, EXEMPLARY, OR                   */
/* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT    LIMITED TO, PROCUREMENT          */
/* OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA,    OR PROFITS; OR         */
/* BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF    LIABILITY,      */
/* WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING    NEGLIGENCE       */
/* OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,             */
/* EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.                            */
/* File source          : Cstartup.s79                                           */
/* Object               : Generic CStartup to AT91SAM7S256                       */
/* 1.0 09/May/06 JPP    : Creation                                               */
/*                                                                               */
/*                                                                               */
/* Note: taken from Atmel web site (www.at91.com)                                */
/*           Keil example project:  AT91SAM7S-Interrupt_SAM7S                    */
/* ================================================================ */

AT91F_Irq_Handler:

/* Manage Exception Entry  */
/* Adjust and save LR_irq in IRQ stack  */
                sub         lr, lr, #4
                stmfd       sp!, {lr}

/* Save r0 and SPSR (need to be saved for nested interrupt)  */
                mrs         r14, SPSR
                stmfd       sp!, {r0,r14}

/* Write in the IVR to support Protect Mode  */
/* No effect in Normal Mode  */
/* De-assert the NIRQ and clear the source in Protect Mode  */
                ldr         r14, =AT91C_BASE_AIC
                ldr         r0 , [r14, #AIC_IVR]
                str         r14, [r14, #AIC_IVR]

/* Enable Interrupt and Switch in Supervisor Mode  */
                msr         CPSR_c, #ARM_MODE_SVC

/* Save scratch/used registers and LR in User Stack  */
                stmfd       sp!, { r1-r3, r12, r14}

/* Branch to the C-language IRQ handler routine pointed by the AIC_IVR  */
                mov         r14, pc
                bx          r0

/* Manage Exception Exit   */
/* Restore scratch/used registers and LR from User Stack  */
                ldmia       sp!, { r1-r3, r12, r14}

/* Disable Interrupt and switch back in IRQ mode  */
                msr         CPSR_c, #I_BIT | ARM_MODE_IRQ

/* Mark the End of Interrupt on the AIC  */
                ldr         r14, =AT91C_BASE_AIC
                str         r14, [r14, #AIC_EOICR]

/* Restore SPSR_irq and r0 from IRQ stack  */
                ldmia       sp!, {r0,r14}
                msr         SPSR_cxsf, r14

/* Restore adjusted  LR_irq from IRQ stack directly in the PC  */
                ldmia       sp!, {pc}^
```

```
/* ================================================================ */
/* Function:              AT91F_Dabt_Handler                                          */
/*                                                                                    */
/* Entered on Data Abort exception.                                                   */
/* Enters blink routine  (3 blinks followed by a pause)                               */
/* processor hangs in the blink loop forever                                          */
/*                                                                                    */
/* ================================================================ */
AT91F_Dabt_Handler:       mov     R0, #3
                          b       blinker


/* ================================================================ */
/* Function:              AT91F_Pabt_Handler                                          */
/*                                                                                    */
/* Entered on Prefetch Abort exception.                                               */
/* Enters blink routine  (2 blinks followed by a pause)                               */
/* processor hangs in the blink loop forever                                          */
/*                                                                                    */
/* ================================================================ */
AT91F_Pabt_Handler:       mov     R0, #2
                          b       blinker


/* ================================================================ */
/* Function:              AT91F_Undef_Handler                                         */
/*                                                                                    */
/* Entered on Undefined Instruction exception.                                        */
/* Enters blink routine  (1 blinks followed by a pause)                               */
/* processor hangs in the blink loop forever                                          */
/*                                                                                    */
/* ================================================================ */
AT91F_Undef_Handler:      mov     R0, #1
                          b       blinker


AT91F_Default_FIQ_handler:    b     AT91F_Default_FIQ_handler

AT91F_Default_IRQ_handler:    b     AT91F_Default_IRQ_handler

AT91F_Spurious_handler:       b     AT91F_Spurious_handler

.end
```

ISRSUPPORT.C

The isrsupport module is adapted from an example posted to the Yahoo LPC2000 user's group by Bill Knight and contains various utility functions to enable/disable interrupts, etc.

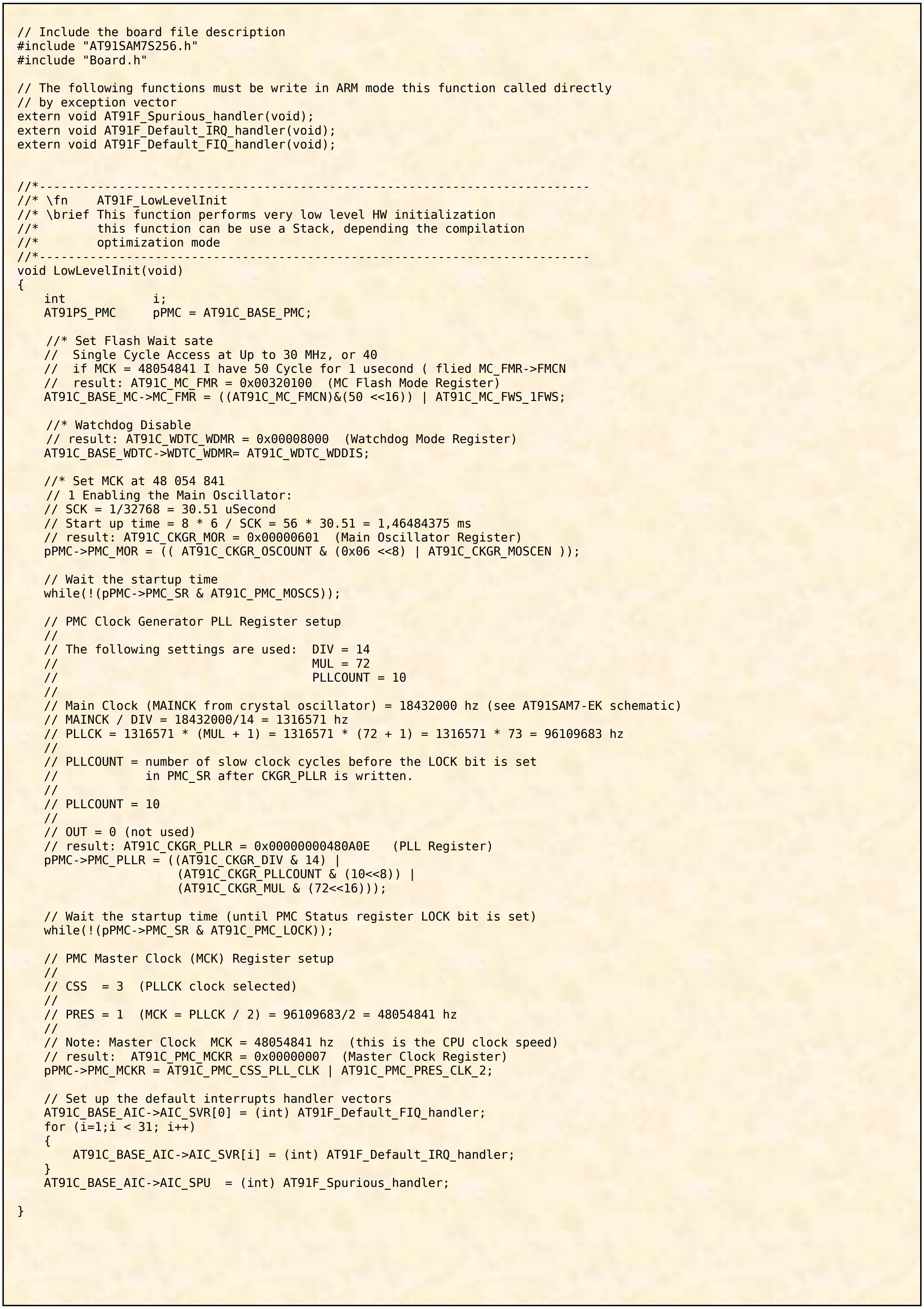
```
//  *********************************************************************************************
//
//  File Name:     isrsupport.c
//  Title:             interrupt enable/disable functions
//
//
//  This module provides the interface routines for setting up and controlling the various interrupt
//   modes present on the ARM processor.
//
//  Copyright 2004, R O SoftWare
//  No guarantees, warrantees, or promises, implied or otherwise.
//  May be used for hobby or commercial purposes provided copyright
//  notice remains intact.
//
//  Note from Jim Lynch:
//  This module was developed by Bill Knight, RO Software and used with his permission.
//  Taken from the Yahoo LPC2000 User's Group - Files Section 'UT050418A.ZIP'
//  Specifically, the module armVIC.c with the include file references removed
//
//  *********************************************************************************************

#define IRQ_MASK 0x00000080
#define FIQ_MASK 0x00000040
#define INT_MASK (IRQ_MASK | FIQ_MASK)
```

```
static inline unsigned __get_cpsr(void)
{
      unsigned long retval;
      asm volatile (" mrs  %0, cpsr" : "=r" (retval) : /* no inputs */  );
      return retval;
}

static inline void __set_cpsr(unsigned val)
{
      asm volatile (" msr  cpsr, %0" : /* no outputs */ : "r" (val)  );
}

unsigned disableIRQ(void)
{
      unsigned _cpsr;
      _cpsr = __get_cpsr();
      __set_cpsr(_cpsr | IRQ_MASK);
      return _cpsr;
}

unsigned restoreIRQ(unsigned oldCPSR)
{
      unsigned _cpsr;
      _cpsr = __get_cpsr();
      __set_cpsr((_cpsr & ~IRQ_MASK) | (oldCPSR & IRQ_MASK));
      return _cpsr;
}

unsigned enableIRQ(void)
{
      unsigned _cpsr;
      _cpsr = __get_cpsr();
      __set_cpsr(_cpsr & ~IRQ_MASK);
      return _cpsr;
}

unsigned disableFIQ(void)
{
      unsigned _cpsr;
      _cpsr = __get_cpsr();
      __set_cpsr(_cpsr | FIQ_MASK);
      return _cpsr;
}

unsigned restoreFIQ(unsigned oldCPSR)
{
      unsigned _cpsr;
      _cpsr = __get_cpsr();
      __set_cpsr((_cpsr & ~FIQ_MASK) | (oldCPSR & FIQ_MASK));
      return _cpsr;
}

unsigned enableFIQ(void)
{
      unsigned _cpsr;
      _cpsr = __get_cpsr();
      __set_cpsr(_cpsr & ~FIQ_MASK);
      return _cpsr;
}
```

LOWLEVELINIT.C

This function, developed by Atmel Technical Support, initializes the PLL clock system. Some annotation has been extended by the author.

```
//  ----------------------------------------------------------------------------
//           ATMEL Microcontroller Software Support  -  ROUSSET  -
//  ----------------------------------------------------------------------------
//   The software is delivered "AS IS" without warranty or condition of any
//   kind, either express, implied or statutory. This includes without
//   limitation any warranty or condition with respect to merchantability or
//   fitness for any particular purpose, or against the infringements of
//   intellectual property rights of others.
//  ----------------------------------------------------------------------------
//   File Name           : Cstartup_SAM7.c
//   Object              : Low level initializations written in C for IAR tools
//   1.0   08/Sep/04 JPP : Creation
//   1.10  10/Sep/04 JPP : Update AT91C_CKGR_PLLCOUNT filed
//  ----------------------------------------------------------------------------
```



```
// Include the board file description
#include "AT91SAM7S256.h"
#include "Board.h"

// The following functions must be write in ARM mode this function called directly
// by exception vector
extern void AT91F_Spurious_handler(void);
extern void AT91F_Default_IRQ_handler(void);
extern void AT91F_Default_FIQ_handler(void);


//*----------------------------------------------------------------------------
//* \fn    AT91F_LowLevelInit
//* \brief This function performs very low level HW initialization
//*        this function can be use a Stack, depending the compilation
//*        optimization mode
//*----------------------------------------------------------------------------
void LowLevelInit(void)
{
    int            i;
    AT91PS_PMC     pPMC = AT91C_BASE_PMC;

    //* Set Flash Wait sate
    //  Single Cycle Access at Up to 30 MHz, or 40
    //  if MCK = 48054841 I have 50 Cycle for 1 usecond ( flied MC_FMR->FMCN
    //  result: AT91C_MC_FMR = 0x00320100  (MC Flash Mode Register)
    AT91C_BASE_MC->MC_FMR = ((AT91C_MC_FMCN)&(50 <<16)) | AT91C_MC_FWS_1FWS;

    //* Watchdog Disable
    // result: AT91C_WDTC_WDMR = 0x00008000  (Watchdog Mode Register)
    AT91C_BASE_WDTC->WDTC_WDMR= AT91C_WDTC_WDDIS;

    //* Set MCK at 48 054 841
    // 1 Enabling the Main Oscillator:
    // SCK = 1/32768 = 30.51 uSecond
    // Start up time = 8 * 6 / SCK = 56 * 30.51 = 1,46484375 ms
    // result: AT91C_CKGR_MOR = 0x00000601  (Main Oscillator Register)
    pPMC->PMC_MOR = (( AT91C_CKGR_OSCOUNT & (0x06 <<8) | AT91C_CKGR_MOSCEN ));

    // Wait the startup time
    while(!(pPMC->PMC_SR & AT91C_PMC_MOSCS));

    // PMC Clock Generator PLL Register setup
    //
    // The following settings are used:  DIV = 14
    //                                   MUL = 72
    //                                   PLLCOUNT = 10
    //
    // Main Clock (MAINCK from crystal oscillator) = 18432000 hz (see AT91SAM7-EK schematic)
    // MAINCK / DIV = 18432000/14 = 1316571 hz
    // PLLCK = 1316571 * (MUL + 1) = 1316571 * (72 + 1) = 1316571 * 73 = 96109683 hz
    //
    // PLLCOUNT = number of slow clock cycles before the LOCK bit is set
    //            in PMC_SR after CKGR_PLLR is written.
    //
    // PLLCOUNT = 10
    //
    // OUT = 0 (not used)
    // result: AT91C_CKGR_PLLR = 0x00000000480A0E   (PLL Register)
    pPMC->PMC_PLLR = ((AT91C_CKGR_DIV & 14) |
                      (AT91C_CKGR_PLLCOUNT & (10<<8)) |
                      (AT91C_CKGR_MUL & (72<<16)));

    // Wait the startup time (until PMC Status register LOCK bit is set)
    while(!(pPMC->PMC_SR & AT91C_PMC_LOCK));

    // PMC Master Clock (MCK) Register setup
    //
    // CSS  = 3  (PLLCK clock selected)
    //
    // PRES = 1  (MCK = PLLCK / 2) = 96109683/2 = 48054841 hz
    //
    // Note: Master Clock  MCK = 48054841 hz  (this is the CPU clock speed)
    // result:  AT91C_PMC_MCKR = 0x00000007  (Master Clock Register)
    pPMC->PMC_MCKR = AT91C_PMC_CSS_PLL_CLK | AT91C_PMC_PRES_CLK_2;

    // Set up the default interrupts handler vectors
    AT91C_BASE_AIC->AIC_SVR[0] = (int) AT91F_Default_FIQ_handler;
    for (i=1;i < 31; i++)
    {
        AT91C_BASE_AIC->AIC_SVR[i] = (int) AT91F_Default_IRQ_handler;
    }
    AT91C_BASE_AIC->AIC_SPU  = (int) AT91F_Spurious_handler;

}
```

MAIN.C

The Main() program, designed by the author, provides a background wait loop that flashes LED1 at approximately a 1 Hz rate, flashes LED2 at a 10 Hz rate triggered by a Timer0 IRQ interrupt, and flashes LED3 whenever you push switch SW1 which triggers a FIQ interrupt . There are also plenty of variables for debugging practice.

There are code snippets, currently commented out, that can trigger an ABORT interrupt that results in a crash blinker code that will identify the source of the abort.

```
//  *******************************************************************************
//                       main.c
//
//     Demonstration program for Atmel AT91SAM7S256-EK Evaluation Board
//
//     blinks LED0 (pin PA0) with an endless loop
//     blinks LED1 (pin PA1) using timer0 interrupt (200 msec rate)
//    switch SW1 (PA19) triggers FIQ interrupt, turns on LED2 (Pin PA2)
//     plenty of variables for debugger practice
//
//  Author:  James P Lynch  May 12, 2007
//  *******************************************************************************


//  *******************************************************
//               Header  Files
//  *******************************************************
#include "AT91SAM7S256.h"
#include "board.h"
#include "string.h"
#include "math.h"
#include "stdlib.h"


//  *******************************************************
//               Function Prototypes
//  *******************************************************
void Timer0IrqHandler(void);
void FiqHandler(void);


//  *******************************************************
//               External References
//  *******************************************************
extern  void LowLevelInit(void);
extern  void TimerSetup(void);
extern  unsigned enableIRQ(void);
extern  unsigned enableFIQ(void);


//  *******************************************************
//              Global Variables
//  *******************************************************
unsigned int   FiqCount = 0;                         // global uninitialized variable
int            q;                                    // global uninitialized variable
int            r;                                    // global uninitialized variable
int            s;                                    // global uninitialized variable
int            m = 2;                                // global initialized variable
int            n = 3;                                // global initialized variable
int            o = 6;                                // global initialized variable

struct comms {
    int    nBytes;
    char   *pBuf;
    char   Buffer[32];
}  Channel = {5, &Channel.Buffer[0], {"Faster than a speeding bullet"}};
```

```
//  *****************************************************************************
//                        main.c
//
//      Demonstration program for Atmel AT91SAM7S256-EK Evaluation Board
//
//      blinks LED0 (pin PA0) with an endless loop
//      blinks LED1 (pin PA1) using timer0 interrupt (200 msec rate)
//      switch SW1 (PA19) triggers FIQ interrupt, turns on LED2 (Pin PA2)
//      plenty of variables for debugger practice
//
//      Author:  James P Lynch  May 12, 2007
//
//  *****************************************************************************
int main (void) {

    // lots of variables for debugging practice
    int           a, b, c;                          // uninitialized variables
    char          d;                                // uninitialized variable
    int           w = 1;                            // initialized variable
    int           k = 2;                            // initialized variable
    static long   x = 5;                            // static initialized variable
    static char   y = 0x04;                         // static initialized variable
    const char    *pText = "The rain in Spain";     // initialized string pointer variable
    struct EntryLock {                              // initialized structure variable
        long      Key;
        int       nAccesses;
        char      Name[17];
    }  Access = {14705, 0, "Sophie Marceau"};
    unsigned long j;                                // loop counter (stack variable)
    unsigned long IdleCount = 0;                    // idle loop blink counter (2x)
    int           *p;                               // pointer to 32-bit word
    typedef void  (*FnPtr)(void);                   // create a "pointer to function" type
    FnPtr         pFnPtr;                           // pointer to a function
    double        x5;                               // variable to test library function
    double        y5 = -172.451;                    // variable to test library function
    const char    DigitBuffer[] = "16383";          // variable to test library function
    long          n;                                // variable to test library function

    // Initialize the Atmel AT91SAM7S256 (watchdog, PLL clock, default interrupts, etc.)
    LowLevelInit();


    // enable the Timer0 peripheral clock
    volatile AT91PS_PMC   pPMC = AT91C_BASE_PMC;    // pointer to PMC data structure
    pPMC->PMC_PCER = (1<<AT91C_ID_TC0);             // enable Timer0 peripheral clock


    // Set up the LEDs (PA0 - PA3)
    volatile AT91PS_PIO   pPIO = AT91C_BASE_PIOA;   // pointer to PIO data structure
    pPIO->PIO_PER = LED_MASK | SW1_MASK;            // PIO Enable Register - allow PIO to control pins P0 - P3 and pin 19
    pPIO->PIO_OER = LED_MASK;                       // PIO Output Enable Register - sets pins P0 - P3 to outputs
    pPIO->PIO_SODR = LED_MASK;                      // PIO Set Output Data Register - turns off the four LEDs


    // Select PA19 (pushbutton) to be FIQ function (Peripheral B)
    pPIO->PIO_BSR = SW1_MASK;


    // Set up the AIC  registers for Timer 0
    volatile AT91PS_AIC   pAIC = AT91C_BASE_AIC;    // pointer to AIC data structure
    pAIC->AIC_IDCR = (1<<AT91C_ID_TC0);             // Disable timer 0 interrupt in AIC Interrupt Disable Command Register

    pAIC->AIC_SVR[AT91C_ID_TC0] =                   // Set the TC0 IRQ handler address in AIC Source
        (unsigned int)Timer0IrqHandler;             // Vector Register[12]
    pAIC->AIC_SMR[AT91C_ID_TC0] =                   // Set the interrupt source type and priority
        (AT91C_AIC_SRCTYPE_INT_HIGH_LEVEL | 0x4 );  // in AIC Source Mode Register[12]
    pAIC->AIC_ICCR = (1<<AT91C_ID_TC0);             // Clear the TC0 interrupt in AIC Interrupt Clear Command Register
    pAIC->AIC_IDCR = (0<<AT91C_ID_TC0);             // Remove disable timer 0 interrupt in AIC Interrupt Disable Command Reg

    pAIC->AIC_IECR = (1<<AT91C_ID_TC0);             // Enable the TC0 interrupt in AIC Interrupt Enable Command Register

    // Set up the AIC registers for FIQ (pushbutton SW1)
    pAIC->AIC_IDCR = (1<<AT91C_ID_FIQ);             // Disable FIQ interrupt in AIC Interrupt Disable Command Register
    pAIC->AIC_SMR[AT91C_ID_FIQ] =                   // Set the interrupt source type in AIC Source
        (AT91C_AIC_SRCTYPE_INT_POSITIVE_EDGE);      // Mode Register[0]
    pAIC->AIC_ICCR = (1<<AT91C_ID_FIQ);             // Clear the FIQ interrupt in AIC Interrupt Clear Command Register
    pAIC->AIC_IDCR = (0<<AT91C_ID_FIQ);             // Remove disable FIQ interrupt in AIC Interrupt Disable Command Register

    pAIC->AIC_IECR = (1<<AT91C_ID_FIQ);             // Enable the FIQ interrupt in AIC Interrupt Enable Command Register


    // Three functions from the libraries
    a = strlen(pText);                              // strlen( ) returns length of a string
    x5 = fabs(y5);                                  // fabs( ) returns absolute value of a double
    n = atol(DigitBuffer);                          // atol( ) converts string to a long
    // Setup timer0 to generate a 10 msec periodic interrupt
    TimerSetup();

    // enable interrupts
    enableIRQ();
    enableFIQ();


    // endless blink loop
    while (1) {
        if  ((pPIO->PIO_ODSR & LED1) == LED1)       // read previous state of LED1
            pPIO->PIO_CODR = LED1;                  // turn LED1 (DS1) on
        else
            pPIO->PIO_SODR = LED1;                  // turn LED1 (DS1) off

        for (j = 1000000; j != 0; j-- );            // wait 1 second 1000000

        IdleCount++;                                // count # of times through the idle loop
        pPIO->PIO_SODR = LED3;                      // turn LED3 (DS3) off

        // uncomment following four lines to cause a data abort(3 blink code)
        //if  (IdleCount >= 10) {                   // let it blink 5 times then crash
        //  p = (int *)0x800000;                    // this address doesn't exist
        //  *p = 1234;                              // attempt to write data to invalid address
        //}

        // uncomment following four lines to cause a prefetch abort (two blinks)
        //if  (IdleCount >= 10) {                   // let it blink 5 times then crash
        //  pFnPtr = (FnPtr)0x800000;               // this address doesn't exist
        //  pFnPtr();                               // attempt to call a function at a illegal address
        //}
    }
}
```

TIMERISR.C

The Timer0 interrupt service routine is called by the AT91F_Irq_Handler in the crt.s assembly language start-up module. The AT91F_Irq_Handler in the start-up routine supports "nested" IRQ interrupts and thus calls a standard C function do most of the interrupt work.

The C language IRQ support routine below clears the interrupt by reading the TCO status register. It then updates a global variable tickcount; which can be inspected by the debugger. Finally it toggles LED2 to give a visual indication that the timer interrupt is functioning properly.

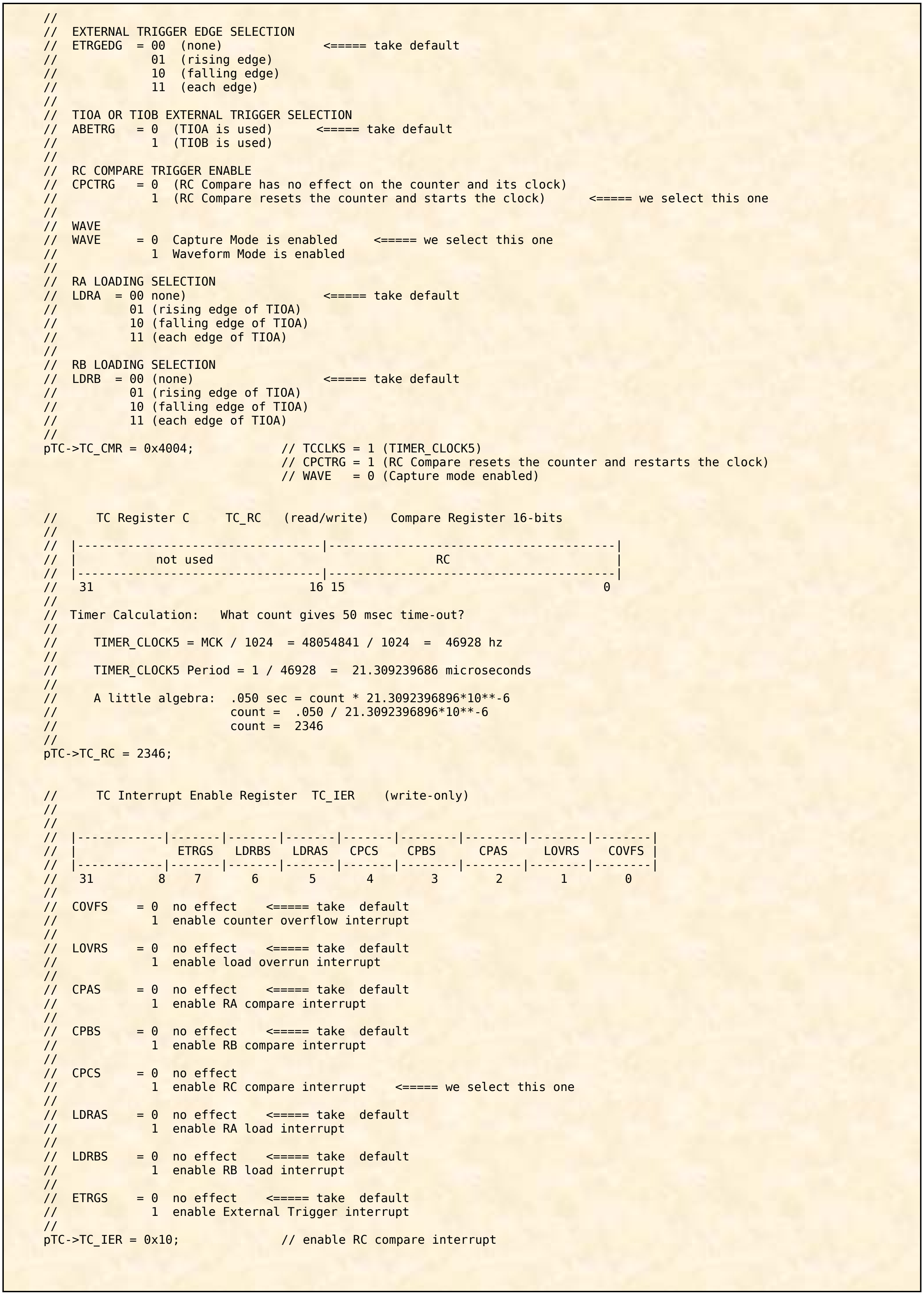
```
// *****************************************************************************
//                              timerisr.c
//
//    Timer 0 Interrupt Service Routine
//
//    entered when Timer0 RC compare interrupt asserts (200 msec period)
//    blinks LED2 (pin PA2)
//
//    Author:  James P Lynch     May 12, 2007
// *****************************************************************************

#include "AT91SAM7S256.h"
#include "board.h"

unsigned long    tickcount = 0;                          // global variable counts interrupts


void Timer0IrqHandler (void) {

    volatile AT91PS_TC      pTC = AT91C_BASE_TC0;        // pointer to timer channel 0 register structure
    volatile AT91PS_PIO     pPIO = AT91C_BASE_PIOA;      // pointer to PIO register structure
    unsigned int            dummy;                       // temporary

    dummy = pTC->TC_SR;                                  // read TC0 Status Register to clear interrupt
    tickcount++;                                         // increment the tick count

    if  ((pPIO->PIO_ODSR & LED2) == LED2)
        pPIO->PIO_CODR = LED2;                           // turn LED2 (DS2) on
    else
        pPIO->PIO_SODR = LED2;                           // turn LED2 (DS2) off
}
```

TIMERSETUP.C

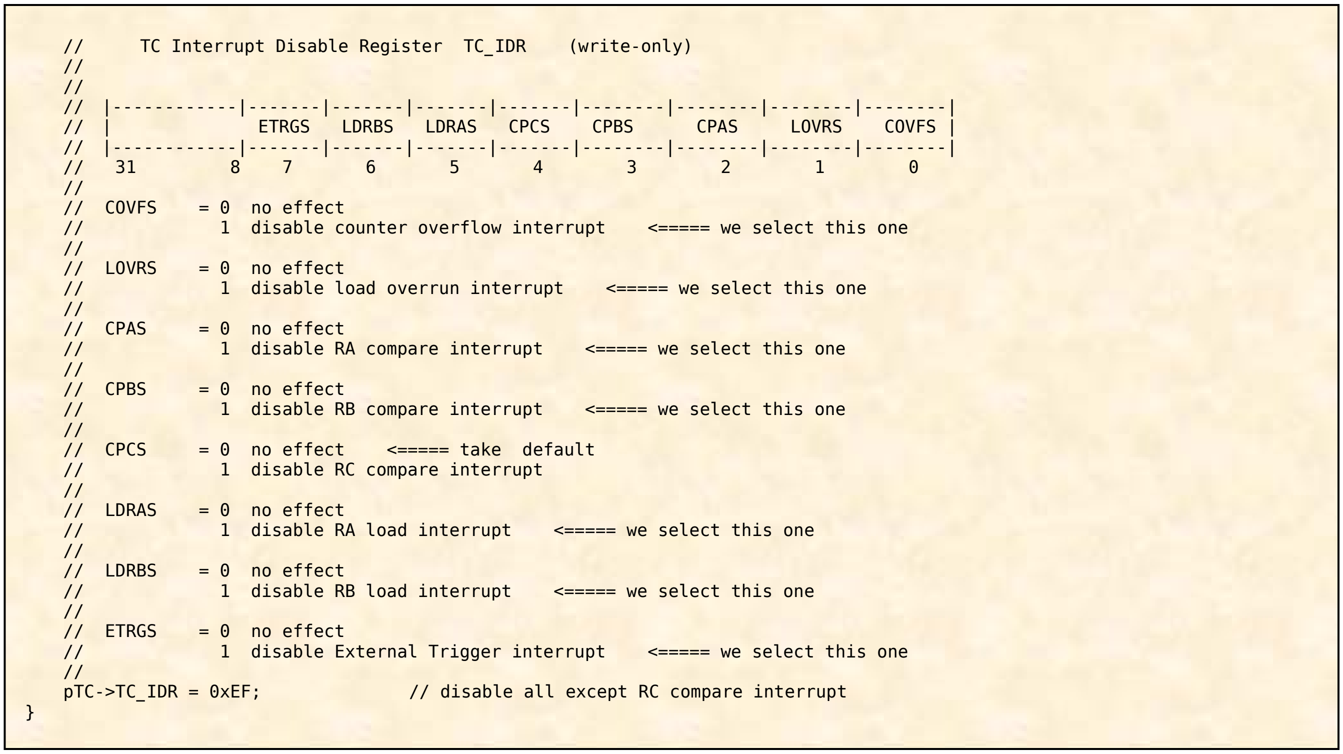
All the peripherals on the Atmel AT91SAM7S256 chip are complex; there is no substitute for a careful and thorough study of the Atmel documentation. In this application, we are using the Timer0 counter/timer to count out a 50 msec time interval. The timersetup.c routine shown below is extensively annotated to make it clear how the clock frequencies and count-match values were determined to get the 50 msec repetition rate. The timer counts up, comparing at each tick the current count with the value in the timer compare register C. When the values match, the IRQ interrupt is asserted. Timer 0 has been set up to automatically restart the timer beginning at zero for the next interval.

```
// *****************************************************************************
//                        timersetup.c
//
//     Purpose:  Set up the 16-bit Timer/Counter
//
//     We will use Timer Channel 0 to develop a 50 msec interrupt.
//
//     The AT91SAM7S-EK board has a 18,432,000 hz crystal oscillator.
//
//     MAINCK  = 18432000 hz
//     PLLCK = (MAINCK / DIV) * (MUL + 1) = 18432000/14 * (72 + 1)
//     PLLCLK = 1316571 * 73 = 96109683 hz
//
//    MCK = PLLCLK / 2 = 96109683 / 2 = 48054841 hz
//
//     TIMER_CLOCK5 = MCK / 1024  = 48054841 / 1024  =  46928 hz
//
//     TIMER_CLOCK5 Period = 1 / 46928  =  21.309239686 microseconds
//
//     A little algebra:  .050 sec = count * 21.3092396896*10**-6
//                        count =  .050 / 21.3092396896*10**-6
//                        count =  2346
//
//
//     Therefore:  set Timer Channel 0 register RC to 9835
//                 turn on capture mode WAVE = 0
//                 enable the clock  CLKEN = 1
//                 select TIMER_CLOCK5  TCCLKS = 100
//                 clock is NOT inverted CLKI = 0
//                 enable RC compare CPCTRG = 1
//                 enable RC compare interrupt  CPCS = 1
//                 disable all the other timer 0 interrupts
//
// Author:  James P Lynch  May 12, 2007
// *****************************************************************************


/**********************************************************
                  Header files
 **********************************************************/
#include "AT91SAM7S256.h"
#include "board.h"



void   TimerSetup(void) {

    //     TC Block Control Register TC_BCR    (read/write)
    //
    //  |------------------------------------------------------------------|------|
    //  |                                                                   SYNC  |
    //  |------------------------------------------------------------------|------|
    //   31                                                             1    0
    //
    //  SYNC = 0 (no effect)    <===== take  default
    //  SYNC = 1 (generate software trigger for all 3 timer channels simultaneously)
    //
    AT91PS_TCB pTCB = AT91C_BASE_TCB;    // create a pointer to TC Global Register structure
    pTCB->TCB_BCR = 0;                   // SYNC trigger not used
```

```
//     TC Block Mode Register  TC_BMR    (read/write)
//
//  |-------------------------------------|-----------|-----------|-----------|
//  |                                       TC2XC2S     TCXC1S       TC0XC0S  |
//  |-------------------------------------|-----------|-----------|-----------|
//   31                                     5       4   3       2   1         0
//
//  TC0XC0S Select = 00 TCLK0 (PA4)
//                 = 01 none           <===== we select this one
//                 = 10 TIOA1 (PA15)
//                 = 11 TIOA2 (PA26)
//
//  TCXC1S  Select = 00 TCLK1 (PA28)
//                 = 01 none           <===== we select this one
//                 = 10 TIOA0 (PA15)
//                 = 11 TIOA2 (PA26)
//
//  TC2XC2S Select = 00 TCLK2 (PA29)
//                 = 01 none           <===== we select this one
//                 = 10 TIOA0 (PA00)
//                 = 11 TIOA1 (PA26)
//
pTCB->TCB_BMR = 0x15;                // external clocks not used


//     TC Channel Control Register  TC_CCR    (read/write)
//
//  |----------------------------------|--------------|------------|------------|
//  |                                      SWTRG          CLKDIS       CLKENS   |
//  |----------------------------------|--------------|------------|------------|
//   31                                         2             1           0
//
//  CLKEN = 0    no effect
//  CLKEN = 1    enables the clock     <===== we select this one
//
//  CLKDIS = 0   no effect             <===== take  default
//  CLKDIS = 1   disables the clock
//
//  SWTRG = 0    no effect
//  SWTRG = 1    software trigger aserted counter reset and clock starts   <===== we select this one
//
AT91PS_TC pTC = AT91C_BASE_TC0;      // create a pointer to channel 0 Register structure
pTC->TC_CCR = 0x5;                   // enable the clock   and start it



//     TC Channel Mode Register  TC_CMR    (read/write)
//
//  |-----------------------------------|------------|---------------|
//  |                                      LDRB            LDRA      |
//  |-----------------------------------|------------|---------------|
//   31                                  19        18 17           16
//
//  |----------|---------|--------------|------------|---------------|
//  |WAVE = 0    CPCTRG                    ABETRG         ETRGEDG    |
//  |----------|---------|--------------|------------|---------------|
//      15         14     13         11      10      9             8
//
//  |----------|---------|--------------|------------|---------------|
//  |  LDBDIS    LDBSTOP     BURST         CLKI            TCCLKS    |
//  |----------|---------|--------------|------------|---------------|
//       7          6     5            4      3      2             0
//
//  CLOCK SELECTION
//  TCCLKS = 000  TIMER_CLOCK1    (MCK/2    =  24027420 hz)
//           001  TIMER_CLOCK2    (MCK/8    =   6006855 hz)
//           010  TIMER_CLOCK3    (MCK/32   =   1501713 hz)
//           011  TIMER_CLOCK4    (MCK/128  =    375428 hz)
//           100  TIMER_CLOCK5    (MCK/1024 =     46928 hz)   <===== we select this one
//           101  XC0
//           101  XC1
//           101  XC2
//
//  CLOCK INVERT
//  CLKI   = 0   counter incremented on rising clock edge    <===== we select this one
//  CLKI   = 1   counter incremented on falling clock edge
//
//  BURST SIGNAL SELECTION
//  BURST  = 00  clock is not gated by any external system   <===== take  default
//           01  XC0 is anded with the clock
//           10  XC1 is anded with the clock
//           11  XC2 is anded with the clock
//
//  COUNTER CLOCK STOPPED WITH RB LOADING
//  LDBSTOP  = 0  counter clock is not stopped when RB loading occurs   <===== take  default
//           = 1  counter clock is stopped when RB loading occur
//
//  COUNTER CLOCK DISABLE WITH RB LOADING
//  LDBDIS   = 0  counter clock is not disabled when RB loading occurs   <===== take  default
//           = 1  counter clock is disabled when RB loading occurs
```



```
//
//  EXTERNAL TRIGGER EDGE SELECTION
//  ETRGEDG  = 00  (none)              <===== take default
//             01  (rising edge)
//             10  (falling edge)
//             11  (each edge)
//
//  TIOA OR TIOB EXTERNAL TRIGGER SELECTION
//  ABETRG   = 0  (TIOA is used)      <===== take default
//             1  (TIOB is used)
//
//  RC COMPARE TRIGGER ENABLE
//  CPCTRG   = 0  (RC Compare has no effect on the counter and its clock)
//             1  (RC Compare resets the counter and starts the clock)      <===== we select this one
//
//  WAVE
//  WAVE     = 0  Capture Mode is enabled     <===== we select this one
//             1  Waveform Mode is enabled
//
//  RA LOADING SELECTION
//  LDRA  = 00 none)                   <===== take default
//          01 (rising edge of TIOA)
//          10 (falling edge of TIOA)
//          11 (each edge of TIOA)
//
//  RB LOADING SELECTION
//  LDRB  = 00 (none)                  <===== take default
//          01 (rising edge of TIOA)
//          10 (falling edge of TIOA)
//          11 (each edge of TIOA)
//
pTC->TC_CMR = 0x4004;             // TCCLKS = 1 (TIMER_CLOCK5)
                                  // CPCTRG = 1 (RC Compare resets the counter and restarts the clock)
                                  // WAVE   = 0 (Capture mode enabled)


//     TC Register C     TC_RC   (read/write)   Compare Register 16-bits
//
//  |---------------------------------|---------------------------------------|
//  |           not used              |                  RC                   |
//  |---------------------------------|---------------------------------------|
//   31                            16 15                                    0
//
//  Timer Calculation:   What count gives 50 msec time-out?
//
//     TIMER_CLOCK5 = MCK / 1024  = 48054841 / 1024  =  46928 hz
//
//     TIMER_CLOCK5 Period = 1 / 46928  =  21.309239686 microseconds
//
//     A little algebra:  .050 sec = count * 21.3092396896*10**-6
//                        count =  .050 / 21.3092396896*10**-6
//                        count =  2346
//
pTC->TC_RC = 2346;


//     TC Interrupt Enable Register  TC_IER    (write-only)
//
//
//  |------------|-------|-------|-------|-------|--------|--------|--------|--------|
//  |             ETRGS   LDRBS   LDRAS   CPCS    CPBS      CPAS     LOVRS    COVFS |
//  |------------|-------|-------|-------|-------|--------|--------|--------|--------|
//   31         8    7       6       5       4        3        2        1        0
//
//  COVFS    = 0  no effect    <===== take  default
//             1  enable counter overflow interrupt
//
//  LOVRS    = 0  no effect    <===== take  default
//             1  enable load overrun interrupt
//
//  CPAS     = 0  no effect    <===== take  default
//             1  enable RA compare interrupt
//
//  CPBS     = 0  no effect    <===== take  default
//             1  enable RB compare interrupt
//
//  CPCS     = 0  no effect
//             1  enable RC compare interrupt    <===== we select this one
//
//  LDRAS    = 0  no effect    <===== take  default
//             1  enable RA load interrupt
//
//  LDRBS    = 0  no effect    <===== take  default
//             1  enable RB load interrupt
//
//  ETRGS    = 0  no effect    <===== take  default
//             1  enable External Trigger interrupt
//
pTC->TC_IER = 0x10;              // enable RC compare interrupt
```



```
    //      TC Interrupt Disable Register  TC_IDR    (write-only)
    //
    //
    //  |------------|-------|-------|-------|-------|--------|--------|--------|--------|
    //  |              ETRGS   LDRBS   LDRAS   CPCS    CPBS      CPAS     LOVRS    COVFS |
    //  |------------|-------|-------|-------|-------|--------|--------|--------|--------|
    //   31         8    7       6       5       4        3        2        1        0
    //
    //  COVFS    = 0  no effect
    //             1  disable counter overflow interrupt    <===== we select this one
    //
    //  LOVRS    = 0  no effect
    //             1  disable load overrun interrupt    <===== we select this one
    //
    //  CPAS     = 0  no effect
    //             1  disable RA compare interrupt    <===== we select this one
    //
    //  CPBS     = 0  no effect
    //             1  disable RB compare interrupt    <===== we select this one
    //
    //  CPCS     = 0  no effect    <===== take  default
    //             1  disable RC compare interrupt
    //
    //  LDRAS    = 0  no effect
    //             1  disable RA load interrupt    <===== we select this one
    //
    //  LDRBS    = 0  no effect
    //             1  disable RB load interrupt    <===== we select this one
    //
    //  ETRGS    = 0  no effect
    //             1  disable External Trigger interrupt    <===== we select this one
    //
    pTC->TC_IDR = 0xEF;              // disable all except RC compare interrupt
}
```

DEMO_AT91SAM7_BLINK_FLASH.CMD

The Linker command script instructs the linker where to place the various parts of your program into FLASH and RAM.

The layout of memory and the subsequent specification of the TOS (top of stack) are critical. In the snippet below we specify 256K of FLASH starting at address 0x00000000 and 64K of RAM starting at address 0x00200000. Given the RAM starting at 0x00200000 and being 65536 bytes in length, the Top of Stack is placed 4 bytes from the end of RAM at 0x0020FFFC. The specification of the “top of stack” (_stack_end = 0x20FFFC) is used by the start-up routine, crt.s, to create the stacks for the various interrupt modes. The statements excerpted below are the ones that you would modify when moving to a different memory layout.

```
/* specify the AT91SAM7S256S  */
MEMORY
{
  flash   : ORIGIN = 0,                LENGTH = 256K /* FLASH EPROM   */
  ram     : ORIGIN = 0x00200000, LENGTH = 64K  /* static RAM area  */
}

/* define a global symbol _stack_end  (see analysis in annotation above) */
_stack_end = 0x20FFFC;
```

It’s a good idea to remind ourselves that the executable code (.text section) goes into FLASH memory and therefore the FLASH must be programmed before attempting execution. I can’t tell you how many times the author has built an application and forgotten to program the FLASH with the new code before starting the debugger.

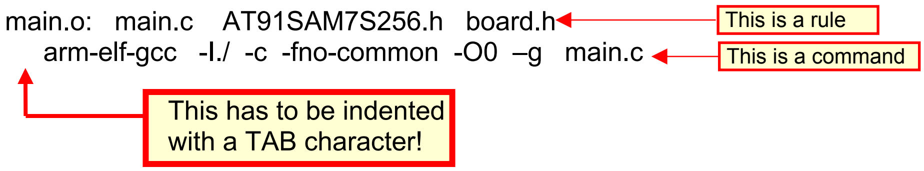
```
/* ****************************************************************************************************** */
/*   demo_at91sam7_blink_flash.cmd              LINKER  SCRIPT                                                */
/*                                                                                                            */
/*                                                                                                            */
/*   The Linker Script defines how the code and data emitted by the GNU C compiler and assembler are          */
/*   to be loaded into memory (code goes into FLASH, variables go into RAM).                                  */
/*                                                                                                            */
/*   Any symbols defined in the Linker Script are automatically global and available to the rest of the       */
/*   program.                                                                                                 */
/*                                                                                                            */
/*   To force the linker to use this LINKER SCRIPT, just add the -T demo_at91sam7_blink_flash.cmd             */
/*   directive to the linker flags in the makefile. For example,                                              */
/*                                                                                                            */
/*               LFLAGS  =  -Map main.map -nostartfiles -T demo_at91sam7_blink_flash.cmd                      */
/*                                                                                                            */
/*                                                                                                            */
/*   The order that the object files are listed in the makefile determines what .text section is              */
/*   placed first.                                                                                            */
/*                                                                                                            */
/*   For example:  $(LD) $(LFLAGS) -o main.out  crt.o main.o lowlevelinit.o                                   */
/*                                                                                                            */
/*                 crt.o is first in the list of objects, so it will be placed at address 0x00000000          */
/*                                                                                                            */
/*                                                                                                            */
/*   The top of the stack (_stack_end) is (last_byte_of_ram +1) - 4                                           */
/*                                                                                                            */
/*   Therefore:   _stack_end = (0x00020FFFF + 1) - 4  =  0x00021000 - 4  =  0x0020FFFC                        */
/*                                                                                                            */
/*   Note that this symbol (_stack_end) is automatically GLOBAL and will be used by the crt.s                 */
/*   startup assembler routine to specify all stacks for the various ARM modes                                */
/*                                                                                                            */
/*                         MEMORY MAP                                                                         */
/*                 |                                |                                                         */
/*          .----..->|--------------------------------|0x00210000                                             */
/*          .      |                                |0x0020FFFC  <---------- _stack_end                       */
/*          .      |    UDF Stack  16 bytes         |                                                         */
/*          .      |                                |                                                         */
/*          .      |--------------------------------|0x0020FFEC                                               */
/*          .      |                                |                                                         */
/*          .      |    ABT Stack  16 bytes         |                                                         */
/*          .      |                                |                                                         */
/*          .      |--------------------------------|0x0020FFDC                                               */
/*          .      |                                |                                                         */
/*          .      |                                |                                                         */
/*          .      |    FIQ Stack  128 bytes        |                                                         */
/*          .      |                                |                                                         */
/*          .      |                                |                                                         */
/*         RAM     |--------------------------------|0x0020FF5C                                               */
/*          .      |                                |                                                         */
/*          .      |                                |                                                         */
/*          .      |    IRQ Stack  128 bytes        |                                                         */
/*          .      |                                |                                                         */
/*          .      |                                |                                                         */
/*          .      |--------------------------------|0x0020FEDC                                               */
/*          .      |                                |                                                         */
/*          .      |    SVC Stack  16 bytes         |                                                         */
/*          .      |                                |                                                         */
/*          .      |--------------------------------|0x0020FECC                                               */
/*          .      |                                |                                                         */
/*          .      |        stack area for          |                                                         */
/*          .      |        user program            |                                                         */
/*          .      |                                |                                                         */
/*          .      |                                |                                                         */
/*          .      |          free ram              |                                                         */
/*          .      |                                |                                                         */
/*          .      |......................................|0x0020045C <---------- _bss_end                     */
/*          .      |              .bss              |                                                         */
/*          .      |  uninitialized variables       |                                                         */
/*          .      |......................................|0x00200444 <---------- _bss_start, _edata           */
/*          .      |              .data             |                                                         */
/*          .      |  initialized variables         |                                                         */
/*          .      |                                |                                                         */
/*          .------>|________________________________|0x00200000                                              */
/*                                                                                                            */
/*                                                                                                            */
/*          .-----> |---------------------------------|0x00040000                                           */
/*          .       |                                 |                                                     */
/*          .       |                                 |                                                     */
/*          .       |          free flash             |                                                     */
/*          .       |                                 |                                                     */
/*          .       |                                 |                                                     */
/*          .       |.................................|0x00001380 <---------- _bss_start, _edata             */
/*          .       |      Copy of .data area         |                                                     */
/*          .       |                                 |                                                     */
/*          .       |    (initialized variables)      |                                                     */
/*          .       |---------------------------------|0x00000F3C <----------- _etext                       */
/*          .       |                                 |                                                     */
/*       FLASH      |             C code              |                                                     */
/*          .       |                                 |                                                     */
/*          .       |                                 |                                                     */
/*          .       |---------------------------------|0x0000015C  ←---------- main()                       */
/*          .       |                                 |                                                     */
/*          .       |     Startup Code  (crt.s)       |                                                     */
/*          .       |          (assembler)            |                                                     */
/*          .       |                                 |                                                     */
/*          .       |---------------------------------|0x00000020                                           */
/*          .       |                                 |                                                     */
/*          .       | Interrupt Vector Table          |                                                     */
/*          .       |           32 bytes              |                                                     */
/*          .------>|---------------------------------|0x00000000 _vec_reset                                */
/*                                                                                                          */
/*                                                                                                          */
/*  Author:  James P. Lynch      May 12, 2007                                                               */
/*                                                                                                          */
/* ******************************************************************************************************** */


/* identify the Entry Point  (_vec_reset is defined in file crt.s)  */
ENTRY(_vec_reset)

/* specify the AT91SAM7S256 memory areas  */
MEMORY
{
    flash     : ORIGIN = 0, LENGTH = 256K                 /* FLASH EPROM      */
    ram       : ORIGIN = 0x00200000, LENGTH = 64K         /* static RAM area    */
}


/* define a global symbol _stack_end  (see analysis in annotation above) */
_stack_end = 0x20FFFC;

/* now define the output sections  */
SECTIONS
{
    . = 0;                                  /* set location counter to address zero  */

    .text :                                 /* collect all sections that should go into FLASH after startup  */
    {
        *(.text)                            /* all .text sections (code)  */
        *(.rodata)                          /* all .rodata sections (constants, strings, etc.)  */
        *(.rodata*)                         /* all .rodata* sections (constants, strings, etc.)  */
        *(.glue_7)                          /* all .glue_7 sections  (no idea what these are) */
        *(.glue_7t)                         /* all .glue_7t sections (no idea what these are) */
        _etext = .;                         /* define a global symbol _etext just after the last code byte */
    } >flash                                /* put all the above into FLASH */

    .data :                                 /* collect all initialized .data sections that go into RAM  */
    {
        _data = .;                          /* create a global symbol marking the start of the .data section  */
        *(.data)                            /* all .data sections  */
        _edata = .;                         /* define a global symbol marking the end of the .data section  */
    } >ram AT >flash                        /* put all the above into RAM (but load the LMA initializer copy into FLASH)  */

    .bss :                                  /* collect all uninitialized .bss sections that go into RAM  */
    {
        _bss_start = .;                     /* define a global symbol marking the start of the .bss section */
        *(.bss)                             /* all .bss sections  */
    } >ram                                  /* put all the above in RAM (it will be cleared in the startup code */

    . = ALIGN(4);                           /* advance location counter to the next 32-bit boundary */
    _bss_end = . ;                          /* define a global symbol marking the end of the .bss section */
}
    _end = .;                               /* define a global symbol marking the end of application RAM */
```

MAKEFILE

The makefile was kept intentionally simple so that a beginner need only read the first chapter of the “GNU Make” document by Richard Stallman and Roland McGrath to understand everything in the makefile.

The makefile is composed of two parts; the part that assembles, compiles and links your program to create a .bin file that you can load into flash, and a special “program” target that is used to independently program the FLASH on chip memory using the OpenOCD JTAG utility.

The essential component of a Makefile is the “rule”. The rule is composed of a target file and dependent files on a single line. If any of the dependent files are newer than the target file, then the commands directly below the rule are executed. The one or more commands MUST be indented by a TAB character (this little nuance beleaguers many novices). For example:



```
main.o:   main.c    AT91SAM7S256.h   board.h
    arm-elf-gcc  -I./  -c  -fno-common  -O0  –g   main.c
```

In the example rule above, if you edit either the main.c source file or the AT91SAM7S256.h or the board.h include files, they will then be “newer” than the main.o target file. Therefore, the commands below must be executed. The single compile command shown updates the target object file so that the target and the dependent files now have the same creation date.

The Make utility checks the rules from top to bottom and this has the effect of only compiling those source files that are “out of date”.

If you click the Eclipse “**Project - Clean**” pull-down menu option, the “clean” target below is performed first followed by the “all” target. This has the effect of recompiling everything since all the objects and binary files are erased first.

If you click the Eclipse “**Project – Build Project**” pull-down menu option, the “all” target is performed and only those source files that are out-of-date are recompiled. In a large application with many source files, this is a real convenience and time saver.

Note that the “clean” and “all” targets are NOT files. In this case, Make will only process them unless you specifically direct it to do so (make clean all or make all). This also explains why in scanning from top to bottom during a “make all”, make stops when it encounters the “program” target (used to program the FLASH). This is explained in more detail in a section to follow.

The ARM7 architecture supports two instruction sets, ARM and THUMB. The ARM instruction set is composed of 32-bit instructions and is very fast (most instructions execute in a single clock cycle). The THUMB instruction set is composed of 16-bit instructions that require less memory space but take longer to execute. To keep this tutorial simple, we’ve set up the project exclusively for the ARM 32-bit instruction set. If you would like to see a good example of mixing ARM and THUMB instruction sets in an ARM7 application, take a look at Richard Barry’s FreeRTOS kernel at www.freertos.com.

This make file is composed of two parts. The first part (identified as the targets “**clean:**” and “**all:**”) assembles, compiles and links your program. It creates a binary file suitable for programming into flash using the OpenOCD flash programming facility or the Atmel SAM-BA flash programming utility. It also produces a map file and a dump file that you can inspect to locate addresses of variables, storage limits and so forth.

The second part (identified as the target “**program:**”) does a batch execution of the OpenOCD JTAG utility to program the binary file into onchip flash. Note that the OpenOCD script file for programming the flash (script.ocd) is part of the project itself. The programming part of the makefile executes just once and OpenOCD is terminated when the flashing is complete. Obviously, the makefile assumes that OpenOCD is not running when it starts the programming operation.

If you are using the SAM-ICE debugger and plan to use the SAM-BA flash programming utility, then the flash programming part of the makefile shown below can be removed if desired.

```
# ***************************************************************
# *     Makefile for Atmel AT91SAM7S256 - flash execution       *
# *                                                             *
# *                                                             *
# *   James P Lynch  May 12, 2007                               *
# ***************************************************************

NAME = demo_at91sam7_blink_flash

# variables
CC      = arm-elf-gcc
LD      = arm-elf-ld -v
AR      = arm-elf-ar
AS      = arm-elf-as
CP      = arm-elf-objcopy
OD     = arm-elf-objdump

CFLAGS  = -I./ -c -fno-common -O0 -g
AFLAGS  = -ahls -mapcs-32 -o crt.o
LFLAGS  =  -Map main.map -Tdemo_at91sam7_blink_flash.cmd
CPFLAGS = --output-target=binary
ODFLAGS   = -x --syms

OBJECTS = crt.o  main.o timerisr.o timersetup.o isrsupport.o lowlevelinit.o blinker.o


# make target called by Eclipse (Project -> Clean ...)
clean:
    -rm $(OBJECTS) crt.lst main.lst main.out main.bin main.hex main.map main.dmp


#make target called by Eclipse  (Project -> Build Project)
all:  main.out
    @ echo "...copying"
    $(CP) $(CPFLAGS) main.out main.bin
    $(OD) $(ODFLAGS) main.out > main.dmp

main.out: $(OBJECTS) demo_at91sam7_blink_flash.cmd
    @ echo "..linking"
    $(LD) $(LFLAGS) -o main.out $(OBJECTS) libc.a libm.a libgcc.a

crt.o: crt.s
    @ echo ".assembling"
    $(AS) $(AFLAGS) crt.s > crt.lst

main.o: main.c
    @ echo ".compiling"
    $(CC) $(CFLAGS) main.c

timerisr.o: timerisr.c
    @ echo ".compiling"
    $(CC) $(CFLAGS) timerisr.c

lowlevelinit.o: lowlevelinit.c
    @ echo ".compiling"
    $(CC) $(CFLAGS) lowlevelinit.c

timersetup.o: timersetup.c
    @ echo ".compiling"
    $(CC) $(CFLAGS) timersetup.c

isrsupport.o: isrsupport.c
    @ echo ".compiling"
    $(CC) $(CFLAGS) isrsupport.c

blinker.o: blinker.c
    @ echo ".compiling"
    $(CC) $(CFLAGS) blinker.c
```

```
# ***********************************************************************************************
#                            FLASH PROGRAMMING
#
# Alternate make target for flash programming only
#
# You must create a special Eclipse make target (program) to run this part of the makefile
# (Project -> Create Make Target...  then set the Target Name and Make Target to "program")
#
# OpenOCD is run in "batch" mode with a special configuration file and a script file containing
# the flash commands. When flash programming completes, OpenOCD terminates.
#
# Note that the script file of flash commands (script.ocd) is part of the project
#
# Programmers: Martin Thomas, Joseph M Dupre, James P Lynch
# ***********************************************************************************************

# specify output filename here (must be *.bin file)
TARGET = main.bin

# specify the directory where openocd executable resides (openocd-ftd2xx.exe or openocd-pp.exe)
OPENOCD_DIR = 'c:\Program Files\openocd-2007re141\bin\'

# specify OpenOCD executable (pp is for the wiggler, ftd2xx is for the USB debuggers)
#OPENOCD = $(OPENOCD_DIR)openocd-pp.exe
OPENOCD = $(OPENOCD_DIR)openocd-ftd2xx.exe

# specify OpenOCD configuration file (pick the one for your device)
#OPENOCD_CFG = $(OPENOCD_DIR)at91sam7s256-wiggler-flash-program.cfg
#OPENOCD_CFG = $(OPENOCD_DIR)at91sam7s256-jtagkey-flash-program.cfg
OPENOCD_CFG = $(OPENOCD_DIR)at91sam7s256-armusbocd-flash-program.cfg

# program the AT91SAM7S256 internal flash memory
program: $(TARGET)
    @echo "Flash Programming with OpenOCD..."          # display a message on the console
    $(OPENOCD) -f $(OPENOCD_CFG)                       # program the onchip FLASH here
    @echo "Flash Programming Finished."                # display a message on the console
```

OpenOCD Programming Script File

OpenOCD normally runs as a “daemon” processing debugger commands when required. To program the onchip FLASH, OpenOCD is run as a one-time-only execution with a list of programming commands read from a script file named **script.ocd**. This file is part of the project. Note that it contains register setups to reset the processor and establish the PLL clocks to full speed. This is necessary to program the FLASH at full speed. Review the source code for lowlevelinit.c to understand how the register settings were derived.

```
#   OpenOCD Target Script for Atmel AT91SAM7S256
#
#   Programmer: James P Lynch
#

wait_halt                      # halt the processor and wait
armv4_5 core_state arm         # select the core state
mww 0xffffff60 0x00320100      # set flash wait state (AT91C_MC_FMR)
mww 0xfffffd44 0xa0008000      # watchdog disable (AT91C_WDTC_WDMR)
mww 0xfffffc20 0xa0000601      # enable main oscillator (AT91C_PMC_MOR)
wait 100                       # wait 100 ms
mww 0xfffffc2c 0x00480a0e      # set PLL register (AT91C_PMC_PLLR)
wait 200                       # wait 200 ms
mww 0xfffffc30 0x7             # set master clock to PLL (AT91C_PMC_MCKR)
wait 100                       # wait 100 ms
mww 0xfffffd08 0xa5000401      # enable user reset AT91C_RSTC_RMR
flash write 0 main.bin 0x0     # program the onchip flash
reset                          # reset processor
shutdown                       # stop OpenOCD
```

Adjusting the Optimization Level

It's a fact of life in embedded programming that debuggers hate optimized code. When you attempt to single-step optimized code, the debuggers will do strange things and appear not to work. To get around this problem, change the compiler optimization level to ZERO. This is already done in the makefile above; note that we modified the CFLAGS macro substitution as follows:

CFLAGS = -I./ -c -fno-common **-O0** –g Where the switch: -O0 means no optimization.

When debugging is completed, you can increase the optimization level to –O3 which will result in more compact and efficient code.

Including Libraries

A library is a collection of already-compiled functions. The GNU linker will search the libraries you specify for any functions you have invoked in the application and only include those functions in the final link (it doesn't include the entire library – just the functions you need). Specifying the libraries and arranging for successful searching in the linker command is a constant source of trouble for the novice programmer as the GNU linker manual can be, well, a little confusing on this subject.

There are three libraries included in YAGARTO that you should be aware of.

libgcc.a	**ARM-specific library supporting floating point and extended arithmetic (must be included)**
libc.a	**Newlib C Library – has functions like strlen( ), isdigit( ) etc. (optional)**
libm.a	**Newlib Math Library – has functions like exp( ), sin( ) etc. (optional)**

Adding Libraries to the Link

There is a foolproof way of dealing with libraries: import the libraries directly into your project and include the libraries on the linker command line after specification of all the object files. For example, the libraries **libc.a, libm.a** and **libgcc.a** are imported into the project folder and are specified in the linker command line as follows:

```
$(LD) $(LFLAGS) -o main.out $(OBJECTS)  libc.a  libm.a  libgcc.a
```

Expanding the macro substitutions above and splitting the linker command line into two lines for the sake of clarity, the linker command actually looks like this:

```
arm-elf-ld -v -Map main.map -Tdemo_at91sam7_blink_flash.cmd -o main.out crt.o main.o timerisr.o
timersetup.o isrsupport.o lowlevelinit.o blinker.o  libc.a  libm.a  libgcc.a
```

Include your libraries after specification of all the object files!

Note: **libgcc.a** should always be last

The libgcc.a must be included; it supports ARM extended precision arithmetic and floating point operations (remember that the ARM7 doesn't have hardware floating point) and without it any floating point operations will cause an "undefined reference" error. The example project includes the libgcc.a library and it should always be appended to the very end of the linker command line, as shown earlier.

The reason for placing the libraries last on the linker command line is that the GNU linker searches from left to right. Any unresolved function calls after searching all the object files you have specified will resort to searching the remaining libraries on the right. The library may not be searched at all for any unresolved references if the library is specified before the object files (or in the middle of them).

For instance, the function atol( ) in the libc.a library will do some extended precision arithmetic and will therefore need some of the support routines in the libgcc.a library. Since the extended precision arithmetic support library libgcc.a is on the far right, the linker will successfully resolve the needed support routines. If libgcc.a was specified before (to the left of) libc.a, then an "undefined reference" error will result.

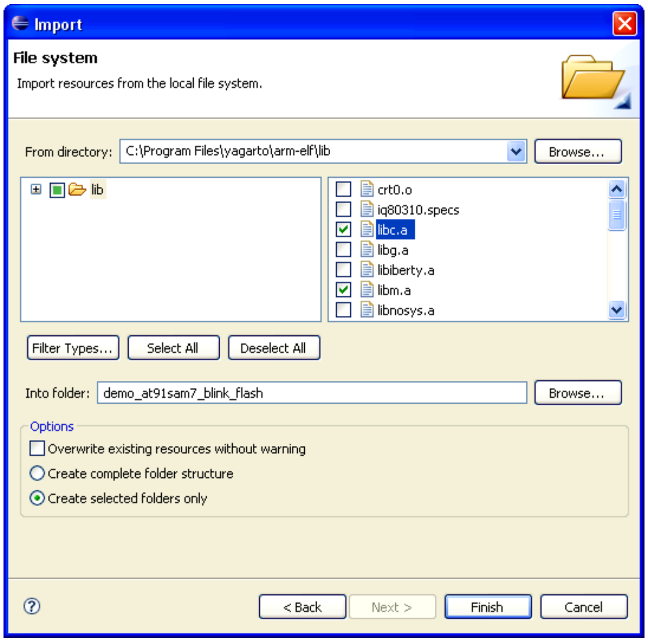
You might be tempted to say "Why should I have a copy of the library in every project and waste disk space?" The idea is to prevent the GNU Linker from hunting for your library. Having the library right in your project folder and specified last on the linker command line is fool-proof. Anyway, disk space on modern PCs is huge – relax!

Where are the Libraries

Michael Fisher has built ARM-compatible versions of the standard GNU libraries as part of YAGARTO.
The libraries used in the sample project may be found here:

Library	Path to library location
libgcc.a	C:\Program Files\yagarto\arm-elf\lib\
libc.a **libm.a**	C:\Program Files\yagarto\lib\gcc\arm-elf\4.1.1\

Just like a source or include file, you import the libraries into the project. Click on "**File – Import – File System**" followed by "**Next**". In the example below, we are importing the libc.a and libm.a libraries. The sample project already includes the three aforementioned libraries and you have thus already imported them. The screenshot below is included just to remind you that libraries are "imported" also.



Import

File system

Import resources from the local file system.

From directory: C:\Program Files\yagarto\arm-elf\lib

☐ crt0.o
☐ iq80310.specs
☑ libc.a
☐ libg.a
☐ libiberty.a
☑ libm.a
☐ libnosys.a

Into folder: demo_at91sam7_blink_flash

Options
☐ Overwrite existing resources without warning
○ Create complete folder structure
● Create selected folders only

Display the Modules in a Library

If you are attempting to sort out an “unresolved reference” problem concerning the libraries, here is a convenient way to look at the object module names in a library.

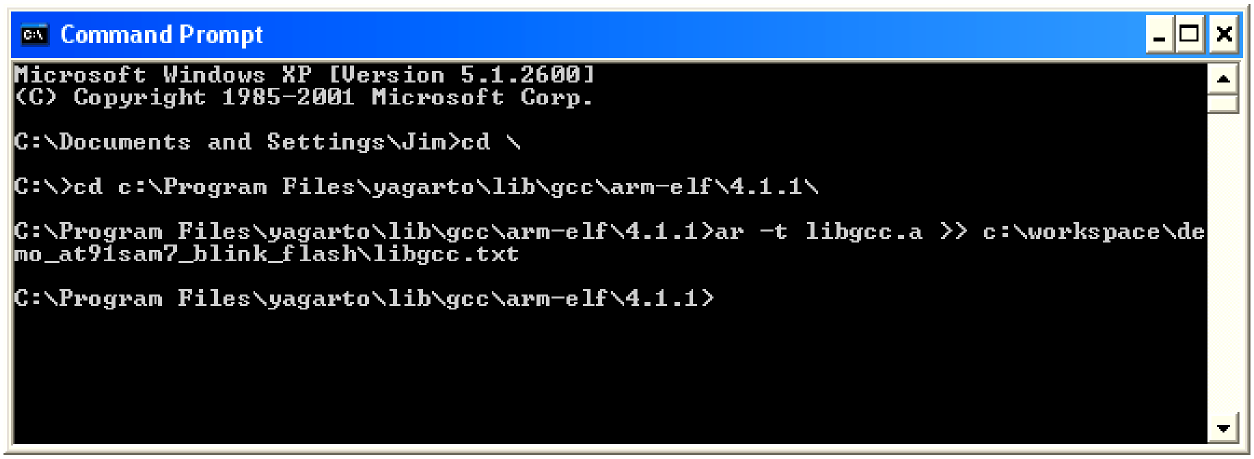
Open a command window and then navigate to the folder where the library resides (see the table above).

As an example, the following command will navigate to the folder where libgcc.a resides.

```
>cd c:\Program Files\yagarto\lib\gcc\arm-elf\4.1.1\
```

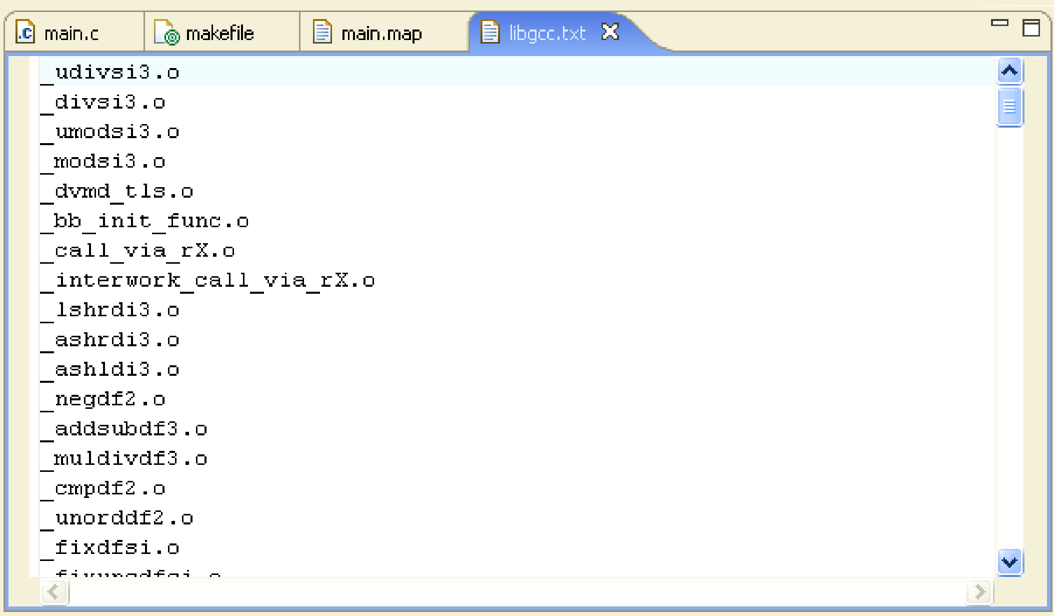
Use the GNU utility AR to display the object modules contained in the libgcc.a library. In the example below, we run the AR utility and send it’s output to the temporary file libgcc.txt in our project workspace (if this file doesn’t exist, it will create it). Here’s the command to do this.

```
>ar –t  libgcc.a  >>  c:\workspace\demo_at91sam7_blink_flash\libgcc.txt
```



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Jim>cd \

C:\>cd c:\Program Files\yagarto\lib\gcc\arm-elf\4.1.1\

C:\Program Files\yagarto\lib\gcc\arm-elf\4.1.1>ar -t libgcc.a >> c:\workspace\de
mo_at91sam7_blink_flash\libgcc.txt

C:\Program Files\yagarto\lib\gcc\arm-elf\4.1.1>
```

Now from within your Eclipse project, you can use “**File – Open File …**” to inspect the temporary text file containing the object module names from the library file libgcc.a. This file is temporary and is not part of your project. You can call up the “right-click” menu to delete it when finished.



```
_udivsi3.o
_divsi3.o
_umodsi3.o
_modsi3.o
_dvmd_tls.o
_bb_init_func.o
_call_via_rX.o
_interwork_call_via_rX.o
_lshrdi3.o
_ashrdi3.o
_ashldi3.o
_negdf2.o
_addsubdf3.o
_muldivdf3.o
_cmpdf2.o
_unorddf2.o
_fixdfsi.o
```

The Bad News about Libraries

Dealing with libraries in an embedded software development environment is fraught with difficulties that test one's patience. Not all the library modules you want to use will work.

For example, many programmers love the printf( ) routine for its convenience and formatting capabilities. Keep in mind that these GNU library routines were written for PC-based LINUX and Windows systems where memory storage is not an issue. Also, we would typically print to the Standard Output (the screen).

In an embedded environment, there is no Standard Output or screen to write to. So where does this printf( ) output go? Do we output to the serial port and, if so, which one? Now we need a putc( ) and getc( ) routine and interrupt support. You will also be shocked to see the compiled size of a routine like printf( ), it may be over 30 Kbytes due to the sophisticated formatting capabilities included.

If you select a library module and use it in your application and it builds with an "undefined reference" link error, chances are that some needed software elements are missing. You can try looking for them in some of the other libraries included in YAGARTO but in many of these cases the search will be frustrating.

The truth is that NewLib (libc.a and libm.a) tend to be too big and incomplete for an embedded environment. It's better to find a small library intended for embedded work and use bits and pieces of that as needed. One good example is Pascal Stang's ARMLIB.

http://hubbard.engr.scu.edu/embedded/arm/armlib/index.html

SourceForge is another good place to look for embedded libraries for the ARM architecture. The SourceForge web site is here:

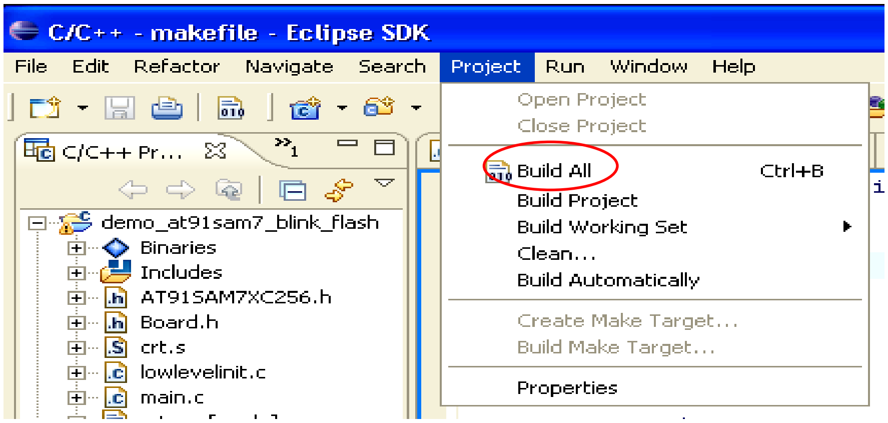
http://sourceforge.net/

To be fair, the more expensive professional tool chains usually have special copies of the libraries designed and compiled specifically for the embedded environment.

Building the FLASH Application

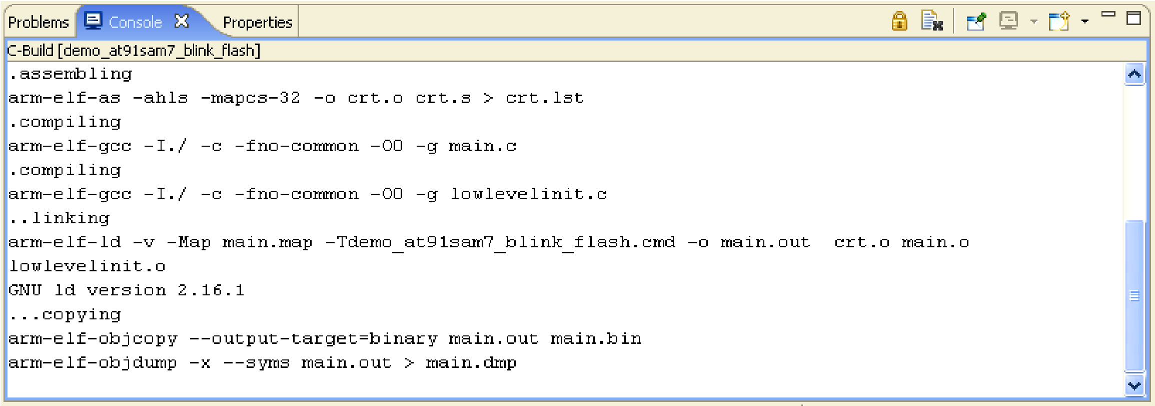
The “**Project**” pull-down menu has several options to build the application. “**Clean**” will erase all object, list, map, and output and dump files, thus forcing Eclipse to compile, assemble and link every file. This may be time-consuming in a large project with many files. “**Build All**” will only compile and link those files that are “out-of-date”.

The usual procedure is to “**Build All**” and this may be selected from the “**Projects**” pull-down menu, as shown below.



Even more convenient is the “**Build All**” button in the Eclipse toolbar.

The Console view at the bottom of the Eclipse screen will show the progress of the build operation.



```
.assembling
arm-elf-as -ahls -mapcs-32 -o crt.o crt.s > crt.lst
.compiling
arm-elf-gcc -I./ -c -fno-common -O0 -g main.c
.compiling
arm-elf-gcc -I./ -c -fno-common -O0 -g lowlevelinit.c
..linking
arm-elf-ld -v -Map main.map -Tdemo_at91sam7_blink_flash.cmd -o main.out  crt.o main.o
lowlevelinit.o
GNU ld version 2.16.1
...copying
arm-elf-objcopy --output-target=binary main.out main.bin
arm-elf-objdump -x --syms main.out > main.dmp
```

Notice that the “objcopy” utility has created a “**main.bin**” file; this is required by the OpenOCD flash programming facility. The makefile also creates a “**main.out**” file that has symbol information; this is used in debugging and also is “loaded” into RAM when you create a RAM-based executable.

If there are compile or link errors, they will be visible in the Console view and the “**Problems**” tab will show more detail about any problems. You can click on the individual “problems” and jump directly to the offending source line.

Using OpenOCD to Program the FLASH memory

If you have purchased the Olimex ARM-USB-OCD or the Amontec JTAGKey JTAG hardware interface, you can use the OpenOCD utility to program the flash.

OpenOCD is a utility that converts Eclipse/GDB remote serial protocol to the JTAG protocol supported by the AT91SAM7 on chip debugging unit. In this role, it acts as a “daemon” which is a program that operates in the background, waiting for you to supply a command. We will see plenty of examples of that when we run the debugger shortly.

The other role for OpenOCD is to program the on chip FLASH using the JTAG. In this role, OpenOCD is run in a “batch” mode where the program is executed with a special configuration file and a “script” file with the flash programming commands.

The OpenOCD configuration files to support flash programming on an Atmel AT91SAM7S are as follows. If you are interested in understanding every nuance of these files, refer to the OpenOCD Wiki here:

http://openfacts.berlios.de/index-en.phtml?title=Open_On-Chip_Debugger

It’s worth mentioning that the non-flash-programming versions of these configuration files are simply the part that’s above the FLASH programming commands. When FLASH programming is completed, OpenOCD is automatically terminated.

OpenOCD Configuration File for Wiggler (FLASH programming version)

```
#define our ports
telnet_port 4444
gdb_port 3333

#commands specific to the Wiggler
interface parport
parport_port 0x378
parport_cable wiggler
jtag_speed 0
jtag_nsrst_delay 200
jtag_ntrst_delay 200

#reset_config <signals> [combination] [trst_type] [srst_type]
reset_config srst_only srst_pulls_trst

#jtag_device <IR length> <IR capture> <IR mask> <IDCODE instruction>
jtag_device 4 0x1 0xf 0xe

#daemon_startup <'attach'|'reset'>
daemon_startup reset

#target <type> <endianess> <reset_mode> <jtag#> [variant]
target arm7tdmi little run_and_init 0 arm7tdmi_r4

#run_and_halt_time <target#> <time_in_ms>
run_and_halt_time 0 30

# commands below are specific to AT91sam7 Flash Programming
# ---------------------------------------------------------

#target_script specifies the flash programming script file
target_script 0 reset script.ocd

#working_area <target#> <address> <size> <'backup'|'nobackup'>
working_area 0 0x40000000 0x4000 nobackup

#flash bank at91sam7 0 0 0 0 <target#>
flash bank at91sam7 0 0 0 0 0
```

OpenOCD Configuration File for JTAGKey (FLASH programming version)

```
#define our ports
telnet_port 4444
gdb_port 3333

#commands specific to the Amontec JTAGKey
interface ft2232
ft2232_device_desc "Amontec JTAGkey A"
ft2232_layout jtagkey
ft2232_vid_pid 0x0403 0xcff8
jtag_speed 2
jtag_nsrst_delay 200
jtag_ntrst_delay 200

#reset_config <signals> [combination] [trst_type] [srst_type]
reset_config srst_only srst_pulls_trst

#jtag_device <IR length> <IR capture> <IR mask> <IDCODE instruction>
jtag_device 4 0x1 0xf 0xe

#daemon_startup <'attach'|'reset'>
daemon_startup reset

#target <type> <endianess> <reset_mode> <jtag#> [variant]
target arm7tdmi little run_and_init 0 arm7tdmi_r4

#run_and_halt_time <target#> <time_in_ms>
run_and_halt_time 0 30

# commands below are specific to AT91sam7 Flash Programming
# ---------------------------------------------------------

#target_script specifies the flash programming script file
target_script 0 reset script.ocd

#working_area <target#> <address> <size> <'backup'|'nobackup'>
working_area 0 0x40000000 0x4000 nobackup

#flash bank at91sam7 0 0 0 0 <target#>
flash bank at91sam7 0 0 0 0 0
```

OpenOCD Configuration File for ARMUSBOCD (FLASH programming version)

```
#define our ports
telnet_port 4444
gdb_port 3333

#commands specific to the Olimex ARM-USB-OCD
interface ft2232
ft2232_device_desc "Olimex OpenOCD JTAG A"
ft2232_layout "olimex-jtag"
ft2232_vid_pid 0x15BA 0x0003
jtag_speed 2
jtag_nsrst_delay 200
jtag_ntrst_delay 200

#reset_config <signals> [combination] [trst_type] [srst_type]
reset_config srst_only srst_pulls_trst

#jtag_device <IR length> <IR capture> <IR mask> <IDCODE instruction>
jtag_device 4 0x1 0xf 0xe

#daemon_startup <'attach'|'reset'>
daemon_startup reset
```

```
#target <type> <endianess> <reset_mode> <jtag#> [variant]
target arm7tdmi little run_and_init 0 arm7tdmi_r4

#run_and_halt_time <target#> <time_in_ms>
run_and_halt_time 0 30

# commands below are specific to AT91sam7 Flash Programming
# ----------------------------------------------------------

#target_script specifies the flash programming script file
target_script 0 reset script.ocd

#working_area <target#> <address> <size> <'backup'|'nobackup'>
working_area 0 0x40000000 0x4000 nobackup

#flash bank at91sam7 0 0 0 0 <target#>
flash bank at91sam7 0 0 0 0 0
```

Note that all three of the configuration files (for FLASH programming) have the following command line:

target_script 0 reset script.ocd

This is directing OpenOCD to execute the script file “**script.ocd**” which has the flash programming commands. The file “script.ocd” is normally included in your project and typically has the following contents as shown below.

SCRIPT.OCD (normal version)

```
#  OpenOCD Target Script for Atmel AT91SAM7S256
#
#  Programmer: James P Lynch
#

wait_halt                                 # halt the processor and wait
armv4_5 core_state arm                    # select the core state
mww 0xffffff60 0x00320100                 # set flash wait state (AT91C_MC_FMR)
mww 0xfffffd44 0xa0008000                 # watchdog disable (AT91C_WDTC_WDMR)
mww 0xfffffc20 0xa0000601                 # enable main oscillator (AT91C_PMC_MOR)
wait 100                                  # wait 100 ms
mww 0xfffffc2c 0x00480a0e                 # set PLL register (AT91C_PMC_PLLR)
wait 200                                  # wait 200 ms
mww 0xfffffc30 0x7                        # set master clock to PLL (AT91C_PMC_MCKR)
wait 100                                  # wait 100 ms
mww 0xfffffd08 0xa5000401                 # enable user reset AT91C_RSTC_RMR
flash write 0 main.bin 0x0                # program the onchip flash
reset                                     # reset processor
shutdown                                  # stop OpenOCD
```

If the flash programming doesn’t work, it may well be that you have accidentally set the “lock” bits on the bottom two pages of flash. You can easily do this by powering up the board with the TST jumper installed; this installs a USB support program in your FLASH memory to enable the board to communicate with the SAM-BA flash programming utility.

In this case, you could add two additional commands to clear the lock bits. Be forewarned that the lock bits can only be set or cleared 100 times, so don’t leave these two commands in the script file.

SCRIPT.OCD (to remove lock bits)

```
#  OpenOCD Target Script for Atmel AT91SAM7S256
#
#  Programmer: James P Lynch
#

wait_halt                                 # halt the processor and wait
armv4_5 core_state arm                    # select the core state

mww 0xffffff64 0x5a000004                 # clear lock bit 0

mww 0xffffff64 0x5a002004                 # clear lock bit 1


mww 0xffffff60 0x00320100                 # set flash wait state (AT91C_MC_FMR)
mww 0xfffffd44 0xa0008000                 # watchdog disable (AT91C_WDTC_WDMR)
mww 0xfffffc20 0xa0000601                 # enable main oscillator (AT91C_PMC_MOR)
wait 100                                  # wait 100 ms
mww 0xfffffc2c 0x00480a0e                 # set PLL register (AT91C_PMC_PLLR)
wait 200                                  # wait 200 ms
mww 0xfffffc30 0x7                        # set master clock to PLL (AT91C_PMC_MCKR)
wait 100                                  # wait 100 ms
mww 0xfffffd08 0xa5000401                 # enable user reset AT91C_RSTC_RMR
flash write 0 main.bin 0x0                # program the onchip flash
reset                                     # reset processor
shutdown                                  # stop OpenOCD
```

Add these two commands if you think the flash memory lock bits are set.

Martin Thomas, guru of the WinARM tool chain, suggested that the flash programming using OpenOCD could be integrated into the makefile as an additional target.

Let’s review again the part of the makefile that programs the flash.

```
# ***********************************************************************************************
#                            FLASH PROGRAMMING
#
# Alternate make target for flash programming only
#
# You must create a special Eclipse make target (program) to run this part of the makefile
# (Project -> Create Make Target...  then set the Target Name and Make Target to "program")
#
# OpenOCD is run in "batch" mode with a special configuration file and a script file containing
# the flash commands. When flash programming completes, OpenOCD terminates.
#
# Note that the script file of flash commands (script.ocd) is part of the project
#
# Programmers: Martin Thomas, Joseph M Dupre, James P Lynch
# ***********************************************************************************************

# specify output filename here (must be *.bin file)
TARGET = main.bin

# specify the directory where openocd executable resides (openocd-ftd2xx.exe or openocd-pp.exe)
OPENOCD_DIR = 'c:\Program Files\openocd-2007re141\bin\'

# specify OpenOCD executable (pp is for the wiggler, ftd2xx is for the USB debuggers)
#OPENOCD = $(OPENOCD_DIR)openocd-pp.exe
OPENOCD = $(OPENOCD_DIR)openocd-ftd2xx.exe

# specify OpenOCD configuration file (pick the one for your device)
#OPENOCD_CFG = $(OPENOCD_DIR)at91sam7s256-wiggler-flash-program.cfg
#OPENOCD_CFG = $(OPENOCD_DIR)at91sam7s256-jtagkey-flash-program.cfg
OPENOCD_CFG = $(OPENOCD_DIR)at91sam7s256-armusbocd-flash-program.cfg

# program the AT91SAM7S256 internal flash memory
program: $(TARGET)
	@echo "Flash Programming with OpenOCD..."          # display a message on the console
	$(OPENOCD) -f $(OPENOCD_CFG)                       # program the onchip FLASH here
	@echo "Flash Programming Finished."                # display a message on the console
```

There are three places in the above makefile excerpt that you must customize.

First, you must correctly specify the folder name where the OpenOCD executable and the configuration files reside as this can change if a newer version of YAGARTO is downloaded.

```
# specify the directory where openocd executable resides (openocd-ftd2xx.exe or openocd-pp.exe)
# Note: you may have to adjust this if a newer version of YAGARTO has been downloaded
OPENOCD_DIR = 'c:\Program Files\openocd-2007re141\bin\'
```

Second, you must choose which version of OpenOCD you are running (wiggler or USB).

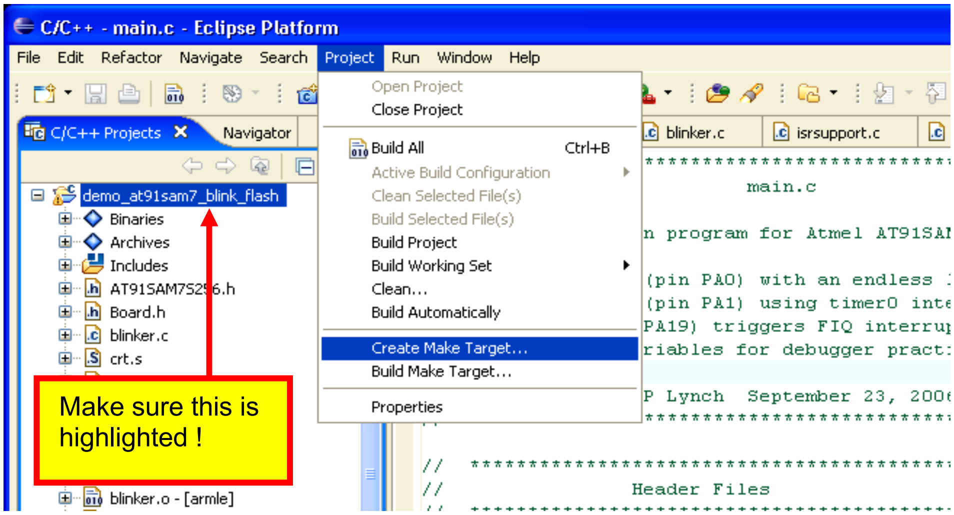
```
# specify OpenOCD executable (pp is for the wiggler, ftd2xx is for the USB debugger)
#OPENOCD = $(OPENOCD_DIR)openocd-pp.exe
OPENOCD = $(OPENOCD_DIR)openocd-ftd2xx.exe
```

Finally, you must choose which OpenOCD configuration file you will be using (wiggler, JTAGKey or ARMUSBOCD).

```
# specify OpenOCD configuration file (pick the one for your device)
#OPENOCD_CFG = $(OPENOCD_DIR)at91sam7s256-wiggler-flash-program.cfg
#OPENOCD_CFG = $(OPENOCD_DIR)at91sam7s256-jtagkey-flash-program.cfg
OPENOCD_CFG = $(OPENOCD_DIR)at91sam7s256-armusbocd-flash-program.cfg
```

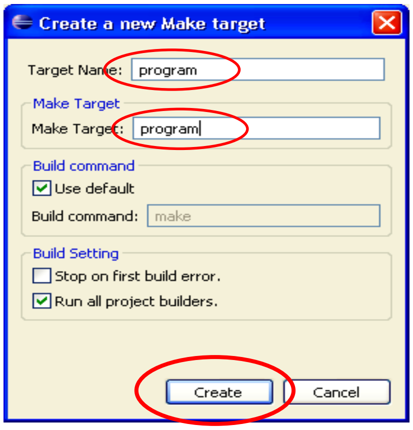
Assuming that you have already performed a "Build All" on the sample program and have an output file (main.bin) to program into the FLASH plus you have set up the hardware as shown earlier, you can now program the FLASH by running the "**program**" target in the makefile.

To prepare to do this, we need to establish "**program**" as a secondary make target. Click "**Project – Create Make Target…**" as shown below. Note that you must have the project itself highlighted in the "**Projects**" view to enable this.



Make sure this is highlighted !

In the “Create a New Make Target” dialog, enter “**program**” into the Target Name text box. Enter “**program**” into the Make Target text box also. Click “**Create**” to finish.



Target Name: program

Make Target

Make Target: program

Build command

☑ Use default

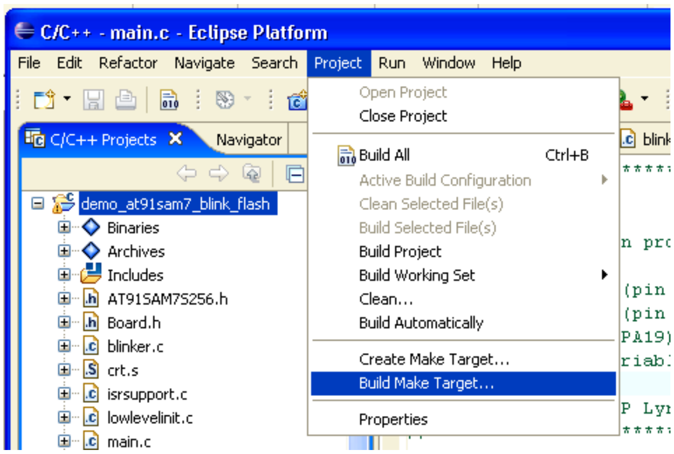
Build command: make

Build Setting

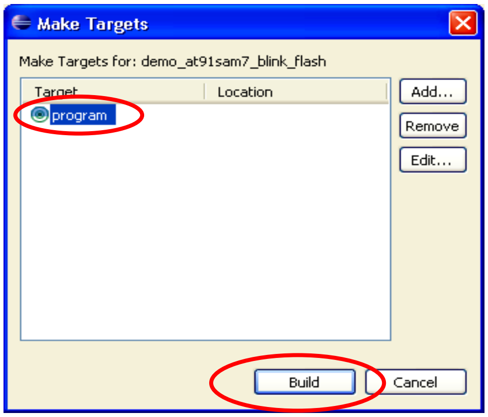
☐ Stop on first build error.

☑ Run all project builders.

There are two ways to execute the alternate Make target. The first way is to use the Project pull-down menu. Click on “**Project – Build Make Target**” as shown below.



Click on the “**program**” icon below to highlight it (there can be multiple alternate targets defined) and then click “**Build**” to execute the makefile alternate target called “program” and thereby program the FLASH.

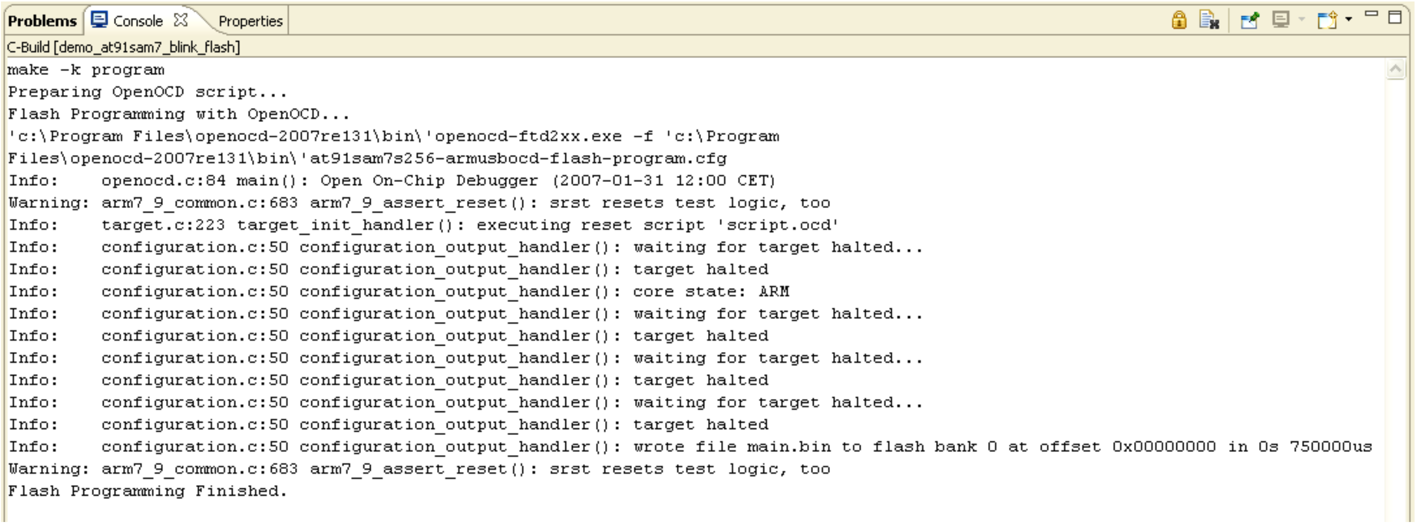


Make Targets for: demo_at91sam7_blink_flash

Target	Location
program	

The FLASH programming algorithm built into OpenOCD will now start. Since the sample program is relatively small, this will run through to completion in just a few seconds.

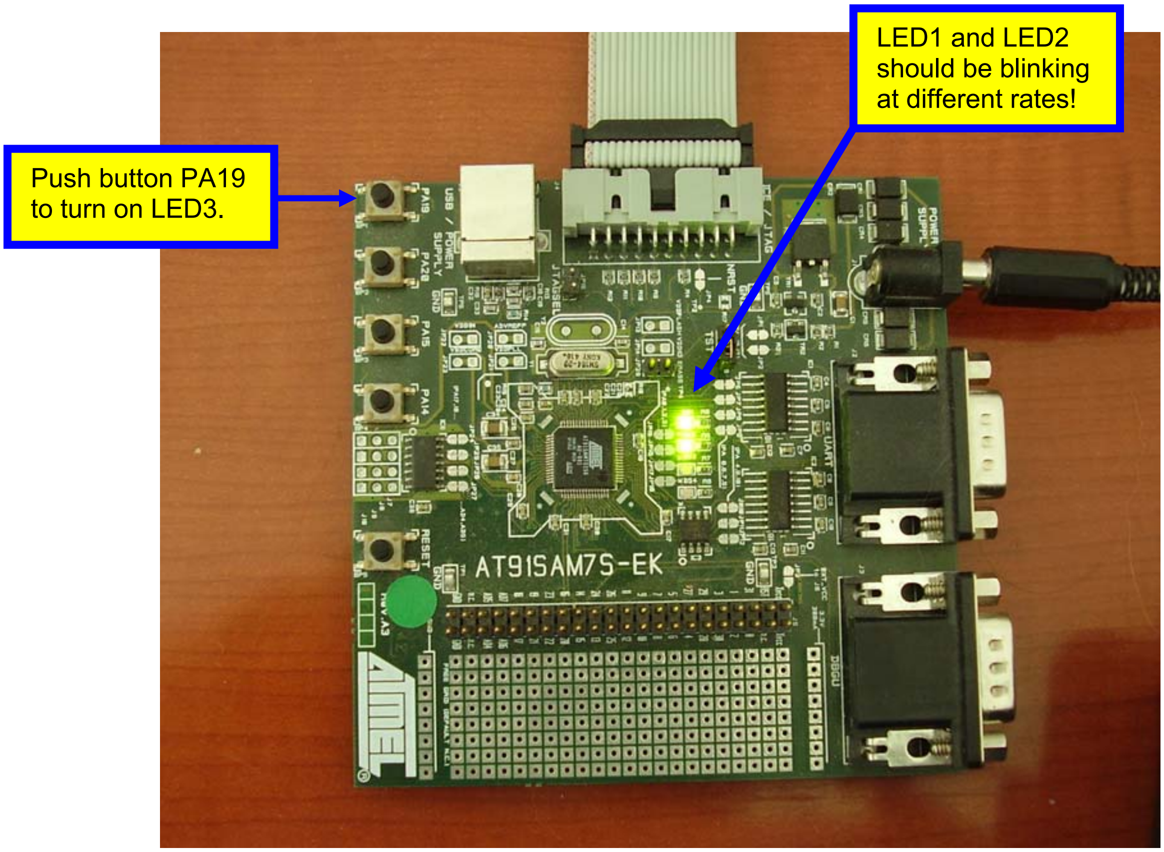
The results of the FLASH programming activity are displayed in the “Console” view as shown below.



C-Build [demo_at91sam7_blink_flash]

```
make -k program
Preparing OpenOCD script...
Flash Programming with OpenOCD...
'c:\Program Files\openocd-2007re131\bin\'openocd-ftd2xx.exe -f 'c:\Program
Files\openocd-2007re131\bin\'at91sam7s256-armusbocd-flash-program.cfg
Info:    openocd.c:84 main(): Open On-Chip Debugger (2007-01-31 12:00 CET)
Warning: arm7_9_common.c:683 arm7_9_assert_reset(): srst resets test logic, too
Info:    target.c:223 target_init_handler(): executing reset script 'script.ocd'
Info:    configuration.c:50 configuration_output_handler(): waiting for target halted...
Info:    configuration.c:50 configuration_output_handler(): target halted
Info:    configuration.c:50 configuration_output_handler(): core state: ARM
Info:    configuration.c:50 configuration_output_handler(): waiting for target halted...
Info:    configuration.c:50 configuration_output_handler(): target halted
Info:    configuration.c:50 configuration_output_handler(): waiting for target halted...
Info:    configuration.c:50 configuration_output_handler(): target halted
Info:    configuration.c:50 configuration_output_handler(): waiting for target halted...
Info:    configuration.c:50 configuration_output_handler(): target halted
Info:    configuration.c:50 configuration_output_handler(): wrote file main.bin to flash bank 0 at offset 0x00000000 in 0s 750000us
Warning: arm7_9_common.c:683 arm7_9_assert_reset(): srst resets test logic, too
Flash Programming Finished.
```

To test the application, hit the “**Reset**” button on the Atmel AT91SAM7S-EK. LED1 should be blinking at about a 1 Hz rate. LED2 should be blinking at a 10 Hz rate (Timer0 interrupt). If you push switch SW1, labeled PA19 on the AT91SAM7S256-EK board, you should see LED3 light up (it will turn off as part of the background loop).

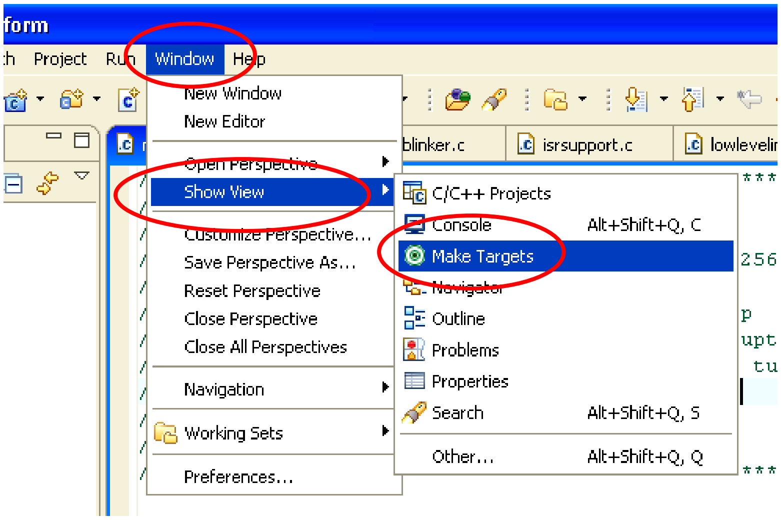


LED1 and LED2 should be blinking at different rates!

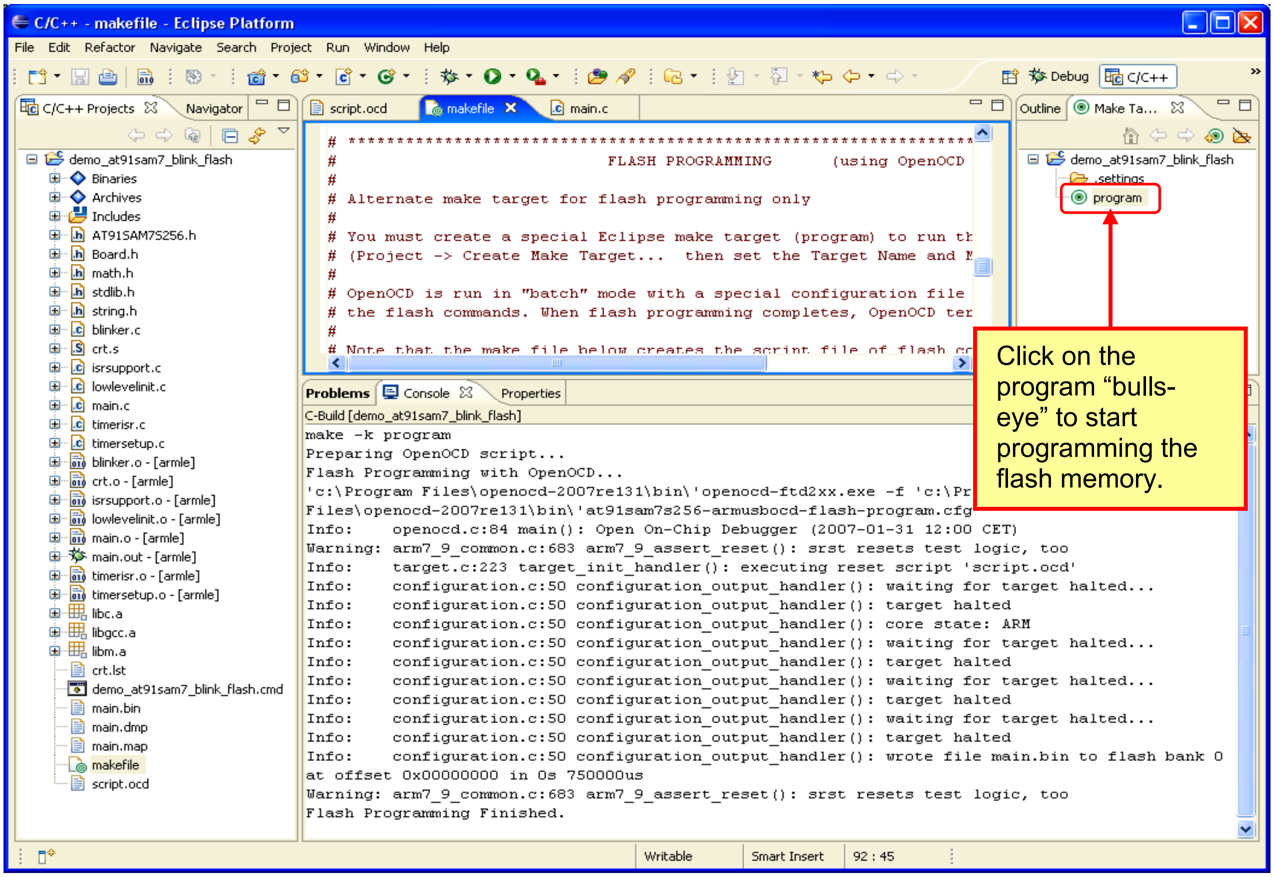
Push button PA19 to turn on LED3.

Congratulations! You now have a full-fledged ARM cross development system operational and it didn’t cost a thing!

There’s one other way to conveniently launch the alternate make target to program the flash. We can display a special “Make Targets” view. Click “**Window – Show View – Make Targets**” as shown below.



Now the “**Make Targets**” view is presented with the “**Outline**” view on the right and if you double-click on “**program’** the alternate make file target will immediately run and the FLASH will be programmed.



```
# ********************************************************************
#                          FLASH PROGRAMMING       (using OpenOCD
#
# Alternate make target for flash programming only
#
# You must create a special Eclipse make target (program) to run th
# (Project -> Create Make Target...  then set the Target Name and M
#
# OpenOCD is run in "batch" mode with a special configuration file
# the flash commands. When flash programming completes, OpenOCD ter
#
# Note that the make file below creates the script file of flash cc
```

```
make -k program
Preparing OpenOCD script...
Flash Programming with OpenOCD...
'c:\Program Files\openocd-2007re131\bin\'openocd-ftd2xx.exe -f 'c:\Pr
Files\openocd-2007re131\bin\'at91sam7s256-armusbocd-flash-program.cfg
Info:    openocd.c:84 main(): Open On-Chip Debugger (2007-01-31 12:00 CET)
Warning: arm7_9_common.c:683 arm7_9_assert_reset(): srst resets test logic, too
Info:    target.c:223 target_init_handler(): executing reset script 'script.ocd'
Info:    configuration.c:50 configuration_output_handler(): waiting for target halted...
Info:    configuration.c:50 configuration_output_handler(): target halted
Info:    configuration.c:50 configuration_output_handler(): core state: ARM
Info:    configuration.c:50 configuration_output_handler(): waiting for target halted...
Info:    configuration.c:50 configuration_output_handler(): target halted
Info:    configuration.c:50 configuration_output_handler(): waiting for target halted...
Info:    configuration.c:50 configuration_output_handler(): target halted
Info:    configuration.c:50 configuration_output_handler(): waiting for target halted...
Info:    configuration.c:50 configuration_output_handler(): target halted
Info:    configuration.c:50 configuration_output_handler(): wrote file main.bin to flash bank 0
at offset 0x00000000 in 0s 750000us
Warning: arm7_9_common.c:683 arm7_9_assert_reset(): srst resets test logic, too
Flash Programming Finished.
```

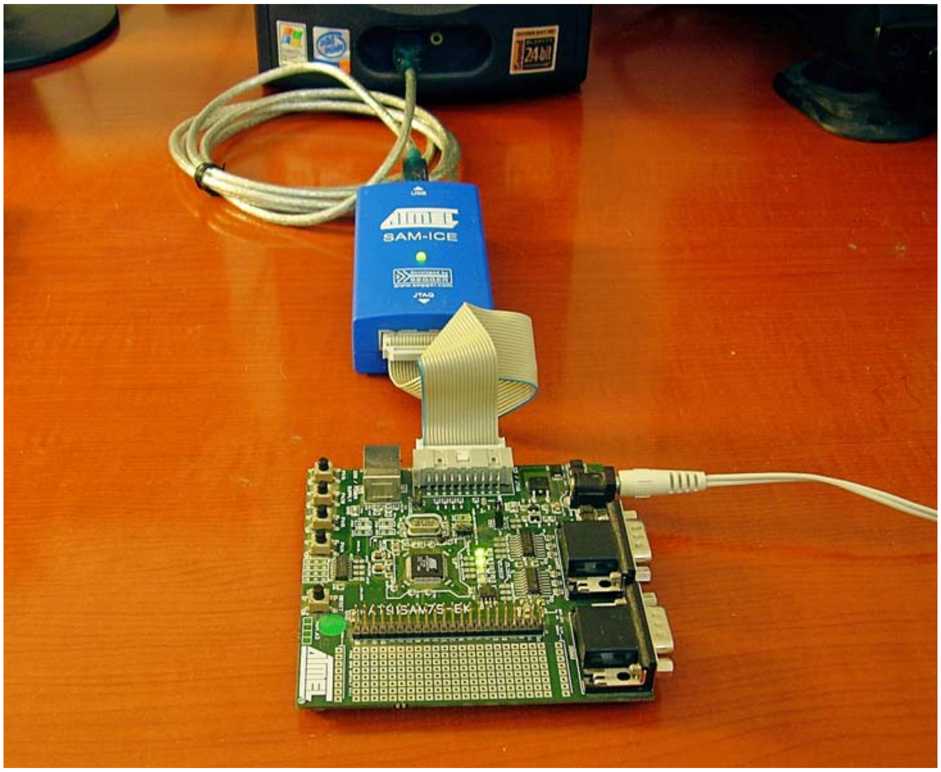
Click on the program “bulls-eye” to start programming the flash memory.

If you are having trouble getting the OpenOCD FLASH programming to work, make sure that the configuration files supplied with the sample programs were copied to the same folder that the OpenOCD executable resides (c:\Program Files\openocd-2007re141\bin). Verify that you adjusted the makefile to select the OpenOCD executable and OpenOCD configuration file that match the debugger hardware device you are using (wiggler, JTAGKey or ARMUSBOCD).

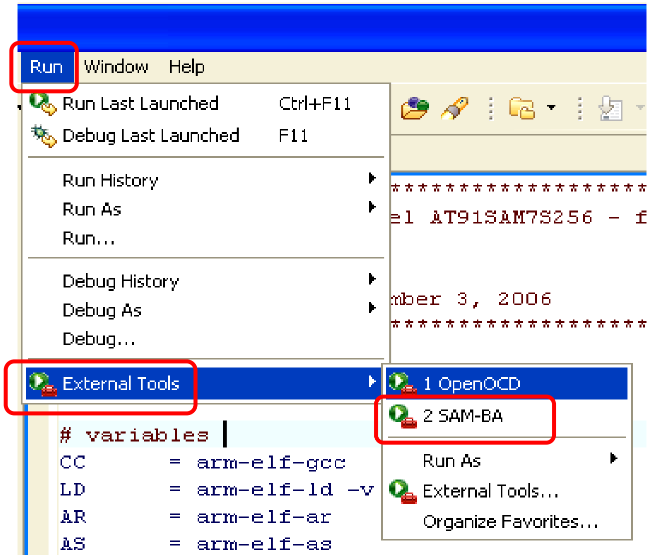
Using SAM-ICE and SAM-BA to Program the FLASH memory

If you have purchased the Atmel SAM-ICE JTAG interface, you can use the free Atmel SAM-BA Flash Programming Utility to program the FLASH using the JTAG connection as described in this section.

In the hardware setup shown below, the AT91SAM7S-EK evaluation board is powered by a simple 9 volt "wall wart" power supply. The SAM-ICE JTAG interface is connected to the PC with a standard USB cable and is connected to the target board's 20-pin JTAG connecter with a ribbon cable.



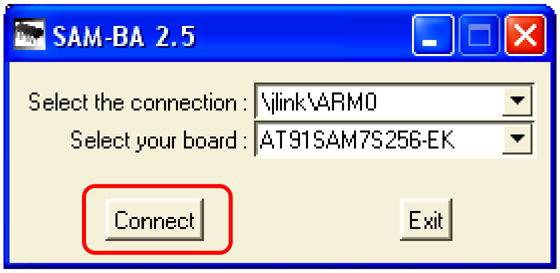
Using the Eclipse "**Run**" pull-down menu, click on "**Run – External Tools – SAM-BA**".



```
*******************
el AT91SAM7S256 - f

mber 3, 2006
*******************
```

```
# variables
CC      = arm-elf-gcc
LD      = arm-elf-ld -v
AR      = arm-elf-ar
AS      = arm-elf-as
```

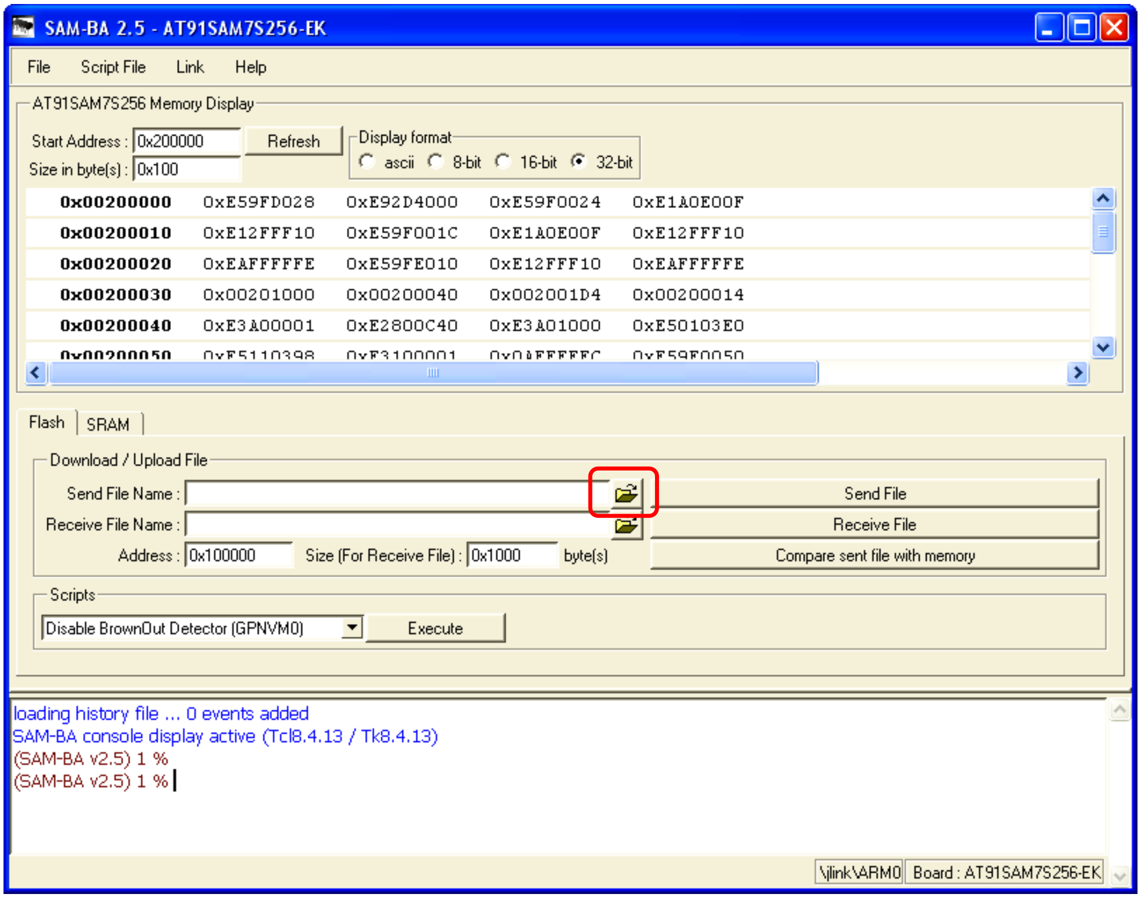
When the small SAM-BA communications dialog window appears, select your board (in this case, it's "**AT91SAM7S256-EK**"). Also select the "**\jlink\ARM0**" connection. Click "**Connect**" to establish a link to the SAM-ICE.



Select the connection : \jlink\ARM0
Select your board : AT91SAM7S256-EK

The SAB-BA main screen will appear. There's a nice memory display on the top wherein you can browse memory. Under the "Scripts" view, there's a script to erase the entire flash. Since programming the flash with SAM-BA automatically erases it first, there's rarely a need to use the "erase" script.

SAM-BA defaults to the "Flash" tab; click it if this is not the case.



Start Address : 0x200000
Size in byte(s) : 0x100
Display format: ○ ascii ○ 8-bit ○ 16-bit ● 32-bit

```
0x00200000  0xE59FD028  0xE92D4000  0xE59F0024  0xE1A0E00F
0x00200010  0xE12FFF10  0xE59F001C  0xE1A0E00F  0xE12FFF10
0x00200020  0xEAFFFFFE  0xE59FE010  0xE12FFF10  0xEAFFFFFE
0x00200030  0x00201000  0x00200040  0x002001D4  0x00200014
0x00200040  0xE3A00001  0xE2800C40  0xE3A01000  0xE50103E0
0x00200050  0xE5110398  0xE3100001  0x0AFFFFFC  0xE59F0050
```

Send File Name :
Receive File Name :
Address : 0x100000 Size (For Receive File) : 0x1000 byte(s)

Scripts: Disable BrownOut Detector (GPNVM0)

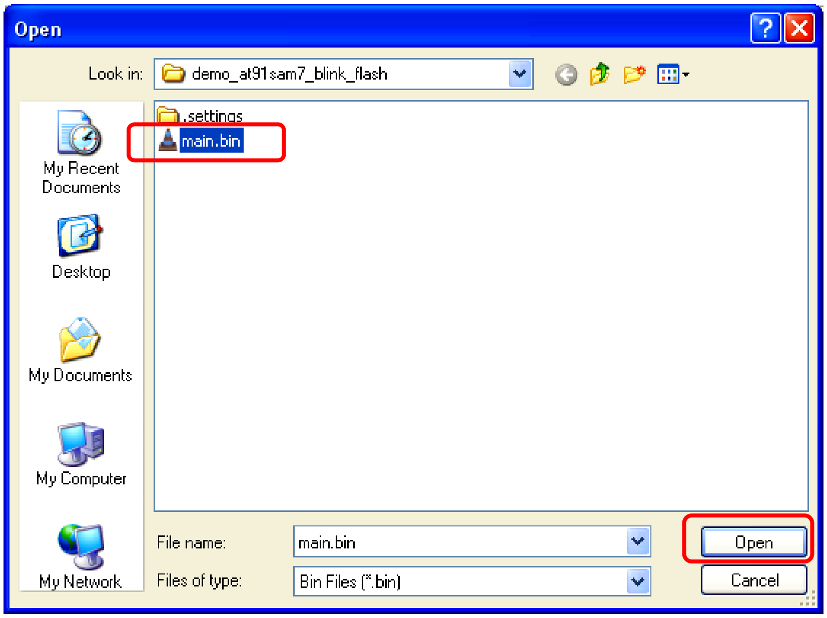
```
loading history file ... 0 events added
SAM-BA console display active (Tcl8.4.13 / Tk8.4.13)
(SAM-BA v2.5) 1 %
(SAM-BA v2.5) 1 %
```

\jlink\ARM0 Board : AT91SAM7S256-EK

Click on the 📂 symbol associated with the "Send File Name:" text box above

to bring up a standard file navigation screen and use it to browse to the project's "**main.bin**" file.

Select the “**main.bin**” as shown below and click “**Open**” to select this file to be programmed into your FLASH on-chip memory.



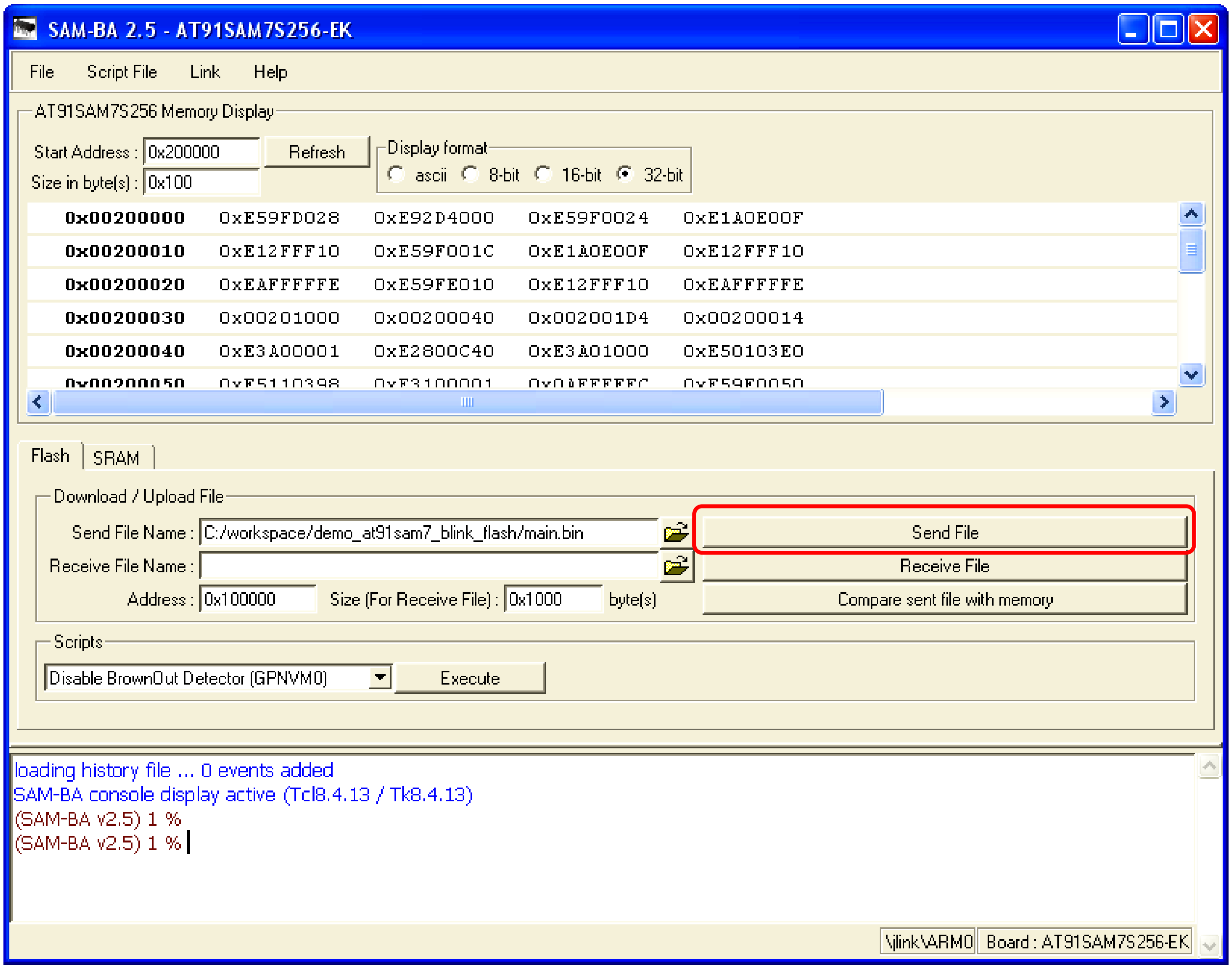
Look in: demo_at91sam7_blink_flash

.settings
main.bin

File name: main.bin

Files of type: Bin Files (*.bin)

Now in the main SAM-BA screen below, click on “**Send File**” to program the flash.



Start Address : 0x200000

Size in byte(s) : 0x100

Display format: ○ ascii ○ 8-bit ○ 16-bit ● 32-bit

```
0x00200000   0xE59FD028   0xE92D4000   0xE59F0024   0xE1A0E00F
0x00200010   0xE12FFF10   0xE59F001C   0xE1A0E00F   0xE12FFF10
0x00200020   0xEAFFFFFE   0xE59FE010   0xE12FFF10   0xEAFFFFFE
0x00200030   0x00201000   0x00200040   0x002001D4   0x00200014
0x00200040   0xE3A00001   0xE2800C40   0xE3A01000   0xE50103E0
0x00200050   0xE5110398   0xE3100001   0x0AFFFFFC   0xE59F0050
```

Send File Name : C:/workspace/demo_at91sam7_blink_flash/main.bin

Receive File Name :

Address : 0x100000 Size (For Receive File) : 0x1000 byte(s)

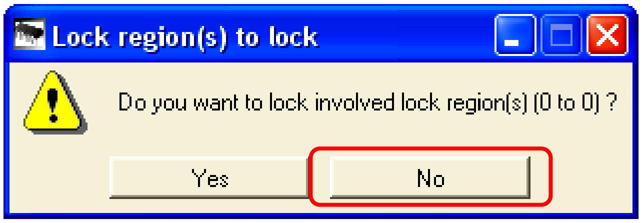
Scripts: Disable BrownOut Detector (GPNVM0)

```
loading history file ... 0 events added
SAM-BA console display active (Tcl8.4.13 / Tk8.4.13)
(SAM-BA v2.5) 1 %
(SAM-BA v2.5) 1 %
```

\jlink\ARM0 Board : AT91SAM7S256-EK

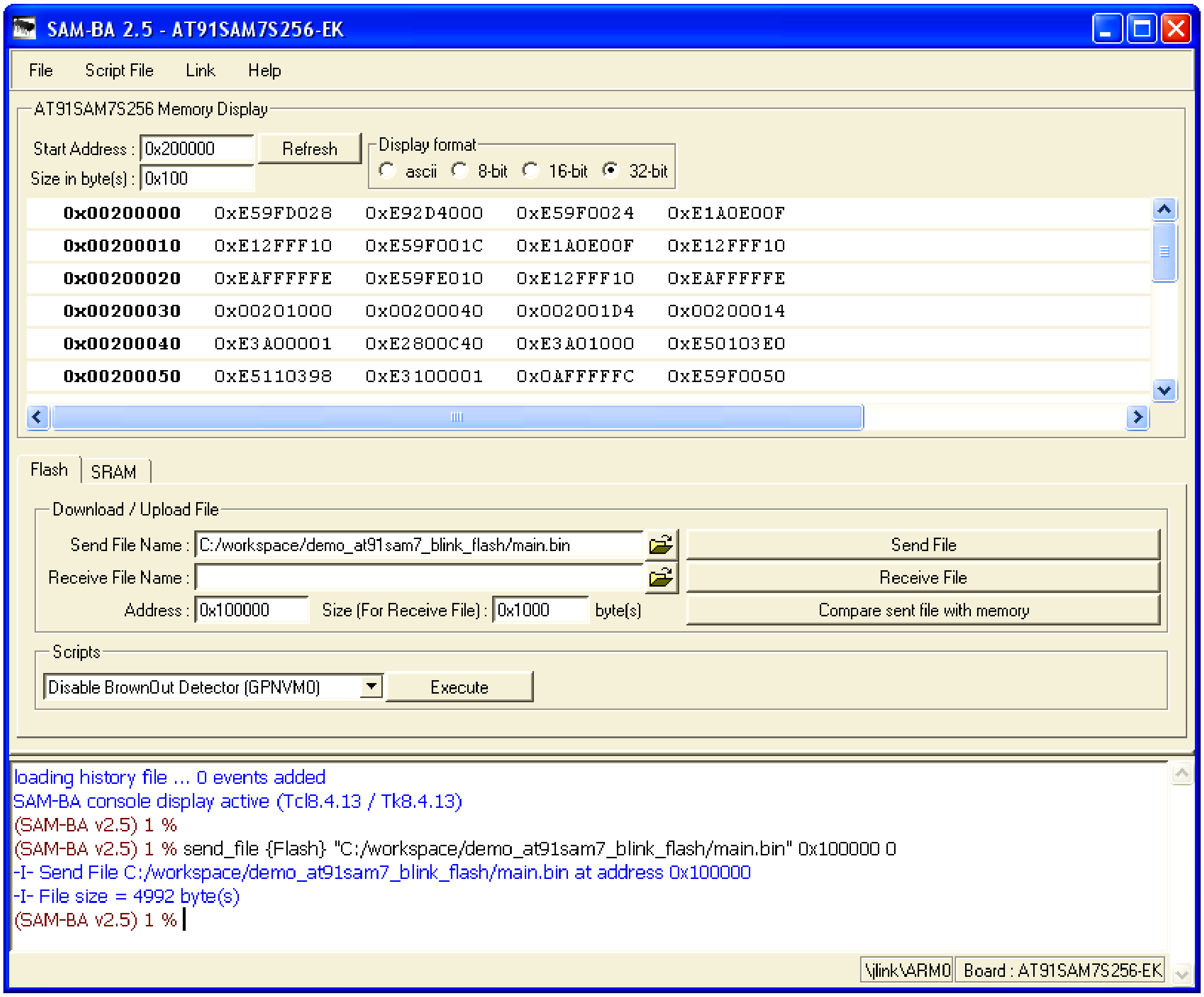
If one or more of the flash regions is “locked”, SAM-BA may ask you if you want to “unlock” the region. Always answer affirmative (Yes) since we don’t want any locked regions before we start programming.

After programming the flash, SAM-BA will ask you if you want to lock the regions just programmed. To be on the safe side, always answer “**No**” and leave the regions unlocked as shown below.



Do you want to lock involved lock region(s) (0 to 0) ?

The little console display at the bottom indicates that 4992 bytes were sent to the flash memory.



Start Address : 0x200000

Size in byte(s) : 0x100

Display format: ○ ascii ○ 8-bit ○ 16-bit ● 32-bit

```
0x00200000   0xE59FD028   0xE92D4000   0xE59F0024   0xE1A0E00F
0x00200010   0xE12FFF10   0xE59F001C   0xE1A0E00F   0xE12FFF10
0x00200020   0xEAFFFFFE   0xE59FE010   0xE12FFF10   0xEAFFFFFE
0x00200030   0x00201000   0x00200040   0x002001D4   0x00200014
0x00200040   0xE3A00001   0xE2800C40   0xE3A01000   0xE50103E0
0x00200050   0xE5110398   0xE3100001   0x0AFFFFFC   0xE59F0050
```

Send File Name : C:/workspace/demo_at91sam7_blink_flash/main.bin

Receive File Name :

Address : 0x100000 Size (For Receive File) : 0x1000 byte(s)

Disable BrownOut Detector (GPNVM0)

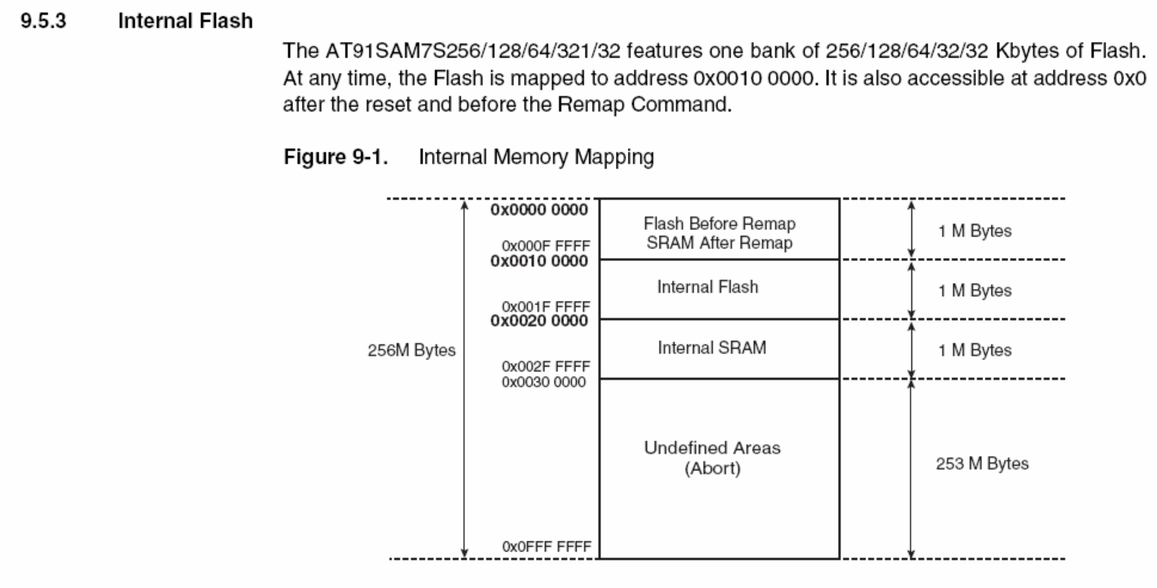
```
loading history file ... 0 events added
SAM-BA console display active (Tcl8.4.13 / Tk8.4.13)
(SAM-BA v2.5) 1 %
(SAM-BA v2.5) 1 % send_file {Flash} "C:/workspace/demo_at91sam7_blink_flash/main.bin" 0x100000 0
-I- Send File C:/workspace/demo_at91sam7_blink_flash/main.bin at address 0x100000
-I- File size = 4992 byte(s)
(SAM-BA v2.5) 1 %
```

\jlink\ARM0 Board : AT91SAM7S256-EK

If you look at the AT91SAM7S-EK evaluation board, you will notice that the application appears frozen.

You must do a power-cycle to get the application to start. This seems to be a bug in the Revision 2.5 of the SAM-BA.

Alert readers might notice that the summary indicates that the 4992 bytes were loaded at address 0x100000. This is true. However, page 19 of AT91SAM7S256 data package shows that FLASH is actually at address 0x100000 and is subsequently “mapped” into the 1 mb region at address 0x000000 at boot when the remap control register MC_RCR bit 0 is cleared (the default).



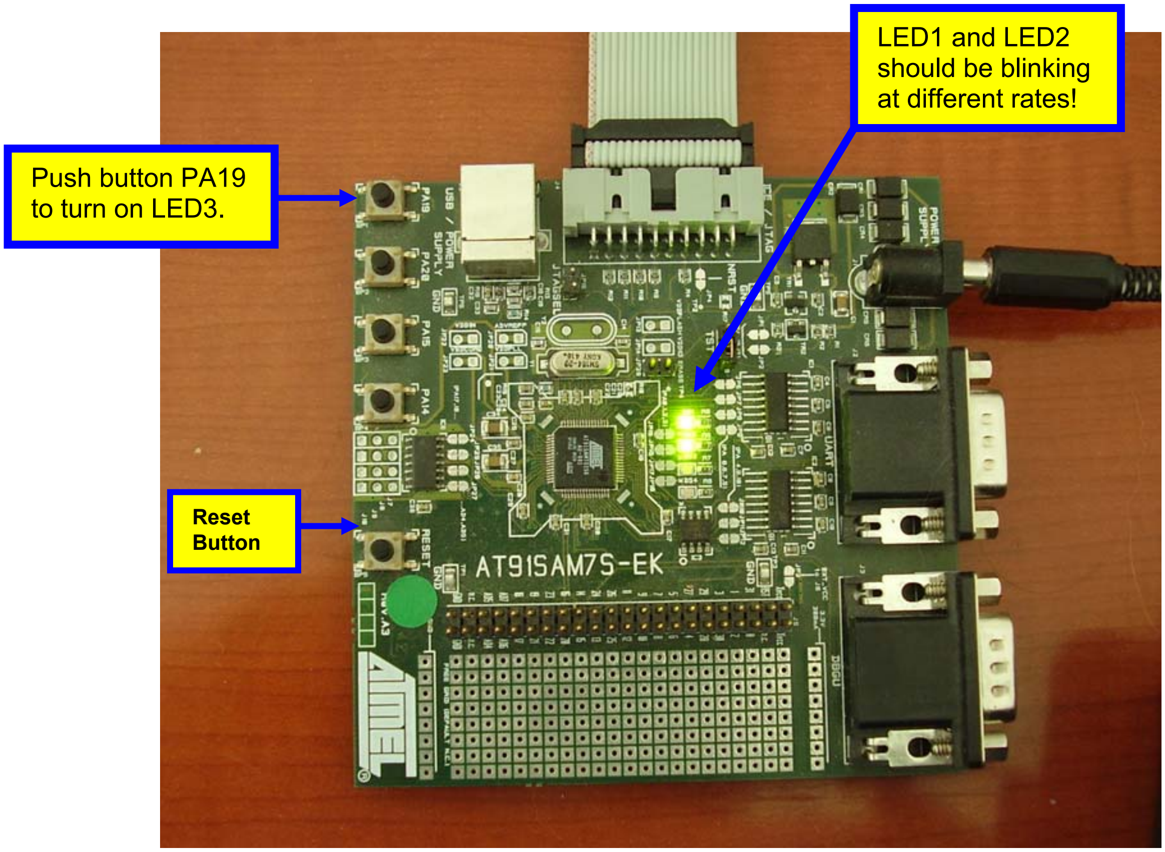
9.5.3 **Internal Flash**

The AT91SAM7S256/128/64/321/32 features one bank of 256/128/64/32/32 Kbytes of Flash. At any time, the Flash is mapped to address 0x0010 0000. It is also accessible at address 0x0 after the reset and before the Remap Command.

Figure 9-1. Internal Memory Mapping

0x0000 0000
0x000F FFFF
0x0010 0000
0x001F FFFF
0x0020 0000
0x002F FFFF
0x0030 0000
0x0FFF FFFF
256M Bytes
Flash Before Remap
SRAM After Remap
Internal Flash
Internal SRAM
Undefined Areas
(Abort)
1 M Bytes
1 M Bytes
1 M Bytes
253 M Bytes

To test the application, cycle the power and hit the “Reset” button on the Atmel AT91SAM7S-EK. The board is still powered from the “wall wart” DC power supply. The LEDs should start blinking.



Congratulations! You now have a full-fledged ARM cross development system operational and it didn’t cost a thing!

Debugging the FLASH Application

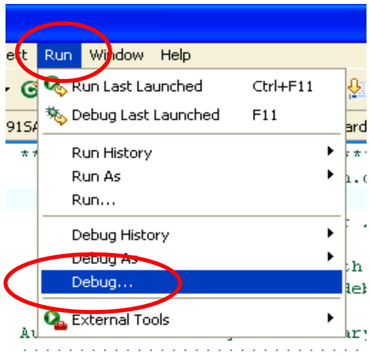
The author once interviewed a job applicant whose response to the question "Describe your debugging technique?" was "I try not to make any errors!" Well, unless you are an infallible programmer like that guy, you will occasionally require the services of a debugger to trap and identify software errors. Eclipse has a wonderful visual source code debugger that interfaces to the GNU GDB debugger.

You can debug an application programmed into on chip FLASH; the built-in on chip JTAG debug circuits allow this. There is only one restriction; you are limited to just two breakpoints. Attempting to specify more than two hardware breakpoints at a time may cause the debugger to malfunction. Otherwise all Eclipse debugging features work properly, such as single-stepping, inspection and modification of variables, memory dumps, etc.

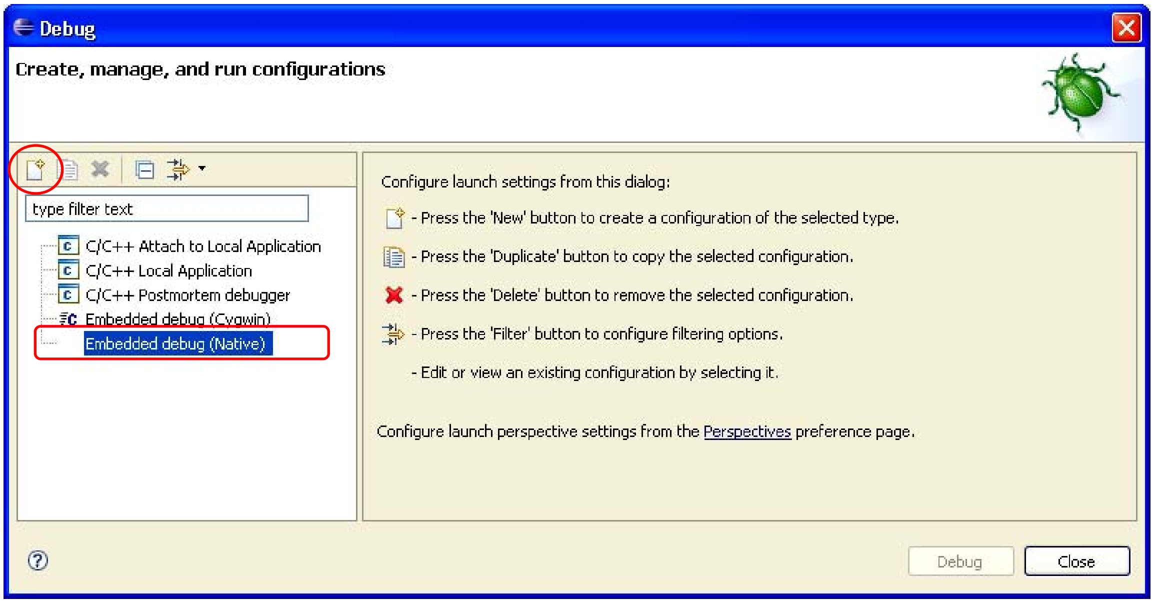
Create a Debug Launch Configuration

Before we can debug the FLASH application, we have to create a Debug Launch Configuration for this project. The Debug Launch Configuration locates the GDB debugger for Eclipse, locates the project's executable file (in this case it's only used to look up symbol information), and provides a startup script of GDB commands that are to be run as the debugger starts up. Most people will define a Debug Launch Configuration for each project they create.

Click on "**Run – Debug…**" to bring up the Debug Configuration Window.



In the "Debug – Create, manage, and run configurations" window shown below, click on "**Embedded debug (Native)**" followed by the "**New**" button. This is the special debug launch configuration created by Zylin.



Configure launch settings from this dialog:

- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.

Configure launch perspective settings from the Perspectives preference page.

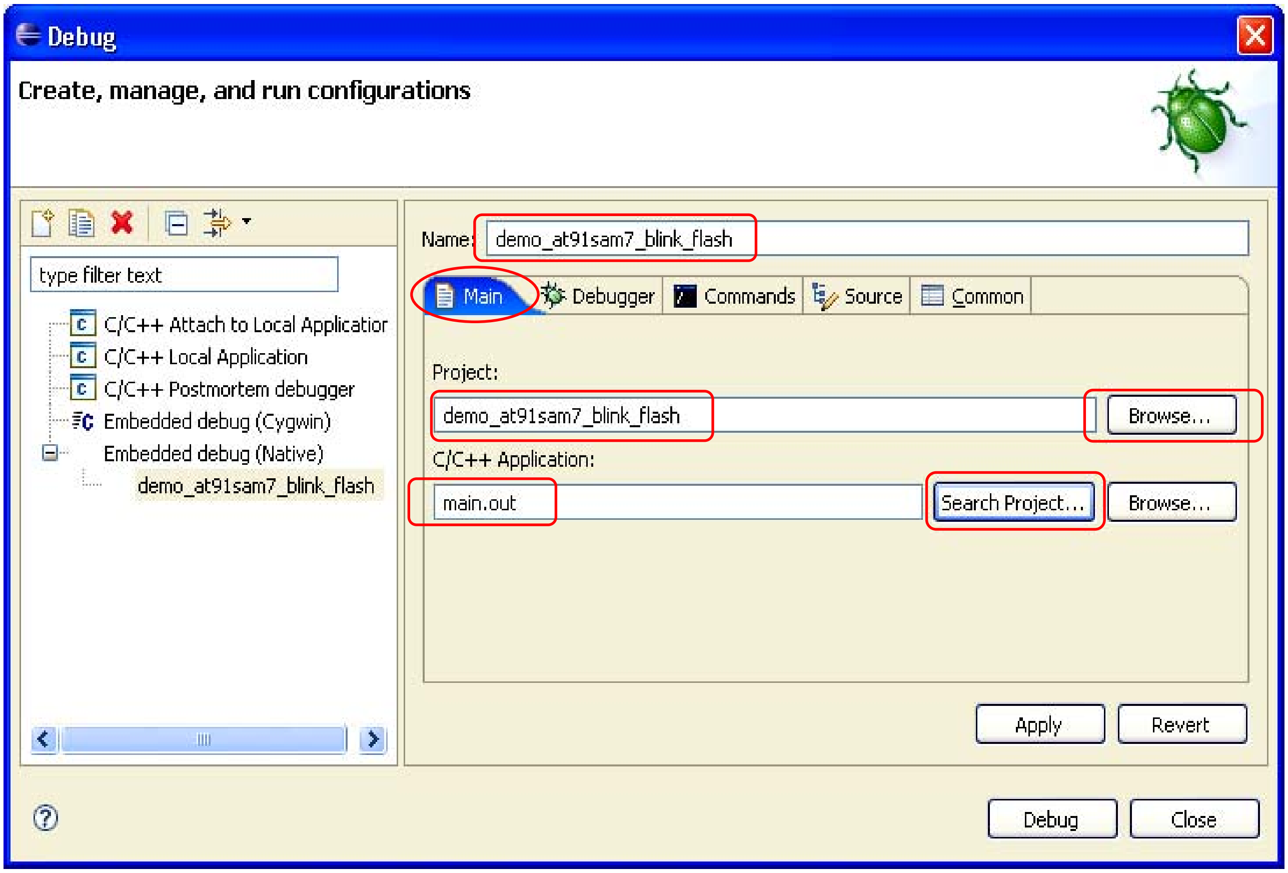
The Debug “Create, manage and run configurations” window changes to the dialog shown below. Start by making sure that the “Main” tab is selected.

In the “Name:” text box, enter the name of this debug launch configuration. The Name can be anything you choose, but since there is usually going to be a debug configuration for each project you set up, the name of the project itself is a wise choice. In this example, we simply use the project name “**demo_at91sam7_blink_flash**” for this purpose.

In the “Project” text box, use the “**Browse**” button to find the project “**demo_at91sam7_blink_flash**”.

In the “C/C++ Application” text box, use the “**Search Project…**” button to find the application file “**main.out”**.

You might be inclined to ask why this is not the “**main.bin**” file? The binary file was used earlier to program the flash, but the debugger needs the application file that has the symbols; this is the “**main.out**” file. While the “**main.out**” file also has the executable code within it, the debugger only uses the symbol information for FLASH debugging.



Debug

Create, manage, and run configurations

Name: demo_at91sam7_blink_flash

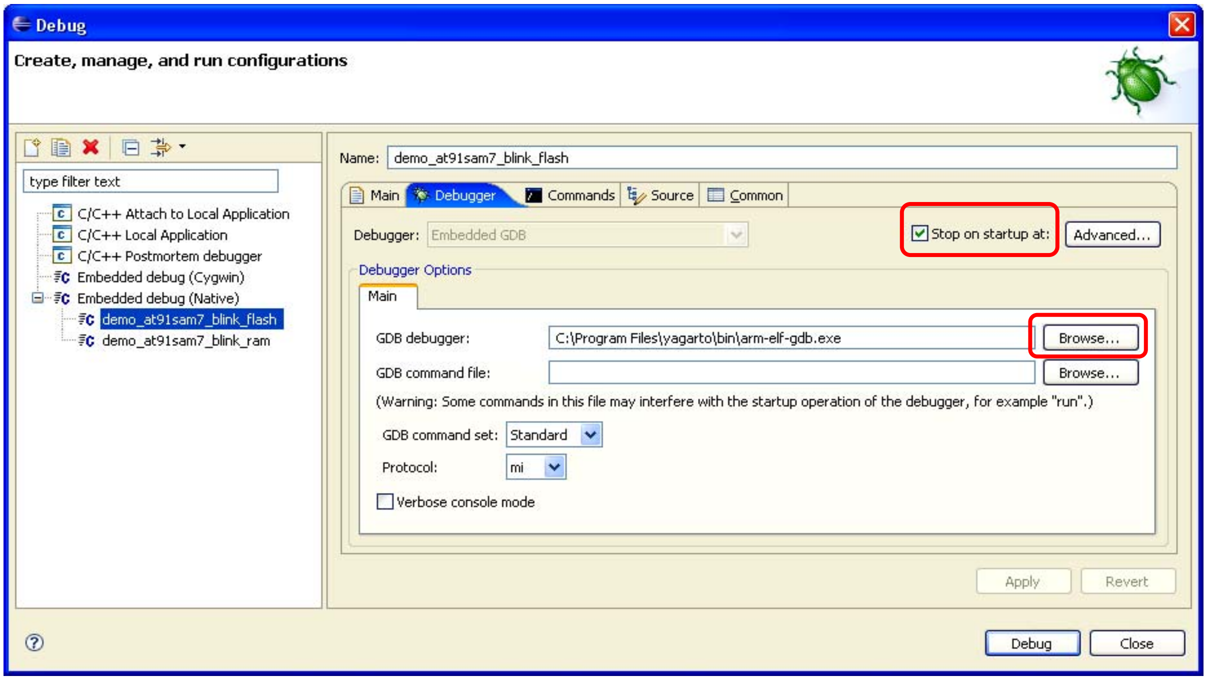
Project: demo_at91sam7_blink_flash

C/C++ Application: main.out

Now select the “Debugger” tab as shown below.

Check the check box that says “**Stop on startup at...**” as this provides our breakpoint at the entry point of main( ).

In the dialog labeled “Debugger Options”, use the “**Browse**” button to locate the GDB Debugger “**arm-elf-gdb.exe**” file. It will be found in the “c:\Program Files\yagarto\bin” folder. The rest of the dialog can be left in its default form.



Name: demo_at91sam7_blink_flash

Debugger: Embedded GDB

☑ Stop on startup at:

GDB debugger: C:\Program Files\yagarto\bin\arm-elf-gdb.exe

GDB command file:

(Warning: Some commands in this file may interfere with the startup operation of the debugger, for example "run".)

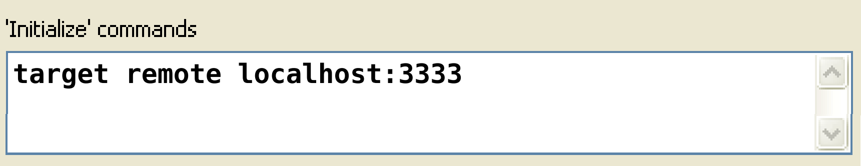
GDB command set: Standard

Protocol: mi

☐ Verbose console mode

Now select the “Comands” tab as shown below.

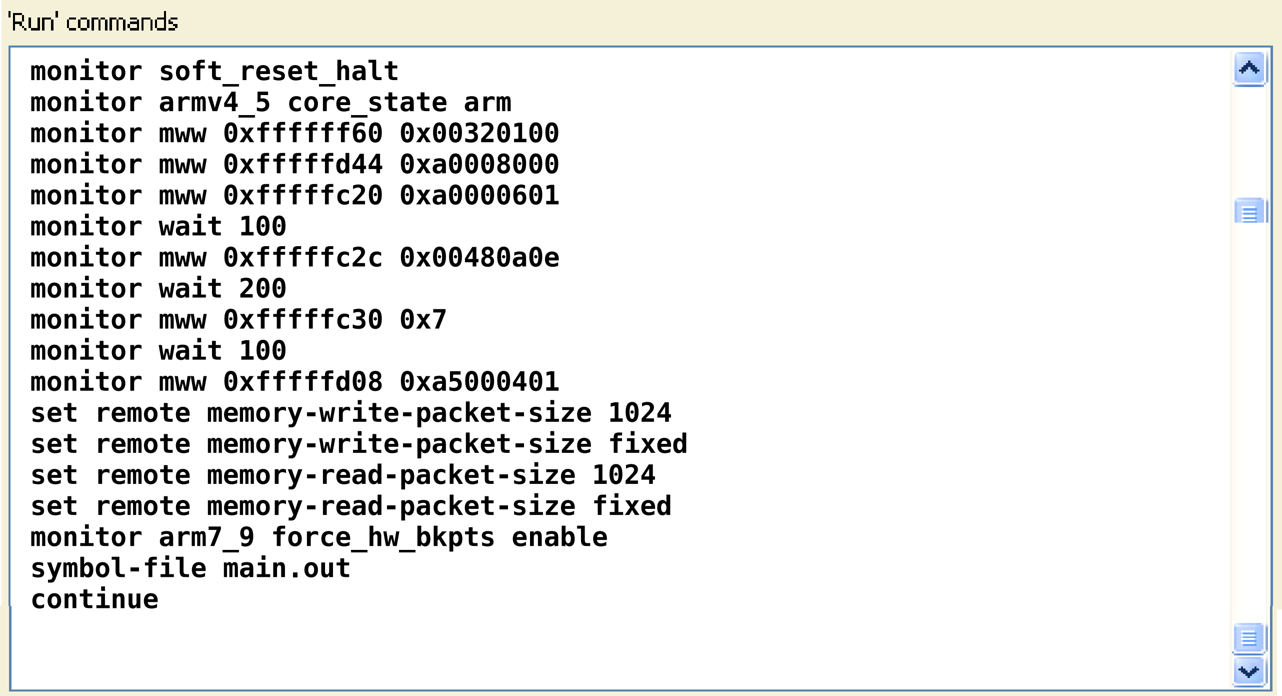
If you are using OpenOCD, enter the single GDB command “**target remote localhost:3333**” in the “Initialize commands” text window exactly as shown below. This command tells the GDB debugger to emit commands in RSP format to the TCP port “localhost:3333” (the port OpenOCD will be listening to).



'Initialize' commands

```
target remote localhost:3333
```

If you are using OpenOCD, enter the following GDB and OpenOCD commands into the “Run commands” text window, exactly as shown below. The “Source” and “Common” tabs can be left in their default state.

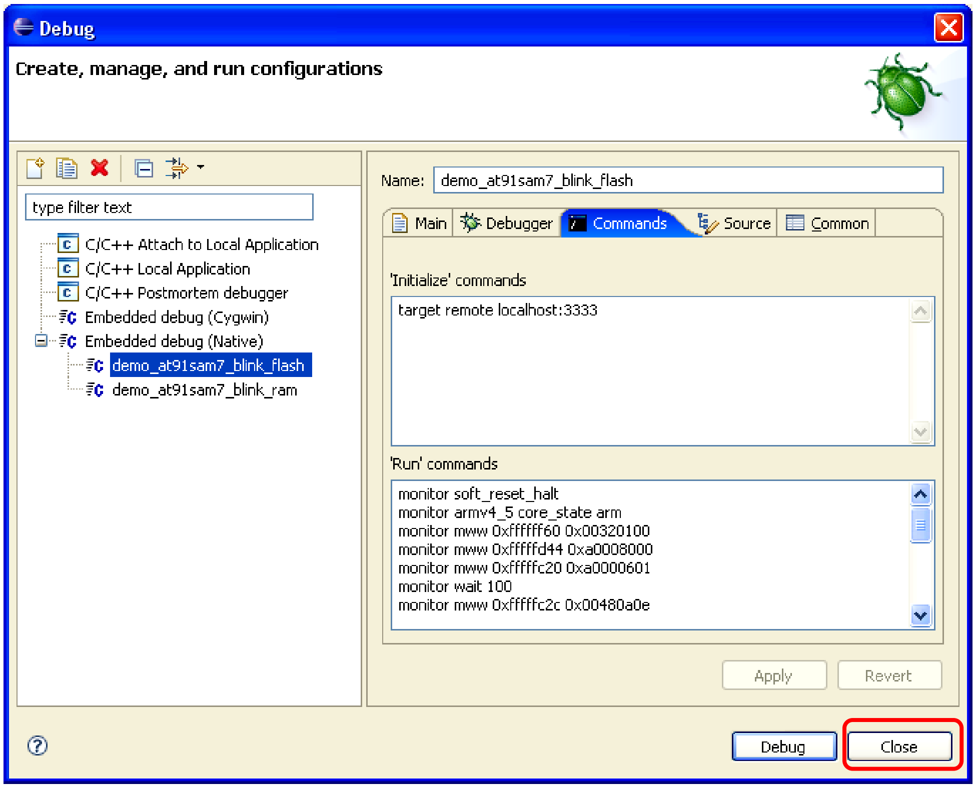


'Run' commands

```
monitor soft_reset_halt
monitor armv4_5 core_state arm
monitor mww 0xffffff60 0x00320100
monitor mww 0xfffffd44 0xa0008000
monitor mww 0xfffffc20 0xa0000601
monitor wait 100
monitor mww 0xfffffc2c 0x00480a0e
monitor wait 200
monitor mww 0xfffffc30 0x7
monitor wait 100
monitor mww 0xfffffd08 0xa5000401
set remote memory-write-packet-size 1024
set remote memory-write-packet-size fixed
set remote memory-read-packet-size 1024
set remote memory-read-packet-size fixed
monitor arm7_9 force_hw_bkpts enable
symbol-file main.out
continue
```

Below is the Debug Launch Configuration “Commands” tab for use with OpenOCD and flash execution. Note that the ‘Run’ commands window below only shows a portion of the commands that were entered. Be sure to enter all the commands as shown above.

The “Source” and “Common” tabs can be left in their default condition. Click on “**Close**” to complete definition of the Debug Launch Configuration for flash debugging with OpenOCD.



Name: demo_at91sam7_blink_flash

'Initialize' commands

```
target remote localhost:3333
```

'Run' commands

```
monitor soft_reset_halt
monitor armv4_5 core_state arm
monitor mww 0xffffff60 0x00320100
monitor mww 0xfffffd44 0xa0008000
monitor mww 0xfffffc20 0xa0000601
monitor wait 100
monitor mww 0xfffffc2c 0x00480a0e
```

To make entry of the ‘Run’ commands more convenient, here is a list of them for “cut-and-paste” transfer to Eclipse.

```
monitor soft_reset_halt
monitor armv4_5 core_state arm
monitor mww 0xffffff60 0x00320100
monitor mww 0xfffffd44 0xa0008000
monitor mww 0xfffffc20 0xa0000601
monitor wait 100
monitor mww 0xfffffc2c 0x00480a0e
monitor wait 200
monitor mww 0xfffffc30 0x7
monitor wait 100
monitor mww 0xfffffd08 0xa5000401
set remote memory-write-packet-size 1024
set remote memory-write-packet-size fixed
set remote memory-read-packet-size 1024
set remote memory-read-packet-size fixed
monitor arm7_9 force_hw_bkpts enable
symbol-file main.out
continue
```

Copy these commands into the “Run Commands” window.

The GDB startup commands for OpenOCD operation shown above require some explanation. If the command line starts with the word “monitor”, then that command is an OpenOCD command. Otherwise, it is a legacy GDB command.

OpenOCD commands are described in the OpenOCD documentation which can be downloaded from:

http://developer.berlios.de/docman/display_doc.php?docid=1367&group_id=4148

GDB commands are described in several books and in the official document that can be downloaded from:
http://dsl.ee.unsw.edu.au/dsl-cdrom/gnutools/doc/gnu-debugger.pdf

First, we have to halt the processor.

```
monitor soft_reset_halt                    # OpenOCD command to halt the processor and wait
```

Next, we identify the ARM core being used

```
monitor armv4_5 core_state arm             # OpenOCD command to select the core state
```

Now we set up the processor's clocks, etc. using the register settings in the lowlevelinit.c function. These are OpenOCD memory write commands used to set the various AT91SAM7S256 clock registers. This guarantees that the processor will be running at full speed when the "continue" command is asserted.

```
monitor mww 0xffffff60 0x00320100          # set flash wait state (AT91C_MC_FMR)
monitor mww 0xfffffd44 0xa0008000          # watchdog disable (AT91C_WDTC_WDMR)
monitor mww 0xfffffc20 0xa0000601          # enable main oscillator (AT91C_PMC_MOR)
monitor wait 100                           # wait 100 ms
monitor mww 0xfffffc2c 0x00480a0e          # set PLL register (AT91C_PMC_PLLR)
monitor wait 200                           # wait 200 ms
monitor mww 0xfffffc30 0x7                 # set master clock to PLL (AT91C_PMC_MCKR)
monitor wait 100                           # wait 100 ms
```

Enable the Reset button in the AT91SAM7S-EK board.

```
monitor mww 0xfffffd08 0xa5000401          # enable user reset AT91C_RSTC_RMR
```

Now increase the GDB packet size to 1024. This will have a slight improvement on FLASH debugging as reads of large data structures, etc. may be speeded up. These are legacy GDB commands.

```
set remote memory-write-packet-size 1024   # Setup GDB for faster downloads
set remote memory-write-packet-size fixed  # Setup GDB for faster downloads
set remote memory-read-packet-size 1024    # Setup GDB for faster downloads
set remote memory-read-packet-size fixed   # Setup GDB for faster downloads
```

This is an OpenOCD command to convert all Eclipse breakpoints to "hardware" breakpoints. Remember, we are only allowed two hardware breakpoints – defining more than two will crash the debugger.

```
monitor arm7_9 force_hw_bkpts enable       # convert all breakpoints to hardware breakpoints
```

Now we have to identify the file that has the symbol information. This is a legacy GDB command.

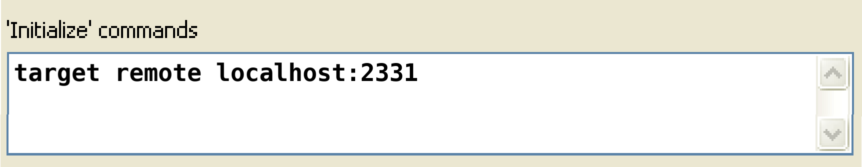
```
symbol-file main.out                       # read the symbol information from main.out
```

Finally we emit the legacy GDB command "continue". The processor was already halted at the Reset vector and will thus start executing until it hits the breakpoint set at main( ).

```
continue                                   # resume execution from reset vector - will break at main( )
```

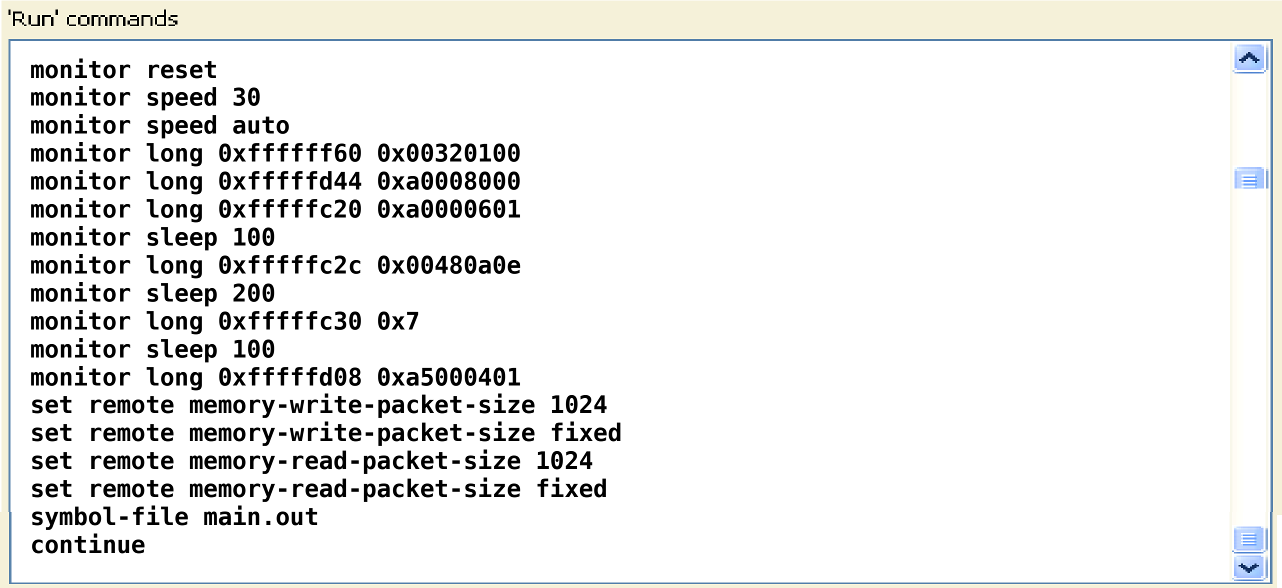
Author's Note: GDB manual states "Any text from a # to the end of a line is a comment; it does nothing". Unfortunately, I've noted that these systems get tripped up occasionally by these comments so they have been left out of all debug windows.

<u>If you are using the J-Link GDB Server</u>, enter the single GDB command “**target remote localhost:2331**” in the “Initialize commands” text window exactly as shown below. This command tells the GDB debugger to emit commands in RSP format to the TCP port “localhost:2331” (the port the J-Link GDB Server will be listening to).



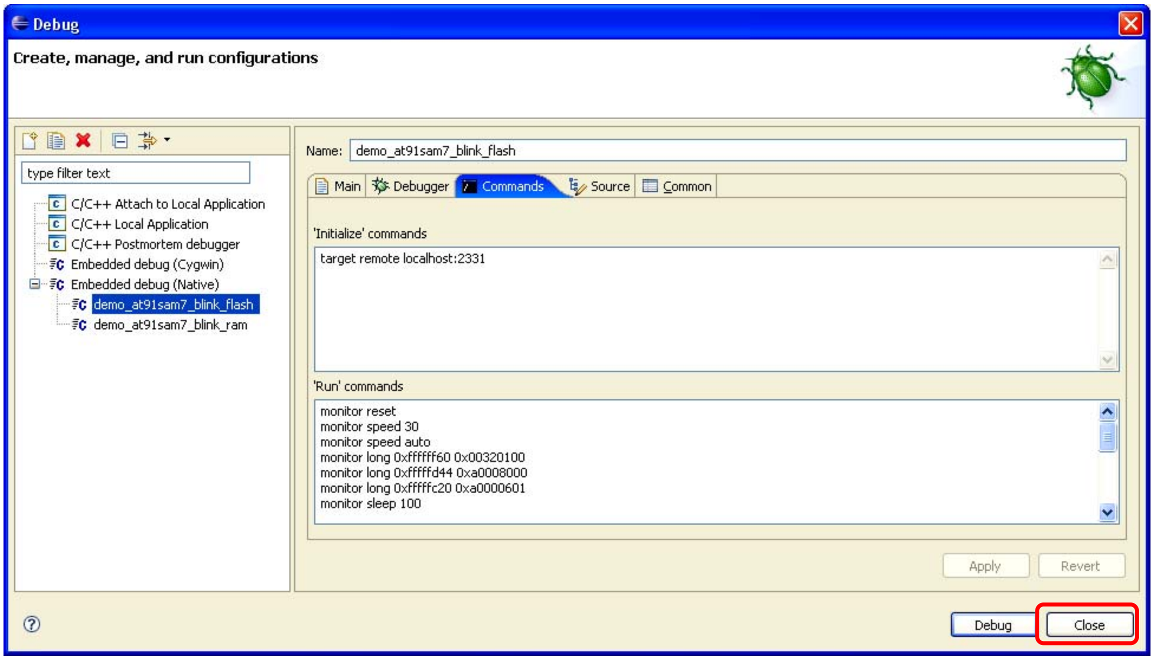
```
target remote localhost:2331
```

<u>If you are using the J-Link GDB Server</u>, enter the following GDB and J-Link GDB Server commands into the “Run commands” text window, exactly as shown below. The “Source” and “Common” tabs can be left in their default state.



```
monitor reset
monitor speed 30
monitor speed auto
monitor long 0xffffff60 0x00320100
monitor long 0xfffffd44 0xa0008000
monitor long 0xfffffc20 0xa0000601
monitor sleep 100
monitor long 0xfffffc2c 0x00480a0e
monitor sleep 200
monitor long 0xfffffc30 0x7
monitor sleep 100
monitor long 0xfffffd08 0xa5000401
set remote memory-write-packet-size 1024
set remote memory-write-packet-size fixed
set remote memory-read-packet-size 1024
set remote memory-read-packet-size fixed
symbol-file main.out
continue
```

Below is the Debug Launch Configuration “Commands” tab for use with the J-Link GDB Server and FLASH execution. Note that the ‘Run’ commands window only shows a portion of the commands that were entered. Be sure to enter all the commands as shown above.



Name: demo_at91sam7_blink_flash

'Initialize' commands:

```
target remote localhost:2331
```

'Run' commands:

```
monitor reset
monitor speed 30
monitor speed auto
monitor long 0xffffff60 0x00320100
monitor long 0xfffffd44 0xa0008000
monitor long 0xfffffc20 0xa0000601
monitor sleep 100
```

Click on "**Close**" above to complete definition of the Debug Launch Configuration for flash debugging with the J-Link GDB Server.

To make entry of the 'Run' commands more convenient, here is a list of them for "cut-and-paste" transfer to Eclipse.

```
monitor reset
monitor speed 30
monitor speed auto
monitor long 0xffffff60 0x00320100
monitor long 0xfffffd44 0xa0008000
monitor long 0xfffffc20 0xa0000601
monitor sleep 100
monitor long 0xfffffc2c 0x00480a0e
monitor sleep 200
monitor long 0xfffffc30 0x7
monitor sleep 100
monitor long 0xfffffd08 0xa5000401
set remote memory-write-packet-size 1024
set remote memory-write-packet-size fixed
set remote memory-read-packet-size 1024
set remote memory-read-packet-size fixed
symbol-file main.out
continue
```

Copy these commands into the "Run Commands" window.

The GDB startup commands for the J-Link GDB Server operation shown above require some explanation. If the command line starts with the word "monitor", then that command is a J-Link GDB Server command. Otherwise, it is a legacy GDB command.

J-Link GDB Server commands are described in the document "JLinkGDBServer.pdf" which is in the Segger documentation folder that you downloaded ("c:\Program Files\SEGGER\JLinkARM_V368b\Doc\Manuals\")

GDB commands are described in several books and in the official document that can be downloaded from:
http://dsl.ee.unsw.edu.au/dsl-cdrom/gnutools/doc/gnu-debugger.pdf

First, we have to halt the processor.

```
monitor reset                            # Reset the chip to get to a known state.
```

Next, we set up the JTAG speed

```
monitor speed 30                         # Set JTAG speed to 30 kHz
monitor speed auto                       # Set auto JTAG speed
```

Now we set up the processor's clocks, etc. using the register settings in the lowlevelinit.c function. These are J-Link GDB Server memory write commands used to set the various AT91SAM7S256 clock registers. This guarantees that the processor will be running at full speed when the "continue" command is asserted.

```
monitor long 0xffffff60 0x00320100       # set flash wait state (AT91C_MC_FMR)
monitor long 0xfffffd44 0xa0008000       # watchdog disable (AT91C_WDTC_WDMR)
monitor long 0xfffffc20 0xa0000601       # enable main oscillator (AT91C_PMC_MOR)
monitor sleep 100                        # wait 100 ms
monitor long 0xfffffc2c 0x00480a0e       # set PLL register (AT91C_PMC_PLLR)
monitor sleep 200                        # wait 200 ms
monitor long 0xfffffc30 0x7              # set master clock to PLL (AT91C_PMC_MCKR)
monitor sleep 100                        # wait 100 ms
```

Enable the Reset button in the AT91SAM7S-EK board.

```
monitor long 0xfffffd08 0xa5000401                # enable user reset AT91C_RSTC_RMR
```

Now increase the GDB packet size to 1024. This will have a slight improvement on FLASH debugging as reads of large data structures, etc. may be speeded up. These are legacy GDB commands.

```
set remote memory-write-packet-size 1024       # Setup GDB for faster downloads
set remote memory-write-packet-size fixed      # Setup GDB for faster downloads
set remote memory-read-packet-size 1024        # Setup GDB for faster downloads
set remote memory-read-packet-size fixed       # Setup GDB for faster downloads
```

Now we have to identify the file that has the symbol information. This is a legacy GDB command.

```
symbol-file main.out                           # read the symbol information from main.out
```

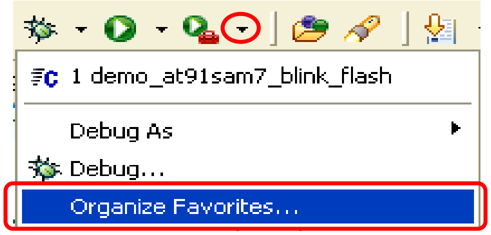
Finally we emit the legacy GDB command "continue". The processor is already halted at the Reset vector and will thus start executing until it hits the breakpoint set at main( ).

```
continue                                       # resume execution from reset vector - will break at main( )
```

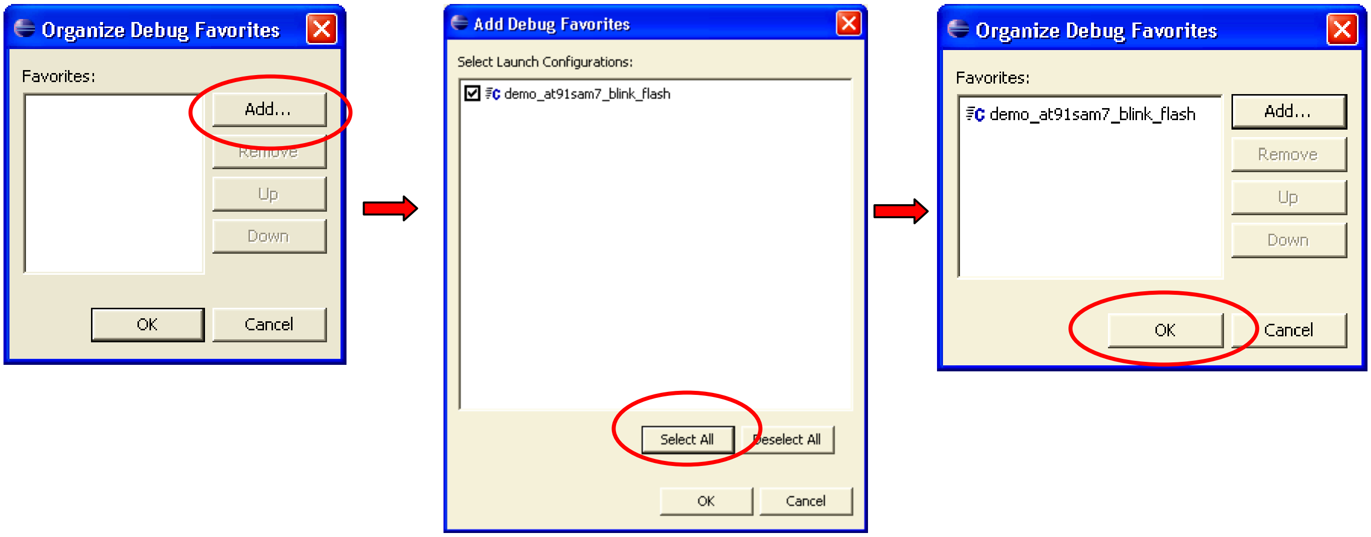
Add the Debug Launch Configuration to the List of Favorites

One final maneuver is to add the “demo_at91sam7_blink_flash” embedded debug launch configuration into the Debug pull-down menu’s list of favorites. This operation is very similar to putting the external tools into the “list of favorites” that you did earlier.

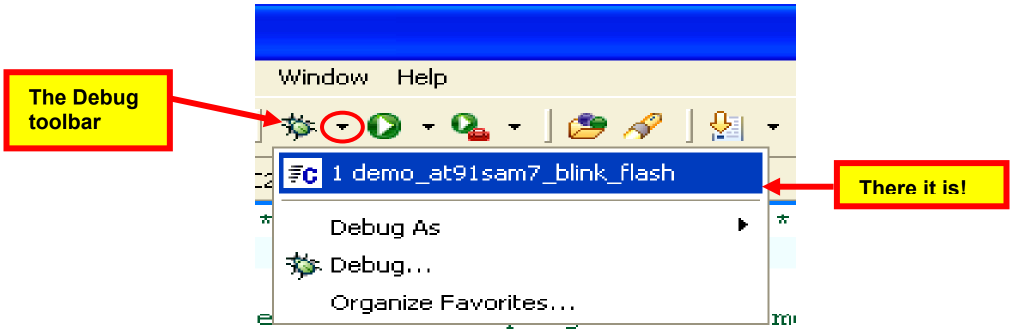
In the toolbar, click on the **down arrowhead** next to the debug symbol and then click “**Organize Favorites…**”



In a sequence similar to other “Organize Favorites” operations that we have already performed, click on “**Add…**” and either checkmark the “**demo_at91sam7_blink_flash**” or click the “**Select All**” button. Finally, click “**OK**” to enter this debug launch configuration into the debugger list of favorites, as shown below.



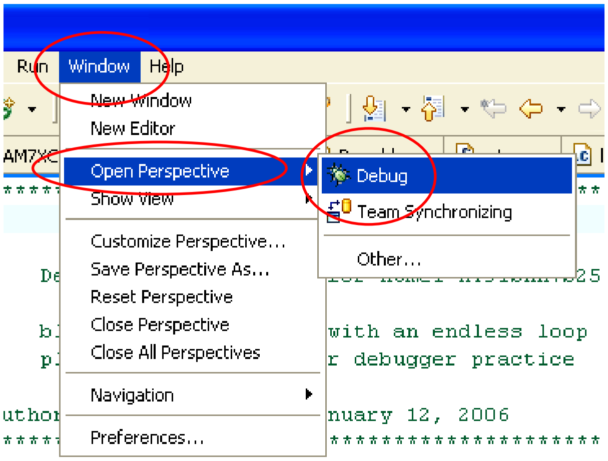
Now when you click on the Debug Toolbar button’s down arrowhead, you will see the “demo_at91SAM7_blink_flash” debug launch configuration installed as a favorite, as shown below.



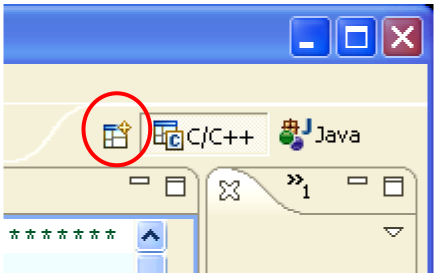
Now everything is in place to debug the project that we loaded into FLASH memory via OpenOCD or SAM-BA.

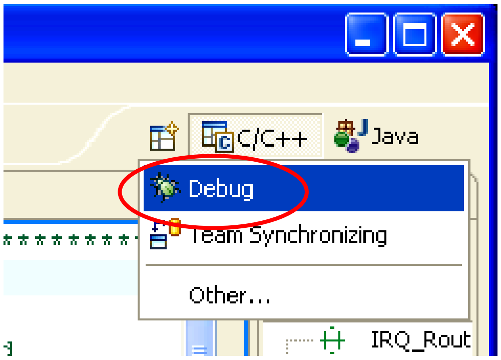
Open the Eclipse Debug Prespective

To debug, we need to switch from the C/C++ perspective to the Debug perspective. The standard way is to click on “**Window – Open Perspective – Debug**” as shown below.

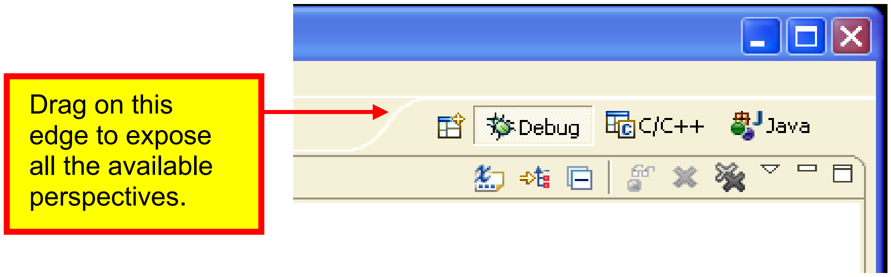


A more convenient way to switch perspectives is to click on the “perspective” buttons at the Eclipse upper-right window location. Click on the “**OpenPerspective**” toolbar button below on the left and then choose “**Debug**” when the other perspectives are displayed.



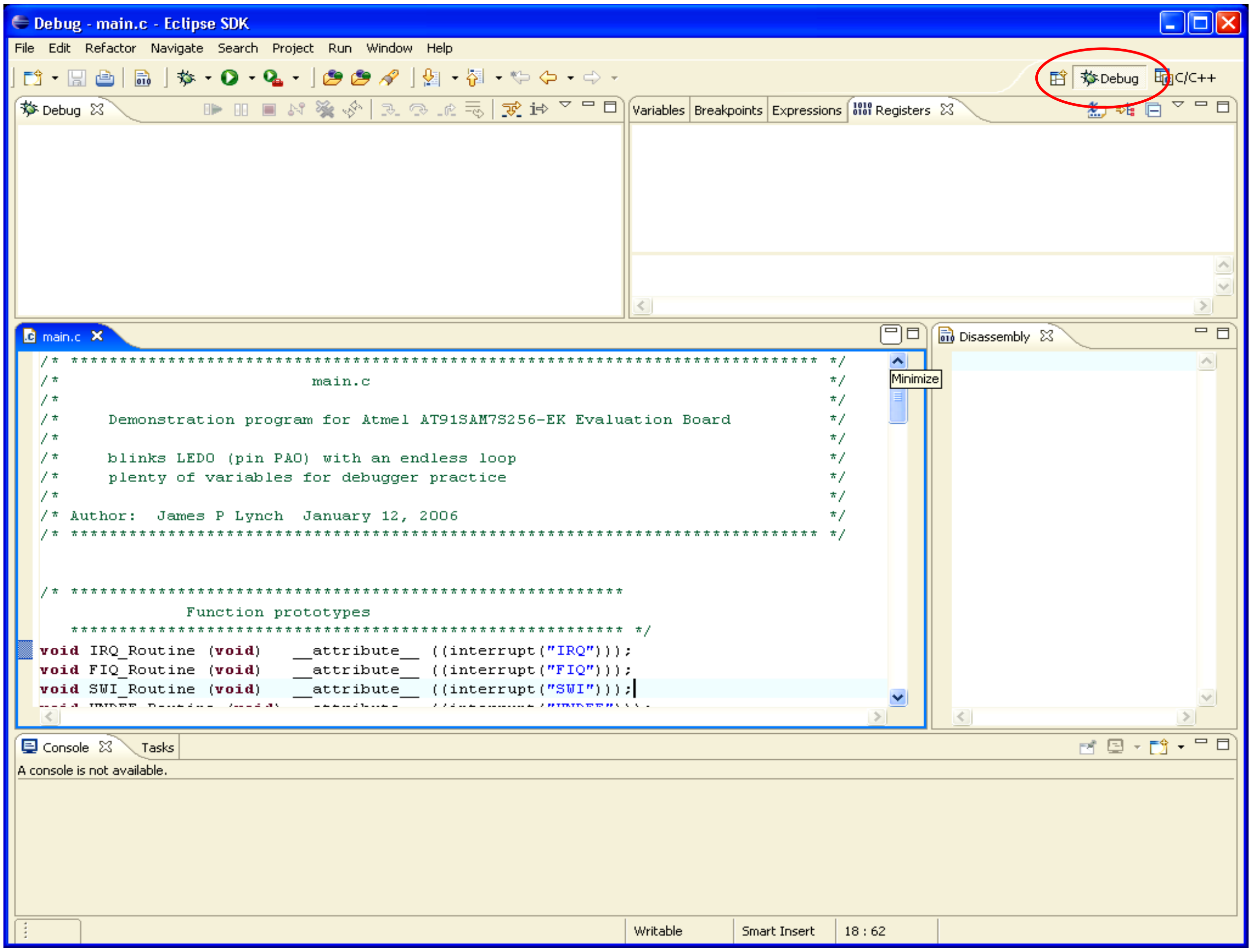


Now we have a “Debug” button as shown below. You may have to drag on the edge to expose all the perspective buttons. You can also right-click on any of the buttons and “Close” them to narrow the display to only the perspectives you are interested in.



Drag on this edge to expose all the available perspectives.

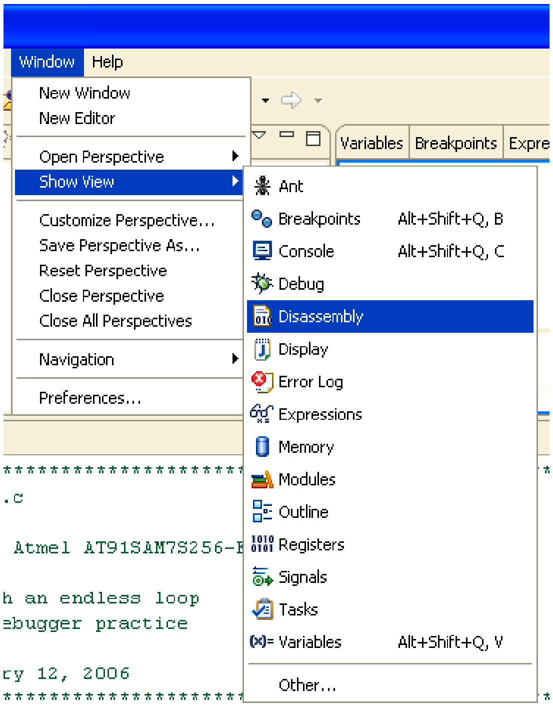
Click on the “Debug” perspective button at the upper-right to open the Debug Perspective display, shown below.



```
/* ************************************************************************** */
/*                              main.c                                         */
/*                                                                            */
/*     Demonstration program for Atmel AT91SAM7S256-EK Evaluation Board        */
/*                                                                            */
/*     blinks LED0 (pin PA0) with an endless loop                              */
/*     plenty of variables for debugger practice                               */
/*                                                                            */
/* Author:  James P Lynch  January 12, 2006                                    */
/* ************************************************************************** */


/* ***************************************************
            Function prototypes
   *************************************************** */
void IRQ_Routine (void)    __attribute__  ((interrupt("IRQ")));
void FIQ_Routine (void)    __attribute__  ((interrupt("FIQ")));
void SWI_Routine (void)    __attribute__  ((interrupt("SWI")));
```

```
A console is not available.
```

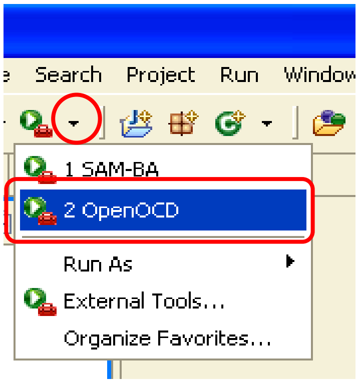
If your display doesn't look exactly like the debug display above, click on "**Window – Show View**" and select any of the missing elements.



Starting OpenOCD

If you have purchased an Olimex or Amontec JTAG debugger, you must have OpenOCD running in the background before starting the Eclipse graphical debugger.

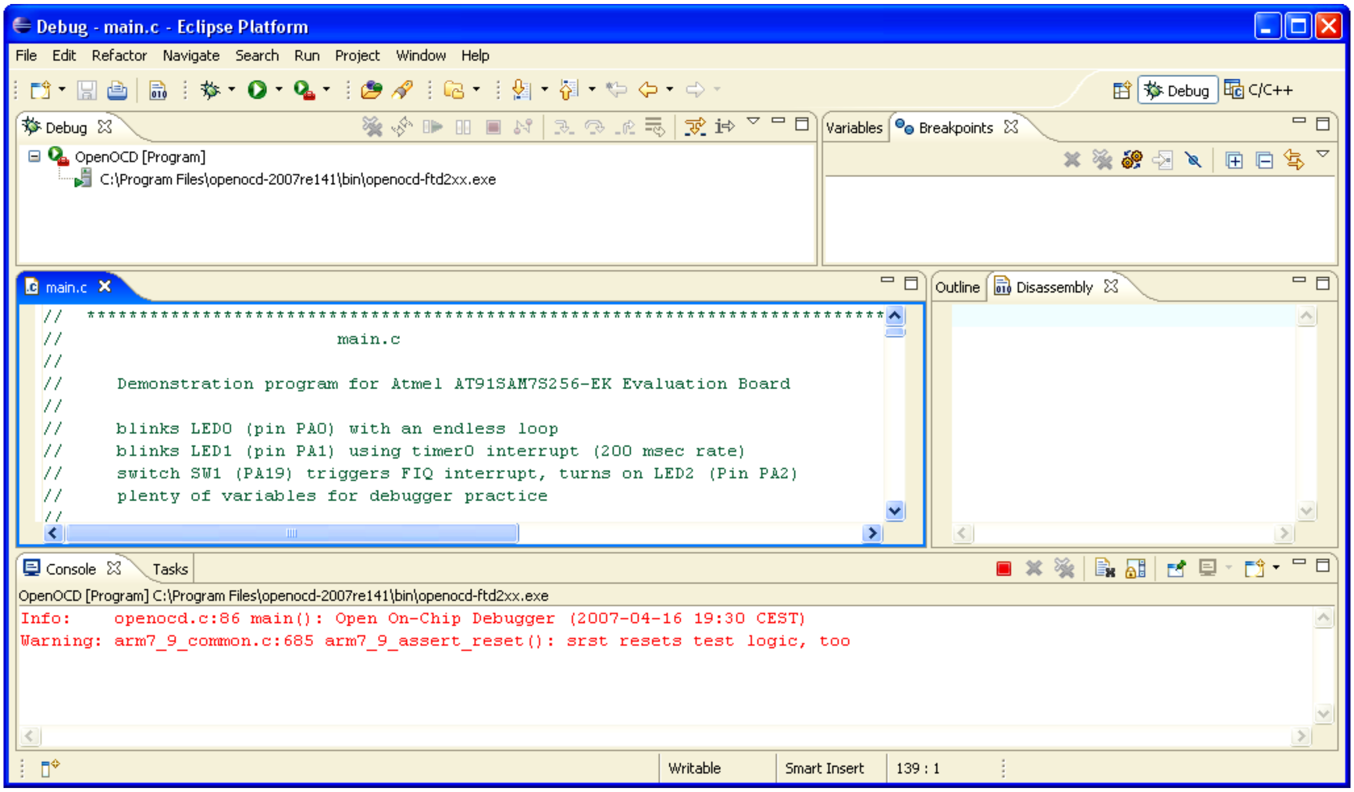
To start OpenOCD, click on the "**External Tools**" toolbar button's down arrowhead and then select "**OpenOCD**". Alternatively, you can click on the "**Run**" pull-down menu and select "**External Tools**" followed by "**OpenOCD**".



Eclipse remembers the last button you selected, so you can usually just click on the red toolbox button itself to start OpenOCD. If you're not sure what "external tool" will be selected, just hover the cursor over the toolbox icon and the "hints" feature will show that "OpenOCD" will be selected.

The debug view will show that OpenOCD is running and the console view shows no errors, just warnings.

Directly below is the Debug perspective just after OpenOCD has started up.



OpenOCD [Program]
C:\Program Files\openocd-2007re141\bin\openocd-ftd2xx.exe

```
//  ****************************************************************************
//                              main.c
//
//      Demonstration program for Atmel AT91SAM7S256-EK Evaluation Board
//
//      blinks LED0 (pin PA0) with an endless loop
//      blinks LED1 (pin PA1) using timer0 interrupt (200 msec rate)
//      switch SW1 (PA19) triggers FIQ interrupt, turns on LED2 (Pin PA2)
//      plenty of variables for debugger practice
//
```

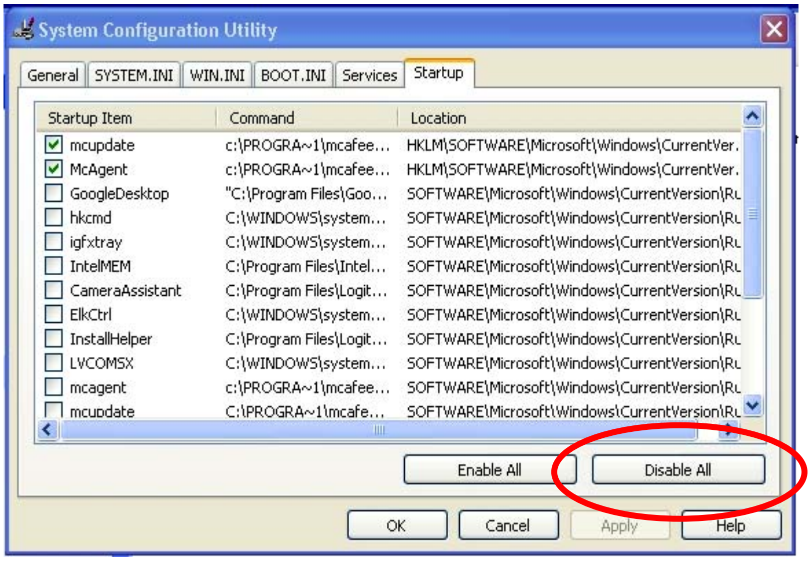
```
OpenOCD [Program] C:\Program Files\openocd-2007re141\bin\openocd-ftd2xx.exe
Info:    openocd.c:86 main(): Open On-Chip Debugger (2007-04-16 19:30 CEST)
Warning: arm7_9_common.c:685 arm7_9_assert_reset(): srst resets test logic, too
```

If for some reason, OpenOCD will not properly start in your system, you can try the following things.

- Cycle power on the target board before starting OpenOCD

- Make sure your computer is not running cpu-intensive applications in the background, such as internet telephone applications (SKYPE for example). The OpenOCD/wiggler system does “bit-banging” on the LPT1 printer port which is fairly low in the Windows priority order.

  For Windows XP users, here is a simple way to get rid of all those background programs. Click “**Start – Help and Support – Use Tools… - System Configuration Utility – Open System Configuration Utility – Startup Tab**”. Click on “**Disable All**”. Windows will ask you to re-boot and the PC will restart with none of the start-up programs running. Use the same procedure to reverse this action.

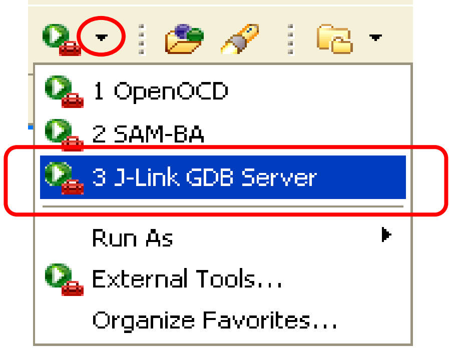


Startup Item	Command	Location
☑ mcupdate	c:\PROGRA~1\mcafee...	HKLM\SOFTWARE\Microsoft\Windows\CurrentVer.
☑ McAgent	c:\PROGRA~1\mcafee...	HKLM\SOFTWARE\Microsoft\Windows\CurrentVer.
□ GoogleDesktop	"C:\Program Files\Goo...	SOFTWARE\Microsoft\Windows\CurrentVersion\Ru
□ hkcmd	C:\WINDOWS\system...	SOFTWARE\Microsoft\Windows\CurrentVersion\Ru
□ igfxtray	C:\WINDOWS\system...	SOFTWARE\Microsoft\Windows\CurrentVersion\Ru
□ IntelMEM	C:\Program Files\Intel...	SOFTWARE\Microsoft\Windows\CurrentVersion\Ru
□ CameraAssistant	C:\Program Files\Logit...	SOFTWARE\Microsoft\Windows\CurrentVersion\Ru
□ ElkCtrl	C:\WINDOWS\system...	SOFTWARE\Microsoft\Windows\CurrentVersion\Ru
□ InstallHelper	C:\Program Files\Logit...	SOFTWARE\Microsoft\Windows\CurrentVersion\Ru
□ LVCOMSX	C:\WINDOWS\system...	SOFTWARE\Microsoft\Windows\CurrentVersion\Ru
□ mcagent	c:\PROGRA~1\mcafee...	SOFTWARE\Microsoft\Windows\CurrentVersion\Ru
□ mcupdate	C:\PROGRA~1\mcafe...	SOFTWARE\Microsoft\Windows\CurrentVersion\Ru

Starting J-Link GDB Server

If you have purchased the Atmel SAM-ICE JTAG debugger, you must have the J-Link GDB Server running in the background before starting the Eclipse graphical debugger.

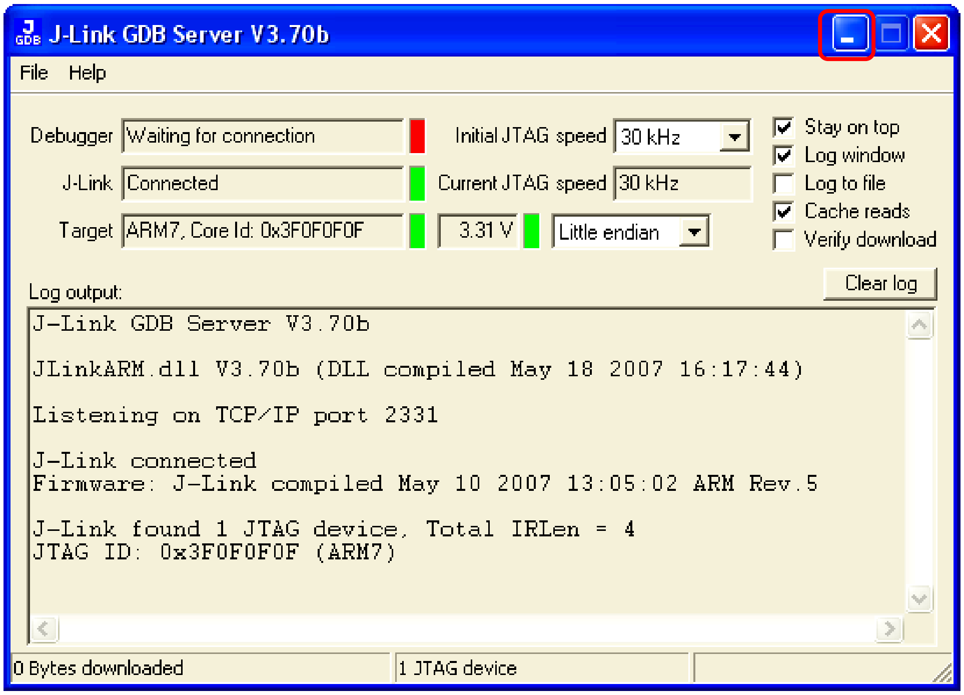
To start J-Link, click on the "**External Tools**" toolbar button's down arrowhead and then select "**J-Link GDB Server**". Alternatively, you can click on the "**Run**" pull-down menu and select "**External Tools**" followed by "**J-Link GDB Server**".



Eclipse remembers the last button you selected, so you can usually just click on the red toolbox button itself to start J-Link. If you're not sure what "external tool" will be selected, just hover the cursor over the toolbox icon and the "hints" feature will show that "J-Link GDB Server" will be selected.

First, a Segger J-Link GDB Server status window will appear as shown below. Notice that the green indicators show that the J-Link GDB Server is connected to your SAM-ICE and the target microprocessor core has been identified. The Debugger status light is indicating red; this is OK since we haven't launched our Eclipse/GDB integrated graphical debugger yet. You should now minimize the Segger status display.

Whatever you do, don't click the ☒ button; that will terminate the J-Link GDB Server!



J-Link GDB Server V3.70b

File Help

Debugger: Waiting for connection 🟥
J-Link: Connected 🟩
Target: ARM7, Core Id: 0x3F0F0F0F 🟩 3.31 V 🟩 Little endian

Initial JTAG speed: 30 kHz
Current JTAG speed: 30 kHz

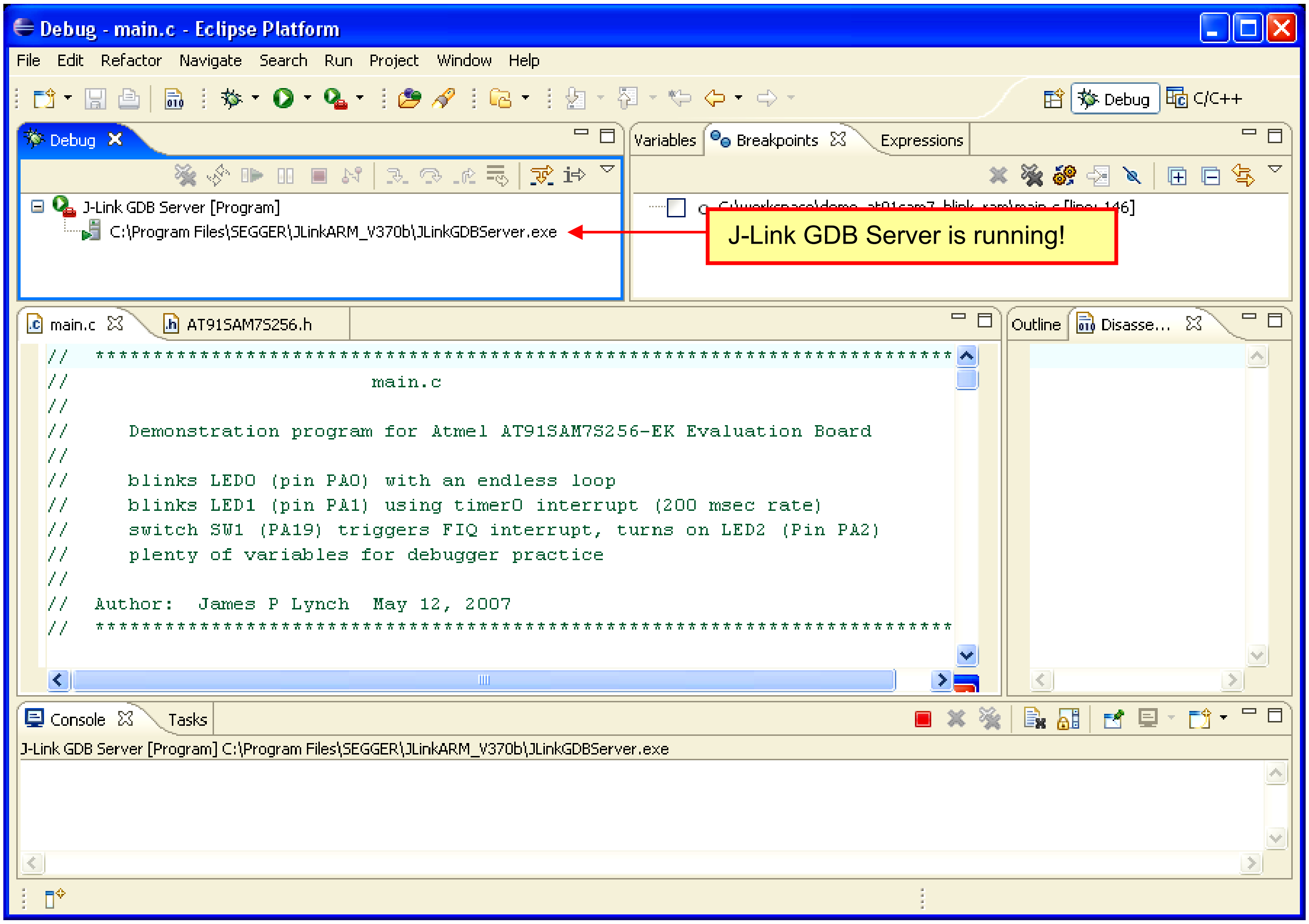
☑ Stay on top
☑ Log window
☐ Log to file
☑ Cache reads
☐ Verify download

Log output:

```
J-Link GDB Server V3.70b

JLinkARM.dll V3.70b (DLL compiled May 18 2007 16:17:44)

Listening on TCP/IP port 2331

J-Link connected
Firmware: J-Link compiled May 10 2007 13:05:02 ARM Rev.5

J-Link found 1 JTAG device, Total IRLen = 4
JTAG ID: 0x3F0F0F0F (ARM7)
```

0 Bytes downloaded | 1 JTAG device

The debug view will show that J-Link GDB Server is running and the console view shows no errors.


Debug - main.c - Eclipse Platform
File Edit Refactor Navigate Search Run Project Window Help
Debug
C/C++
Debug
J-Link GDB Server [Program]
C:\Program Files\SEGGER\JLinkARM_V370b\JLinkGDBServer.exe
Variables
Breakpoints
Expressions
C:\workspace\demo_at91sam7_blink_ram\main.c [line: 146]
J-Link GDB Server is running!
main.c
AT91SAM7S256.h
Outline
Disasse...
Console
Tasks

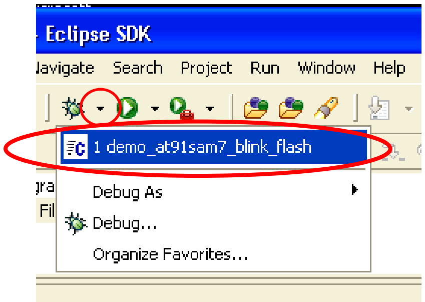
```
//  ***********************************************************************
//                              main.c
//
//     Demonstration program for Atmel AT91SAM7S256-EK Evaluation Board
//
//     blinks LED0 (pin PA0) with an endless loop
//     blinks LED1 (pin PA1) using timer0 interrupt (200 msec rate)
//     switch SW1 (PA19) triggers FIQ interrupt, turns on LED2 (Pin PA2)
//     plenty of variables for debugger practice
//
//  Author:  James P Lynch  May 12, 2007
//  ***********************************************************************
```

```
J-Link GDB Server [Program] C:\Program Files\SEGGER\JLinkARM_V370b\JLinkGDBServer.exe
```

Start the Eclipse Debugger

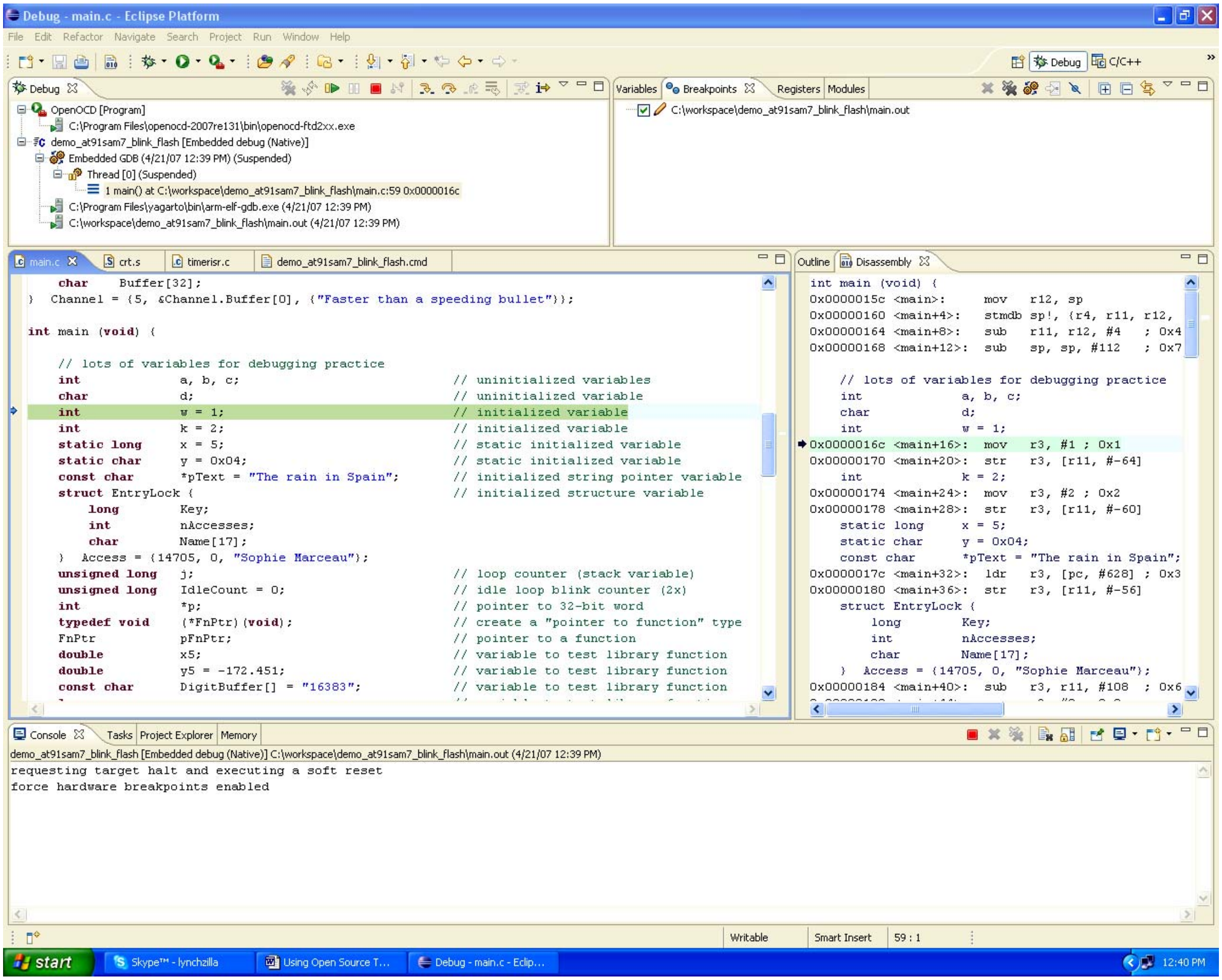
To start the Eclipse debugger, click on the "**Debug**" toolbar button's down arrowhead and select the debug launch configuration "**demo_at91sam7_blink_flash**" as shown below.

Alternatively, you can start the debugger by clicking on "**Run – Debug…**" and then select the "**demo_at91sam7_blink_flash**" embedded launch configuration and then click "**debug**". Obviously, the debug toolbar button is more convenient.


Eclipse SDK
Navigate Search Project Run Window Help
1 demo_at91sam7_blink_flash
Debug As
Debug…
Organize Favorites…

There's not a lot of difference in the behavior of the Eclipse/GDB integrated graphical debugger whether you run it from OpenOCD or the J-Link GDB Server.

Eclipse Debugger Startup - OpenOCD



```
OpenOCD [Program]
    C:\Program Files\openocd-2007re131\bin\openocd-ftd2xx.exe
demo_at91sam7_blink_flash [Embedded debug (Native)]
    Embedded GDB (4/21/07 12:39 PM) (Suspended)
        Thread [0] (Suspended)
            1 main() at C:\workspace\demo_at91sam7_blink_flash\main.c:59 0x0000016c
    C:\Program Files\yagarto\bin\arm-elf-gdb.exe (4/21/07 12:39 PM)
    C:\workspace\demo_at91sam7_blink_flash\main.out (4/21/07 12:39 PM)
```

☑ C:\workspace\demo_at91sam7_blink_flash\main.out

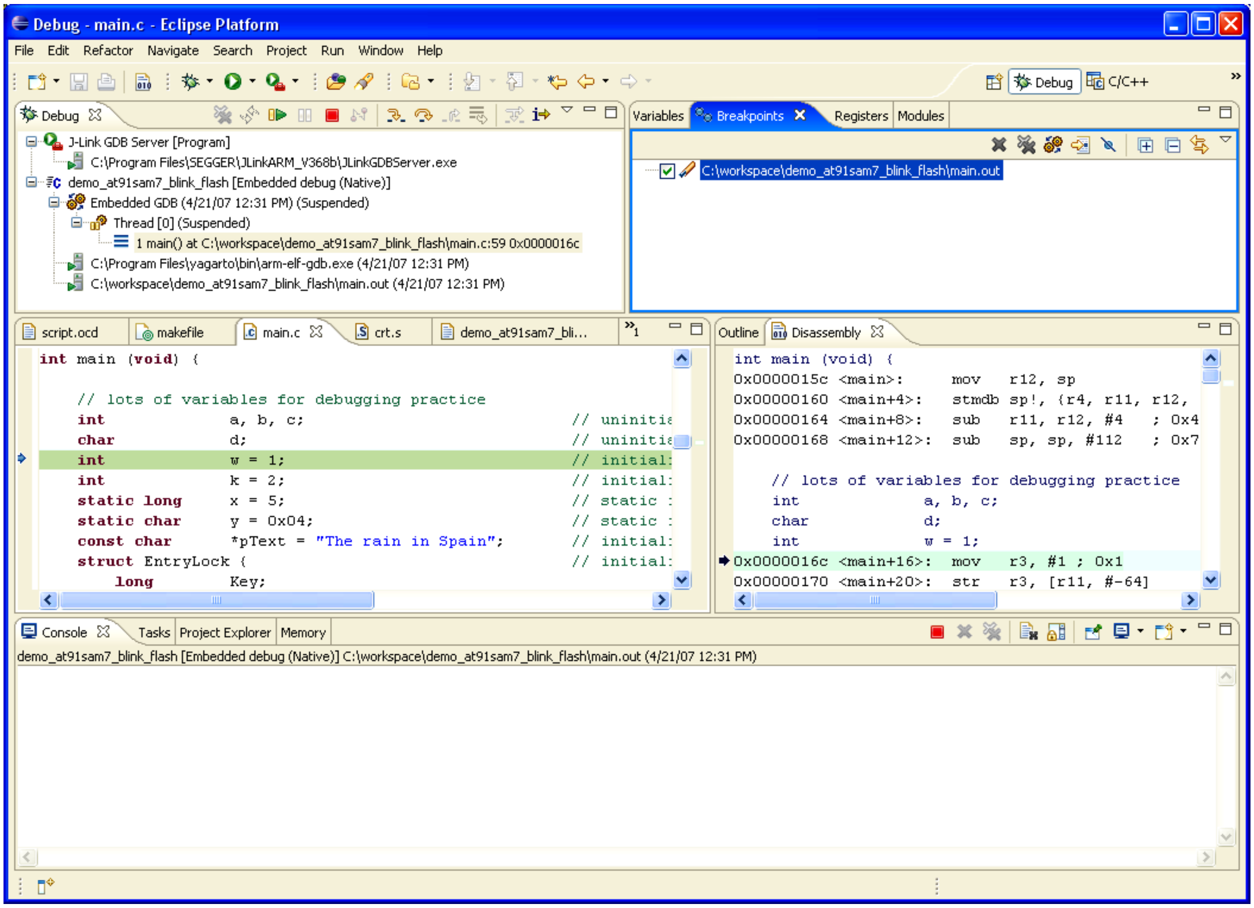
```
    char     Buffer[32];
}  Channel = {5, &Channel.Buffer[0], {"Faster than a speeding bullet"}};

int main (void) {

    // lots of variables for debugging practice
    int               a, b, c;                              // uninitialized variables
    char              d;                                    // uninitialized variable
    int               w = 1;                                // initialized variable
    int               k = 2;                                // initialized variable
    static long       x = 5;                                // static initialized variable
    static char       y = 0x04;                             // static initialized variable
    const char        *pText = "The rain in Spain";         // initialized string pointer variable
    struct EntryLock {                                      // initialized structure variable
        long          Key;
        int           nAccesses;
        char          Name[17];
    }  Access = {14705, 0, "Sophie Marceau"};
    unsigned long     j;                                    // loop counter (stack variable)
    unsigned long     IdleCount = 0;                        // idle loop blink counter (2x)
    int               *p;                                   // pointer to 32-bit word
    typedef void      (*FnPtr)(void);                       // create a "pointer to function" type
    FnPtr             pFnPtr;                               // pointer to a function
    double            x5;                                   // variable to test library function
    double            y5 = -172.451;                        // variable to test library function
    const char        DigitBuffer[] = "16383";              // variable to test library function
```

```
int main (void) {
0x0000015c <main>:     mov   r12, sp
0x00000160 <main+4>:   stmdb sp!, {r4, r11, r12,
0x00000164 <main+8>:   sub   r11, r12, #4   ; 0x4
0x00000168 <main+12>:  sub   sp, sp, #112   ; 0x7

    // lots of variables for debugging practice
    int               a, b, c;
    char              d;
    int               w = 1;
0x0000016c <main+16>:  mov   r3, #1 ; 0x1
0x00000170 <main+20>:  str   r3, [r11, #-64]
    int               k = 2;
0x00000174 <main+24>:  mov   r3, #2 ; 0x2
0x00000178 <main+28>:  str   r3, [r11, #-60]
    static long       x = 5;
    static char       y = 0x04;
    const char        *pText = "The rain in Spain";
0x0000017c <main+32>:  ldr   r3, [pc, #628] ; 0x3
0x00000180 <main+36>:  str   r3, [r11, #-56]
    struct EntryLock {
        long          Key;
        int           nAccesses;
        char          Name[17];
    }  Access = {14705, 0, "Sophie Marceau"};
0x00000184 <main+40>:  sub   r3, r11, #108  ; 0x6
```

demo_at91sam7_blink_flash [Embedded debug (Native)] C:\workspace\demo_at91sam7_blink_flash\main.out (4/21/07 12:39 PM)

```
requesting target halt and executing a soft reset
force hardware breakpoints enabled
```

Eclipse Debugger Startup – J-Link GDB Server



```
J-Link GDB Server [Program]
    C:\Program Files\SEGGER\JLinkARM_V368b\JLinkGDBServer.exe
demo_at91sam7_blink_flash [Embedded debug (Native)]
    Embedded GDB (4/21/07 12:31 PM) (Suspended)
        Thread [0] (Suspended)
            1 main() at C:\workspace\demo_at91sam7_blink_flash\main.c:59 0x0000016c
    C:\Program Files\yagarto\bin\arm-elf-gdb.exe (4/21/07 12:31 PM)
    C:\workspace\demo_at91sam7_blink_flash\main.out (4/21/07 12:31 PM)
```

☑ C:\workspace\demo_at91sam7_blink_flash\main.out

```
int main (void) {

    // lots of variables for debugging practice
    int                 a, b, c;                                // uninitia
    char                d;                                      // uninitia
    int                 w = 1;                                  // initial:
    int                 k = 2;                                  // initial:
    static long         x = 5;                                  // static :
    static char         y = 0x04;                               // static :
    const char          *pText = "The rain in Spain";           // initial:
    struct EntryLock {                                          // initial:
        long            Key;
```

```
int main (void) {
0x0000015c <main>:     mov   r12, sp
0x00000160 <main+4>:   stmdb sp!, {r4, r11, r12,
0x00000164 <main+8>:   sub   r11, r12, #4    ; 0x4
0x00000168 <main+12>:  sub   sp, sp, #112    ; 0x7

    // lots of variables for debugging practice
    int                 a, b, c;
    char                d;
    int                 w = 1;
0x0000016c <main+16>:  mov   r3, #1 ; 0x1
0x00000170 <main+20>:  str   r3, [r11, #-64]
```

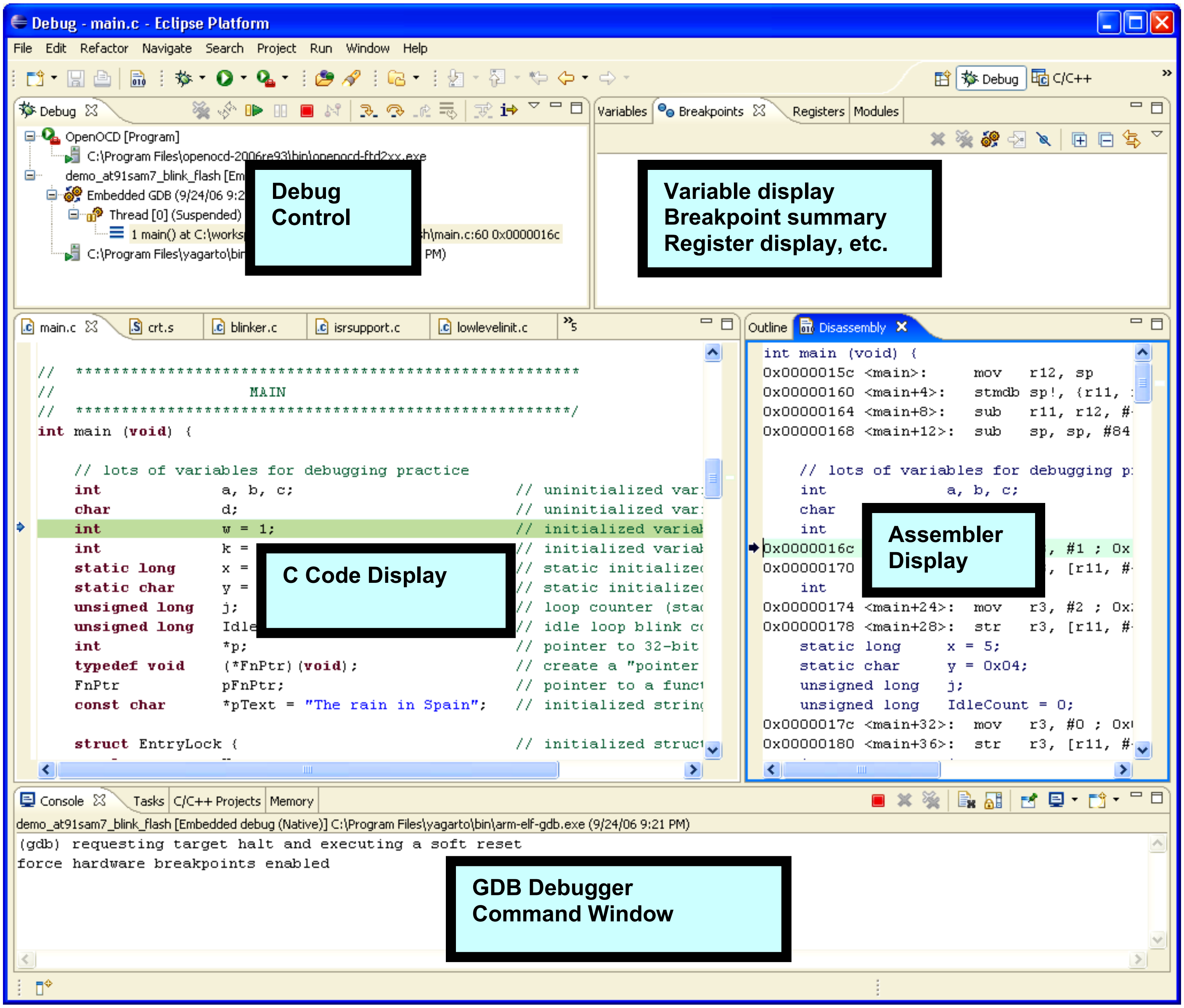
demo_at91sam7_blink_flash [Embedded debug (Native)] C:\workspace\demo_at91sam7_blink_flash\main.out (4/21/07 12:31 PM)

In both examples, Eclipse started the application and stopped at the main( ) entry point. Specifically, it stopped on line 59 of the source file main.c.

If the Eclipse debugger doesn't connect properly, then there will be a progress bar at the bottom right status line that runs forever. In this case, terminate everything and power cycle the target board again.

Components of the DEBUG Perspective

Before operating the Eclipse debugger, let’s review the components of the Debug perspective.



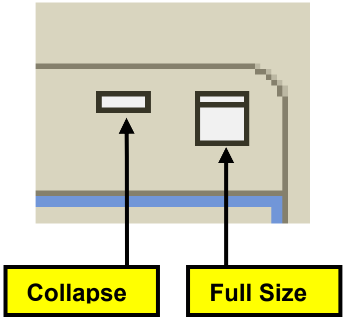
```
OpenOCD [Program]
    C:\Program Files\openocd-2006re93\bin\openocd-ftd2xx.exe
demo_at91sam7_blink_flash [Em
    Embedded GDB (9/24/06 9:2
        Thread [0] (Suspended)
            1 main() at C:\works                    h\main.c:60 0x0000016c
    C:\Program Files\yagarto\bi                     PM)
```

```
//   ************************************************************
//                          MAIN
//   ************************************************************/
int main (void) {

    // lots of variables for debugging practice
    int                 a, b, c;                     // uninitialized var:
    char                d;                           // uninitialized var:
    int                 w = 1;                       // initialized variab
    int                 k =                          // initialized variab
    static long         x =                          // static initialized
    static char         y =                          // static initialized
    unsigned long       j;                           // loop counter (stac
    unsigned long       Idle                         // idle loop blink co
    int                 *p;                          // pointer to 32-bit
    typedef void        (*FnPtr)(void);              // create a "pointer
    FnPtr               pFnPtr;                      // pointer to a funct
    const char          *pText = "The rain in Spain";   // initialized string

    struct EntryLock {                               // initialized struct
```

```
int main (void) {
0x0000015c <main>:      mov    r12, sp
0x00000160 <main+4>:    stmdb  sp!, {r11, :
0x00000164 <main+8>:    sub    r11, r12, #
0x00000168 <main+12>:   sub    sp, sp, #84

    // lots of variables for debugging p:
    int                 a, b, c;
    char
    int
0x0000016c                              , #1 ; 0x
0x00000170                              , [r11, #
    int
0x00000174 <main+24>:   mov    r3, #2 ; 0x:
0x00000178 <main+28>:   str    r3, [r11, #
    static long         x = 5;
    static char         y = 0x04;
    unsigned long       j;
    unsigned long       IdleCount = 0;
0x0000017c <main+32>:   mov    r3, #0 ; 0x
0x00000180 <main+36>:   str    r3, [r11, #
```

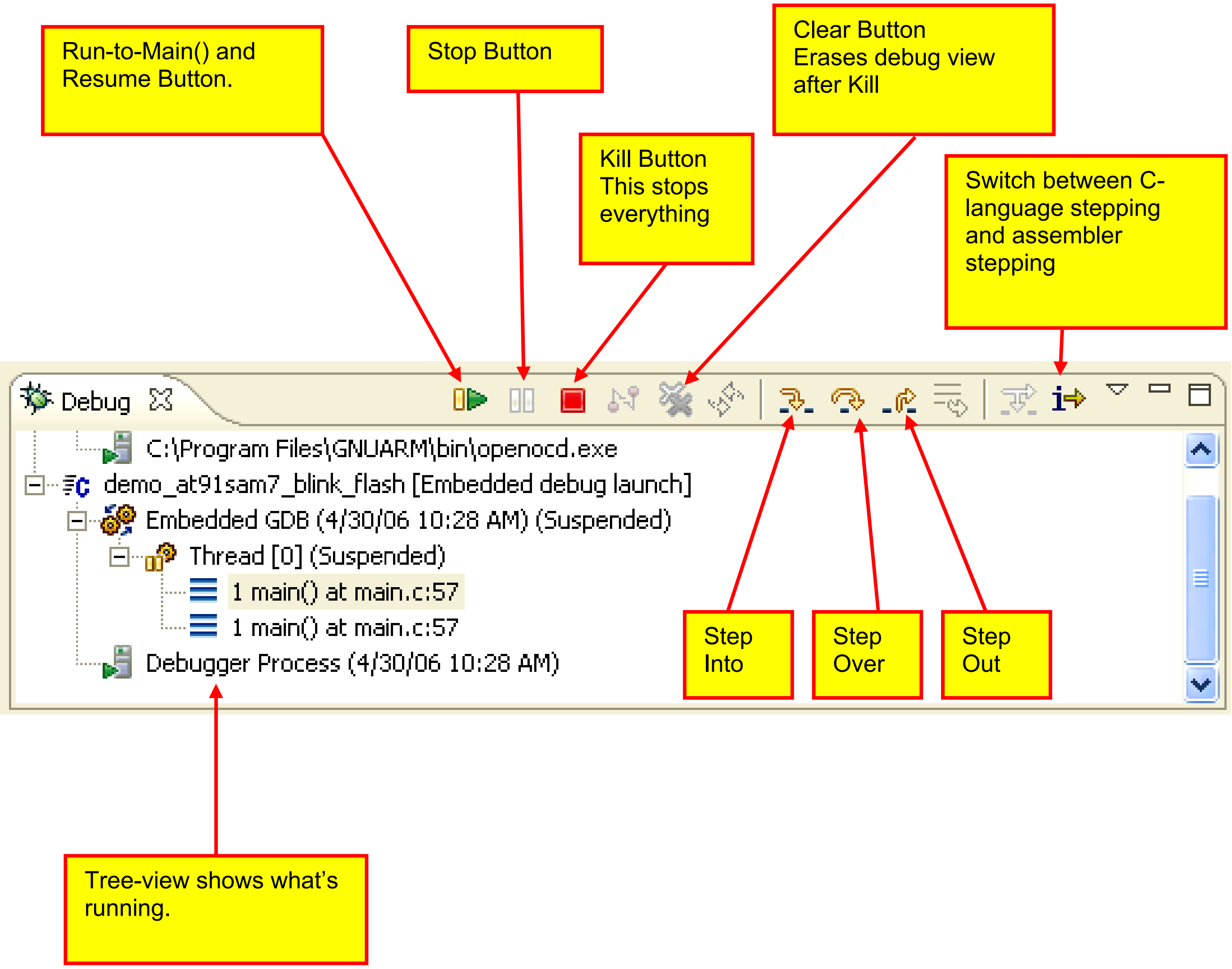
```
demo_at91sam7_blink_flash [Embedded debug (Native)] C:\Program Files\yagarto\bin\arm-elf-gdb.exe (9/24/06 9:21 PM)
(gdb) requesting target halt and executing a soft reset
force hardware breakpoints enabled
```

While this may be obvious to most, you can expand to full screen and then collapse any of the windows in the Debug perspective by clicking on the “maximize” and “minimize” buttons at the top right corner of each window.



Debug Control

The Debug view should be on display at all times. It has the **Run**, **Stop** and **Step** buttons. The tree-structured display shows what is running; in this case it's the **OpenOCD** utility and our application, shown as **Thread[0]**.



Run-to-Main() and Resume Button.
Stop Button
Kill Button This stops everything
Clear Button Erases debug view after Kill
Switch between C-language stepping and assembler stepping
Debug
Step Into
Step Over
Step Out
Tree-view shows what's running.

```
C:\Program Files\GNUARM\bin\openocd.exe
demo_at91sam7_blink_flash [Embedded debug launch]
  Embedded GDB (4/30/06 10:28 AM) (Suspended)
    Thread [0] (Suspended)
      1 main() at main.c:57
      1 main() at main.c:57
  Debugger Process (4/30/06 10:28 AM)
```

Notes:

- When you resume execution by clicking on the **Resume/Continue** button, many of the buttons are "grayed out." Click on "**Thread[0]**" to highlight it and the buttons will re-appear. This is due to the possibility of multiple threads running simultaneously and you must choose which thread to pause or step. In our ARM development system, we only have one thread.

- You can only set two breakpoints at a time when debugging FLASH. If you are stepping, it behooves you to have no breakpoints set since Eclipse needs one of the hardware breakpoints for single-stepping.

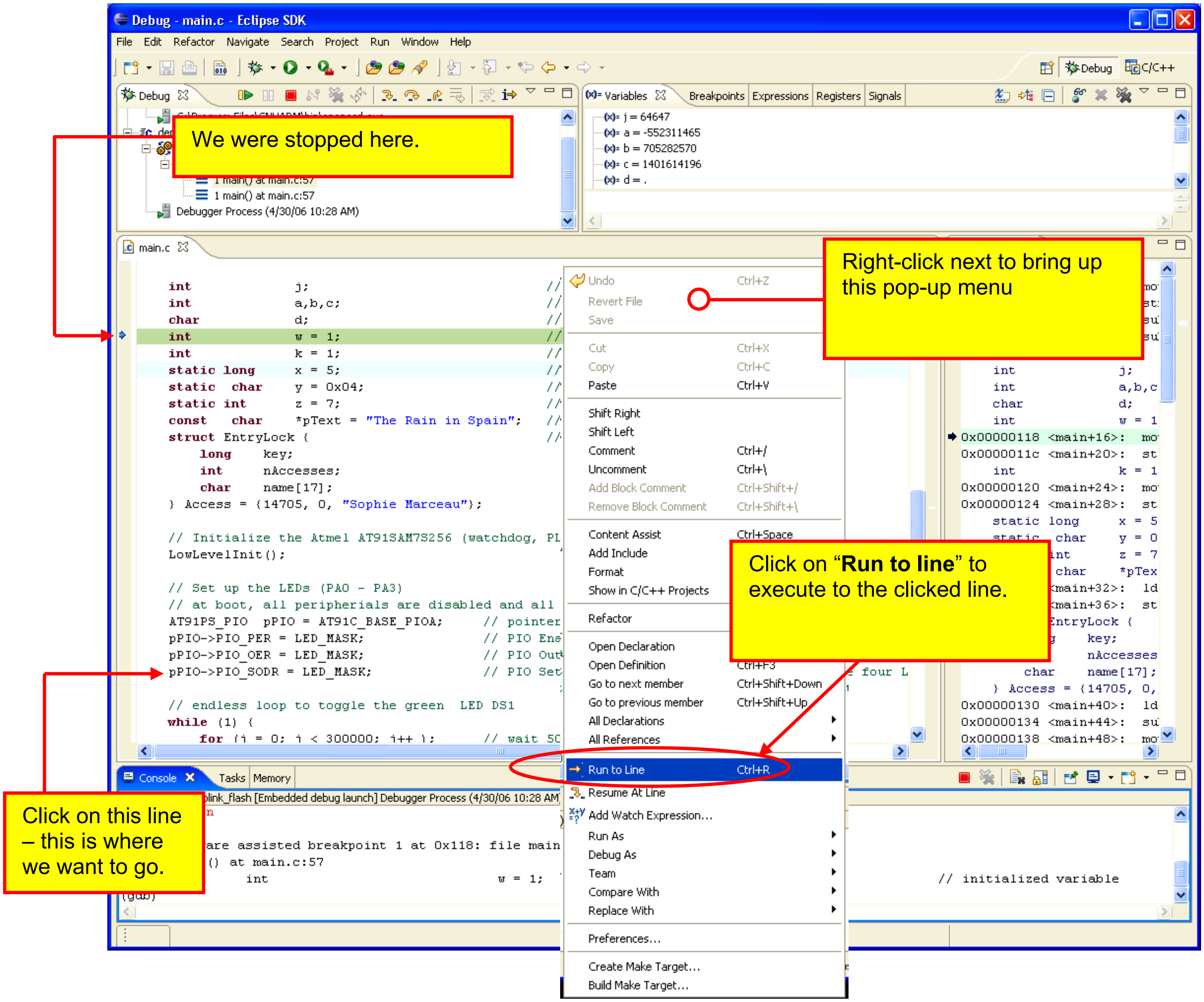
- If you re-compile your application, you must stop the debugger and OpenOCD or J-Link GDB Server, re-build and burn the main.bin file into FLASH using the OpenOCD FLASH programming facility or the Atmel SAM-BA flash programming utility.

Run and Stop with the Right-Click Menu

The easiest method of running is to employ the right-click menu. In the example below, the blue arrowhead cursor indicates where the program is currently stopped - just after main( ).

To go to the **pPIO->PIO_SODR = LED_MASK;** statement several lines away, click on the line where you want to go (this should highlight the line and place the cursor there).

Now **right click** on that line. Notice that the rather large pop-up menu has a "**Run to Line**" option.



We were stopped here.

Right-click next to bring up this pop-up menu

Click on "**Run to line**" to execute to the clicked line.

Click on this line – this is where we want to go.

```
1 main() at main.c:57
1 main() at main.c:57
Debugger Process (4/30/06 10:28 AM)
```

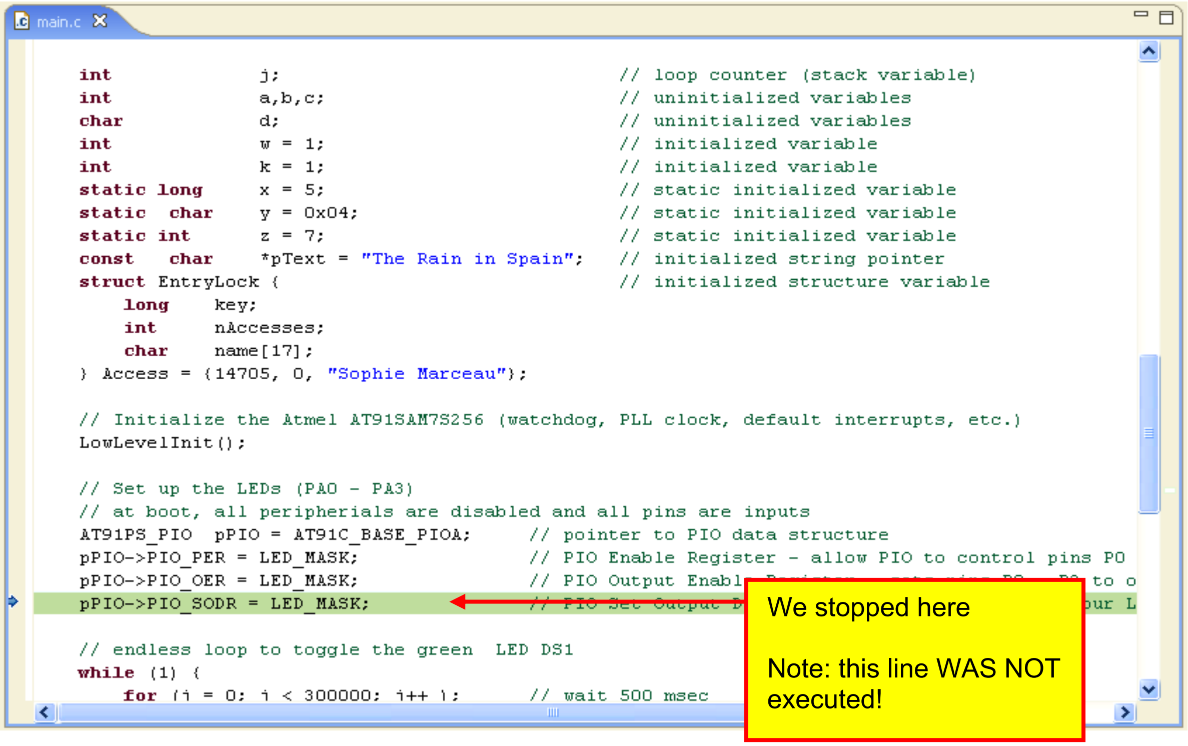
```
j = 64647
a = -552311465
b = 705282570
c = 1401614196
d = .
```

```
    int             j;                                   //
    int             a,b,c;                               //
    char            d;                                   //
    int             w = 1;                               //
    int             k = 1;                               //
    static long     x = 5;                               //
    static  char    y = 0x04;                            //
    static int      z = 7;                               //
    const   char    *pText = "The Rain in Spain";        //
    struct EntryLock {                                   //
        long    key;
        int     nAccesses;
        char    name[17];
    } Access = {14705, 0, "Sophie Marceau"};

    // Initialize the Atmel AT91SAM7S256 (watchdog, PL
    LowLevelInit();

    // Set up the LEDs (PA0 - PA3)
    // at boot, all peripherials are disabled and all
    AT91PS_PIO  pPIO = AT91C_BASE_PIOA;      // pointer
    pPIO->PIO_PER = LED_MASK;                // PIO Ena
    pPIO->PIO_OER = LED_MASK;                // PIO Out
    pPIO->PIO_SODR = LED_MASK;               // PIO Set

    // endless loop to toggle the green  LED DS1
    while (1) {
        for (j = 0; j < 300000; j++ );       // wait 50
```

```
    int             j;
    int             a,b,c
    char            d;
    int             w = 1
0x00000118 <main+16>:  mo
0x0000011c <main+20>:  st
    int             k = 1
0x00000120 <main+24>:  mo
0x00000124 <main+28>:  st
    static long     x = 5
    static  char    y = 0
            int     z = 7
            char    *pTex
0x00000128 <main+32>:  ld
0x0000012c <main+36>:  st
        EntryLock {
        g       key;
                nAccesses
        char    name[17];
    } Access = {14705, 0,
0x00000130 <main+40>:  ld
0x00000134 <main+44>:  su
0x00000138 <main+48>:  mo
```

```
blink_flash [Embedded debug launch] Debugger Process (4/30/06 10:28 AM)
n

are assisted breakpoint 1 at 0x118: file main
() at main.c:57
     int                    w = 1;                       // initialized variable
(gdb)
```

When you click on the "**Run to line**" choice, the program will execute to the line the cursor resides on and then stop (N.B. it will not execute the line).



```
int              j;                               // loop counter (stack variable)
int              a,b,c;                           // uninitialized variables
char             d;                               // uninitialized variables
int              w = 1;                           // initialized variable
int              k = 1;                           // initialized variable
static long      x = 5;                           // static initialized variable
static  char     y = 0x04;                        // static initialized variable
static int       z = 7;                           // static initialized variable
const   char     *pText = "The Rain in Spain";    // initialized string pointer
struct EntryLock {                                // initialized structure variable
    long     key;
    int      nAccesses;
    char     name[17];
} Access = {14705, 0, "Sophie Marceau"};

// Initialize the Atmel AT91SAM7S256 (watchdog, PLL clock, default interrupts, etc.)
LowLevelInit();

// Set up the LEDs (PA0 - PA3)
// at boot, all peripherials are disabled and all pins are inputs
AT91PS_PIO  pPIO = AT91C_BASE_PIOA;      // pointer to PIO data structure
pPIO->PIO_PER = LED_MASK;                // PIO Enable Register - allow PIO to control pins P0
pPIO->PIO_OER = LED_MASK;                // PIO Output Enabl                             to o
pPIO->PIO_SODR = LED_MASK;               // PIO Set Output D                            our L

// endless loop to toggle the green  LED DS1
while (1) {
    for (j = 0; j < 300000; j++ );       // wait 500 msec
```

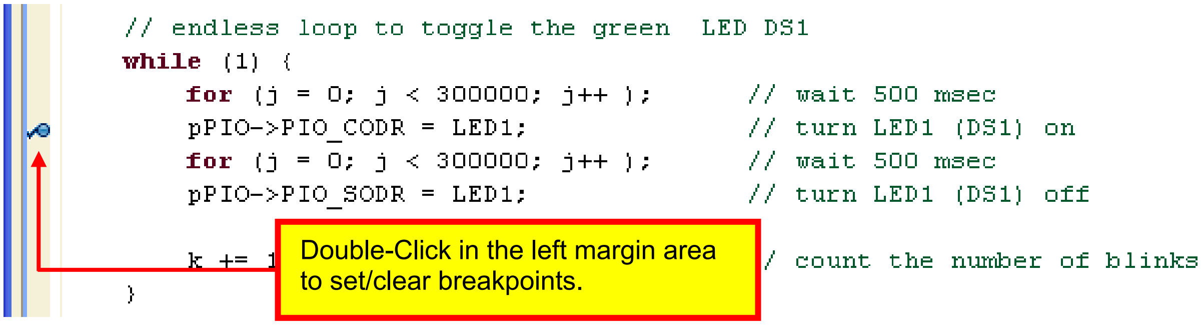
We stopped here

Note: this line WAS NOT executed!

You can right-click the “**Resume at Line**” choice to continue execution from that point. If there are no other breakpoints set, then the Blink application will start blinking continuously.

Setting a Breakpoint

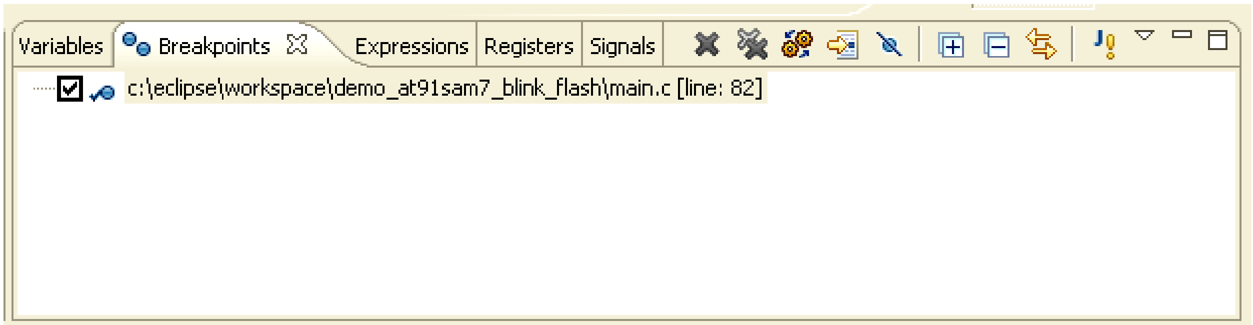
Setting a breakpoint is very simple; just double-click on the far left edge of the line. Double-clicking on the same spot will remove it.



```
// endless loop to toggle the green  LED DS1
while (1) {
    for (j = 0; j < 300000; j++ );       // wait 500 msec
    pPIO->PIO_CODR = LED1;               // turn LED1 (DS1) on
    for (j = 0; j < 300000; j++ );       // wait 500 msec
    pPIO->PIO_SODR = LED1;               // turn LED1 (DS1) off

    k += 1                               / count the number of blinks
}
```

Double-Click in the left margin area to set/clear breakpoints.

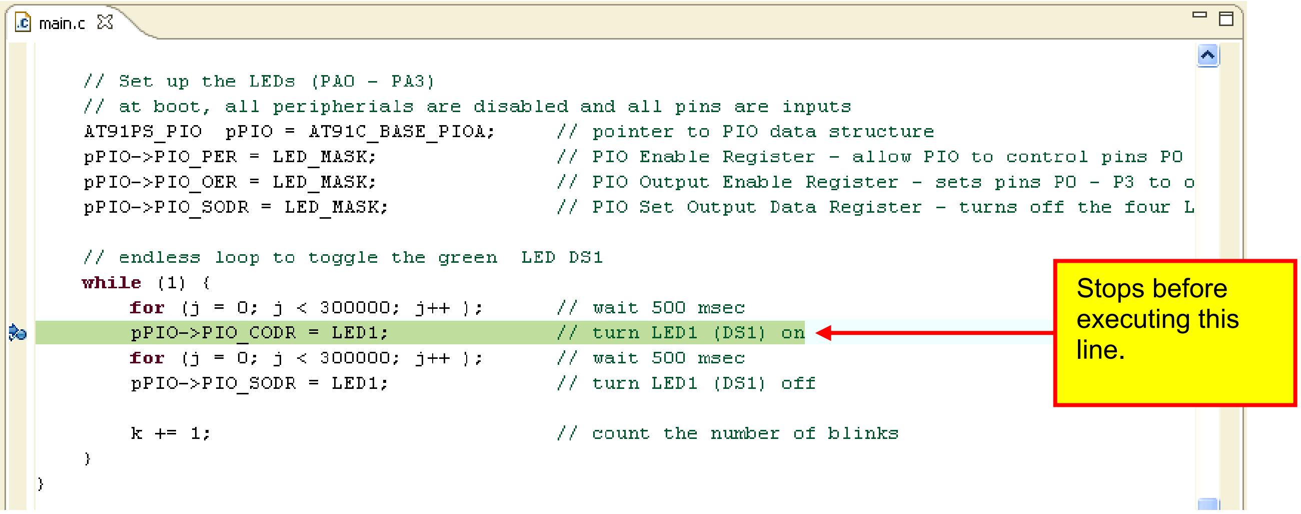
Note in the upper right “Breakpoint Summary” pane, the new breakpoint at line 82 has been indicated, as shown below.



☑ c:\eclipse\workspace\demo_at91sam7_blink_flash\main.c [line: 82]

Now click on the "**Run/Continue**" button in the Debug view.

Assuming that this is the only breakpoint set, the program will execute to the breakpoint line and stop.



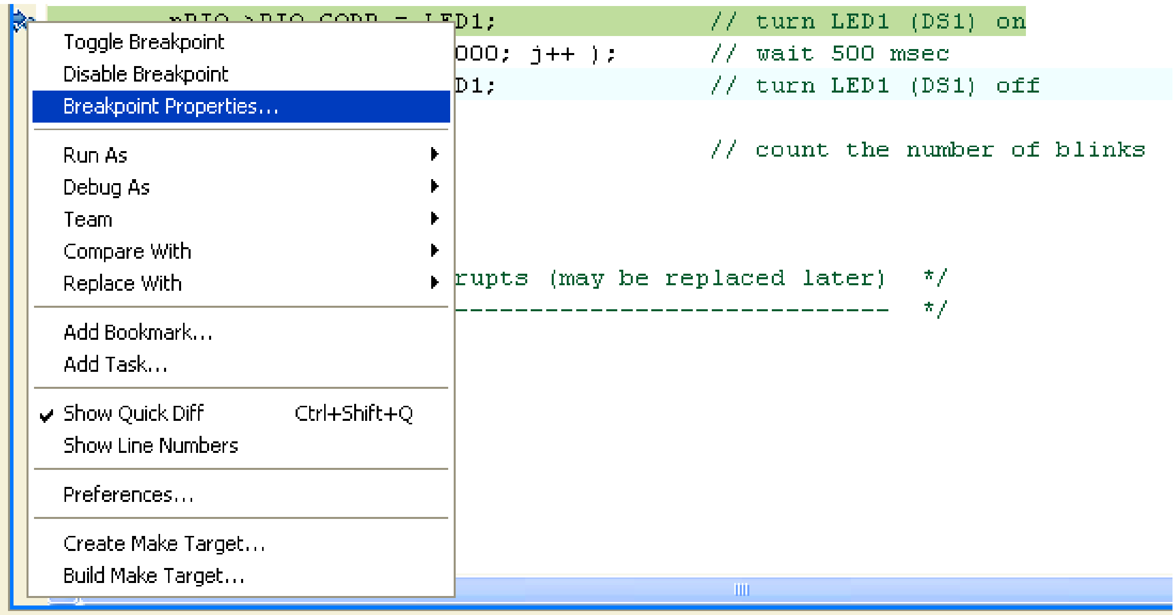
```
    // Set up the LEDs (PA0 - PA3)
    // at boot, all peripherials are disabled and all pins are inputs
    AT91PS_PIO  pPIO = AT91C_BASE_PIOA;      // pointer to PIO data structure
    pPIO->PIO_PER = LED_MASK;                // PIO Enable Register - allow PIO to control pins P0
    pPIO->PIO_OER = LED_MASK;                // PIO Output Enable Register - sets pins P0 - P3 to o
    pPIO->PIO_SODR = LED_MASK;               // PIO Set Output Data Register - turns off the four L

    // endless loop to toggle the green  LED DS1
    while (1) {
        for (j = 0; j < 300000; j++ );       // wait 500 msec
        pPIO->PIO_CODR = LED1;               // turn LED1 (DS1) on
        for (j = 0; j < 300000; j++ );       // wait 500 msec
        pPIO->PIO_SODR = LED1;               // turn LED1 (DS1) off

        k += 1;                              // count the number of blinks
    }
}
```

Stops before executing this line.

Since this is a FLASH application and breakpoints are "hardware" breakpoints, you are limited to **only two breakpoints specified at a time**. Setting more than two breakpoints will cause the debugger to malfunction!

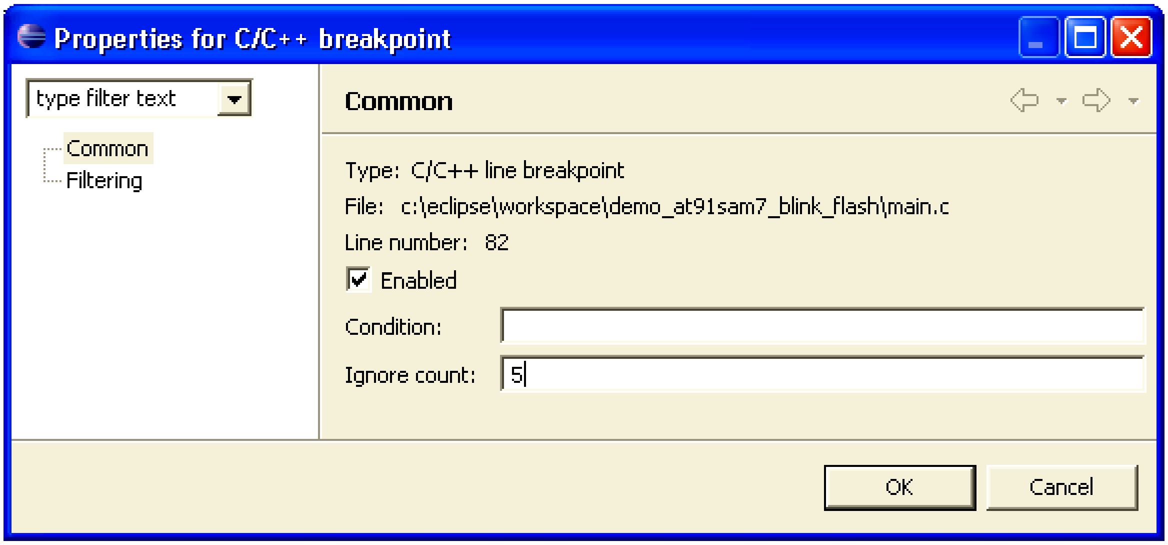
The breakpoints can be more complex. For example, to ignore the breakpoint 5 times and then stop, right-click on the breakpoint symbol on the far left.

This brings up the pop-up menu below; click on "**Breakpoint Properties …**".



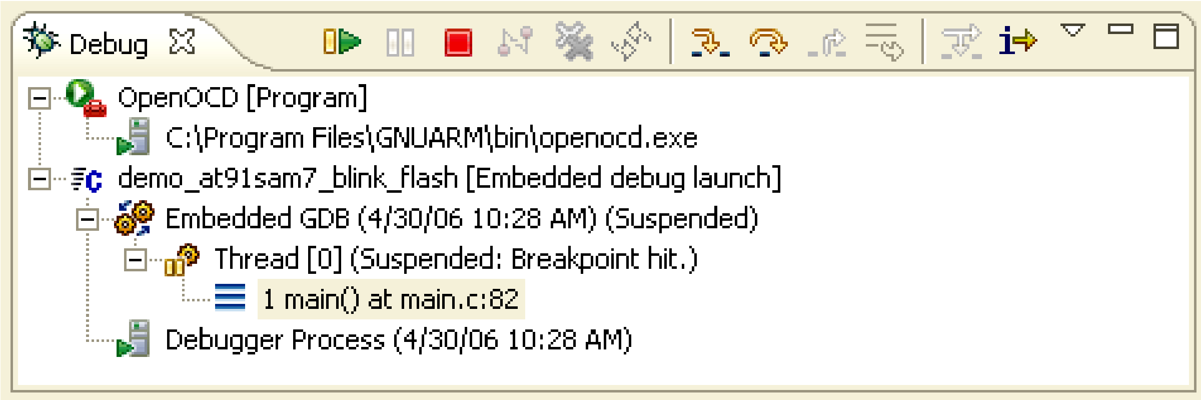
```
pPIO->PIO_CODR = LED1;              // turn LED1 (DS1) on
000; j++ );                         // wait 500 msec
D1;                                 // turn LED1 (DS1) off

                                    // count the number of blinks

rupts (may be replaced later)   */
------------------------------   */
```

In the “**Properties for C/C++ breakpoint**” window, set the **Ignore Count** to 5. This means that the debugger will ignore the first five times it encounters the breakpoint and then stop.



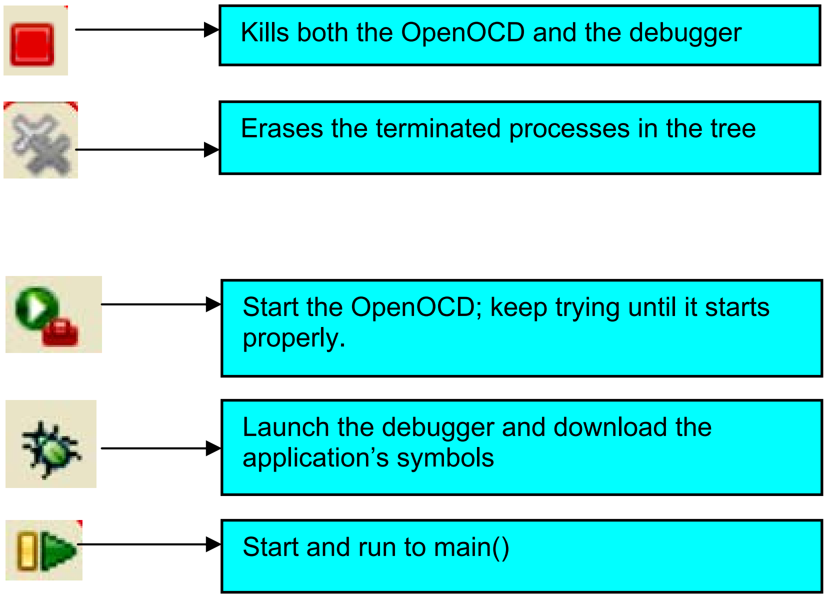
Type: C/C++ line breakpoint
File: c:\eclipse\workspace\demo_at91sam7_blink_flash\main.c
Line number: 82
☑ Enabled
Condition:
Ignore count: 5

To test this setup, we must terminate and re-launch the debugger.



OpenOCD [Program]
C:\Program Files\GNUARM\bin\openocd.exe
demo_at91sam7_blink_flash [Embedded debug launch]
Embedded GDB (4/30/06 10:28 AM) (Suspended)
Thread [0] (Suspended: Breakpoint hit.)
1 main() at main.c:82
Debugger Process (4/30/06 10:28 AM)

Get used to this sequence:



Kills both the OpenOCD and the debugger

Erases the terminated processes in the tree

Start the OpenOCD; keep trying until it starts properly.

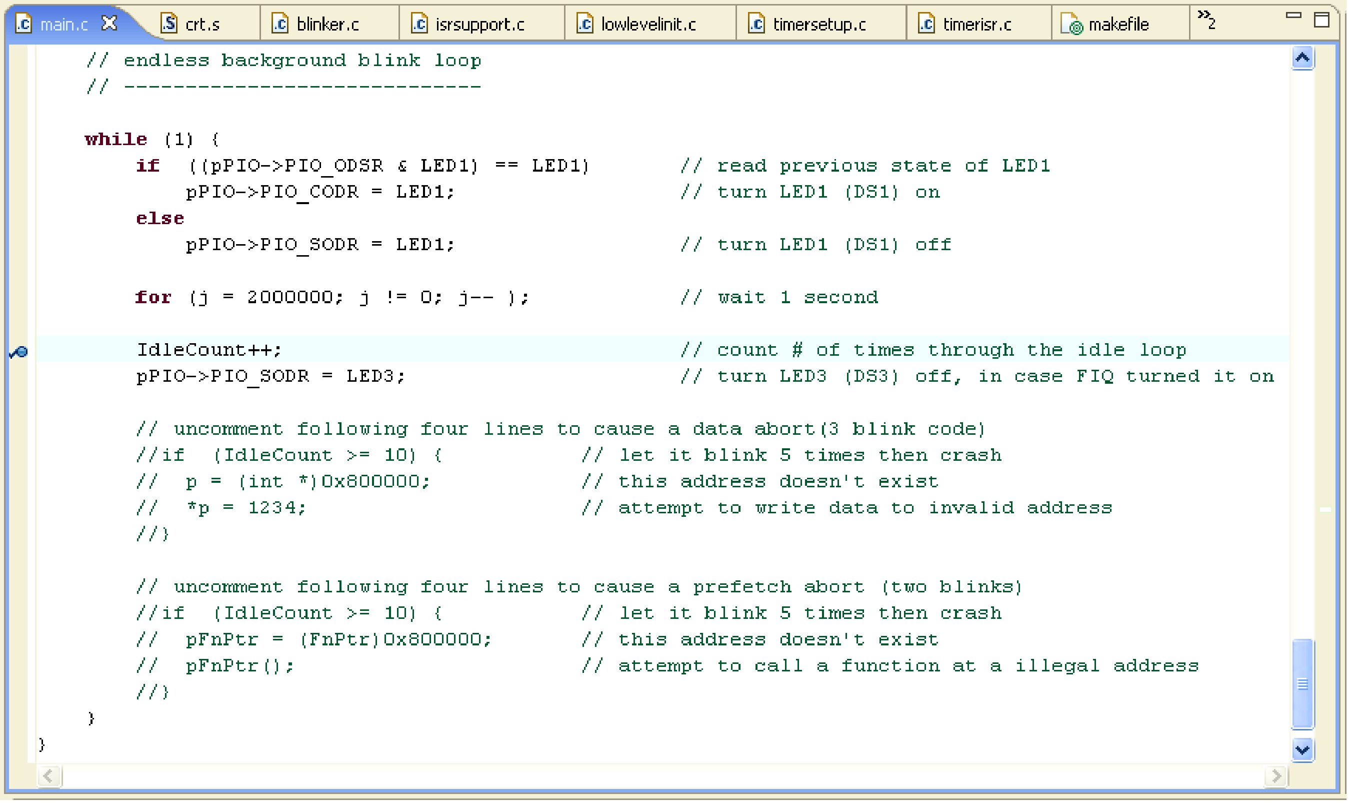
Launch the debugger and download the application’s symbols

Start and run to main()

Now when you hit the **Run/Continue** button again, the program will blink 5 times and stop. Don't expect this feature to run in real-time. Each time the breakpoint is encountered the debugger will automatically continue until the "ignore" count is reached. This involves quite a bit of debugger communication at a very slow baud rate especially if you're using a "wiggler". The "wiggler" works by bit-banging the PC's parallel LPT1 port; this limits the JTAG speed to less than 500 kHz.

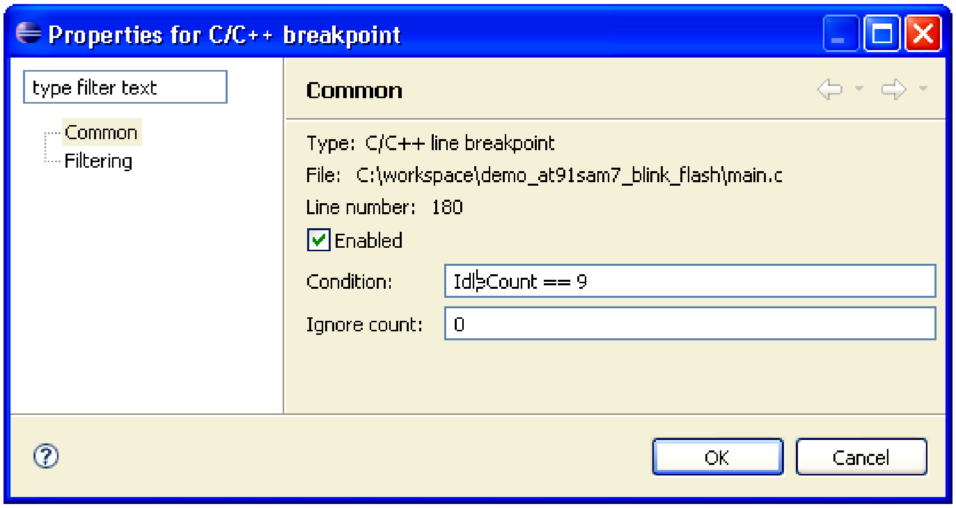
In addition to specifying a "ignore" count, the breakpoint can be made **conditional** on an expression. The general idea is that you set a breakpoint and then specify a conditional expression that must be met before the debugger will stop on the specified source line.

In this example, there's a line in the blink loop that increments a variable "IdleCount". Double-click on that line to set a breakpoint.



```
    // endless background blink loop
    // -----------------------------

    while (1) {
        if  ((pPIO->PIO_ODSR & LED1) == LED1)       // read previous state of LED1
            pPIO->PIO_CODR = LED1;                  // turn LED1 (DS1) on
        else
            pPIO->PIO_SODR = LED1;                  // turn LED1 (DS1) off

        for (j = 2000000; j != 0; j-- );            // wait 1 second

        IdleCount++;                                // count # of times through the idle loop
        pPIO->PIO_SODR = LED3;                      // turn LED3 (DS3) off, in case FIQ turned it on

        // uncomment following four lines to cause a data abort(3 blink code)
        //if  (IdleCount >= 10) {           // let it blink 5 times then crash
        //  p = (int *)0x800000;            // this address doesn't exist
        //  *p = 1234;                      // attempt to write data to invalid address
        //}

        // uncomment following four lines to cause a prefetch abort (two blinks)
        //if  (IdleCount >= 10) {           // let it blink 5 times then crash
        //  pFnPtr = (FnPtr)0x800000;       // this address doesn't exist
        //  pFnPtr();                       // attempt to call a function at a illegal address
        //}
    }
}
```

Right click on the breakpoint symbol and select "**Breakpoint Properties**". In the Breakpoint Properties window, set the condition text box to "**IdleCount == 9**".



Type: C/C++ line breakpoint

File: C:\workspace\demo_at91sam7_blink_flash\main.c

Line number: 180

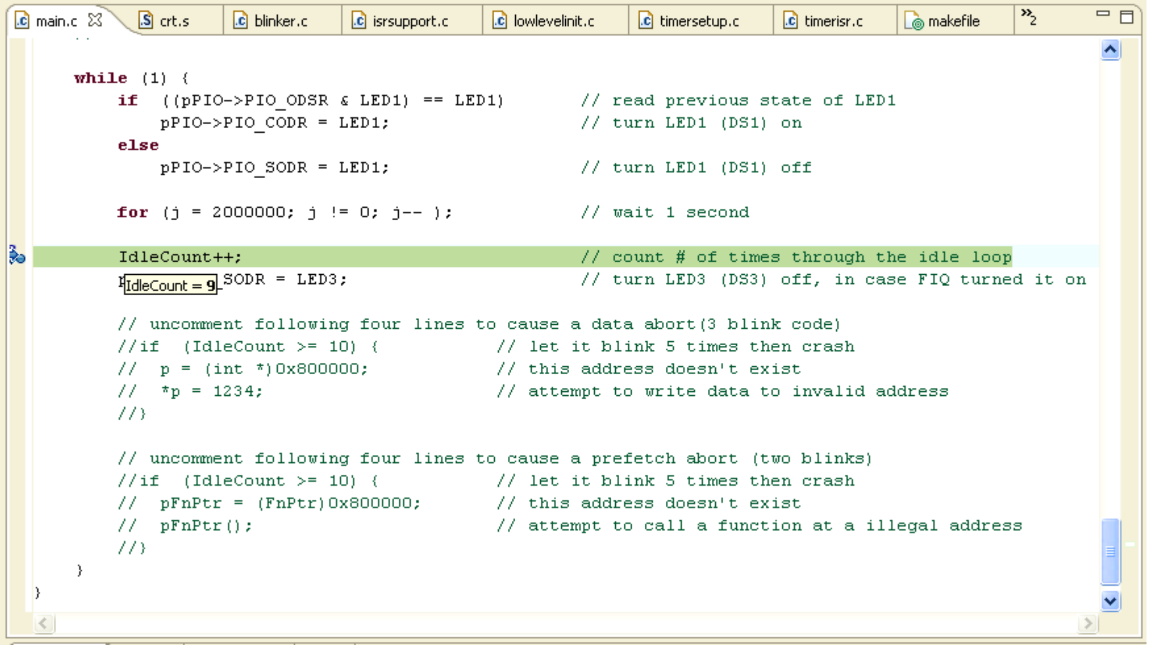
☑ Enabled

Condition: IdleCount == 9

Ignore count: 0

If you need to restart the debugger, you need to **kill the OpenOCD and the Debugger and then restart both**; as specified above. This is necessary for this release of **CDT** because the "**Restart**" button appears inoperative. The advantage is that you don't have to change the Eclipse perspective – just stay in the Debug perspective.

Start the application and it will stop on the breakpoint line (this will take a long time, 9 seconds on my Dell computer). If you park the cursor over the variable IdleCount after the program has suspended on the breakpoint, it will display that the current value is 9.



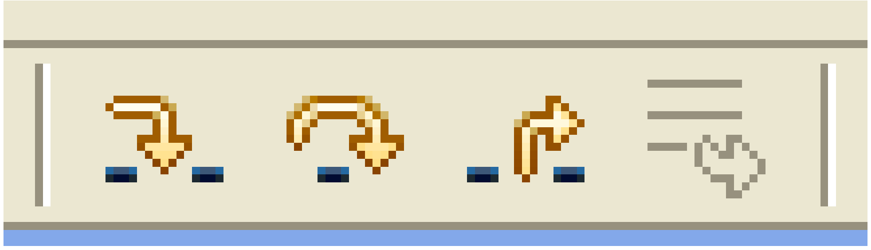
```
    while (1) {
        if  ((pPIO->PIO_ODSR & LED1) == LED1)       // read previous state of LED1
            pPIO->PIO_CODR = LED1;                  // turn LED1 (DS1) on
        else
            pPIO->PIO_SODR = LED1;                  // turn LED1 (DS1) off

        for (j = 2000000; j != 0; j-- );            // wait 1 second

        IdleCount++;                                // count # of times through the idle loop
        pIdleCount = 9 SODR = LED3;                 // turn LED3 (DS3) off, in case FIQ turned it on

        // uncomment following four lines to cause a data abort(3 blink code)
        //if  (IdleCount >= 10) {               // let it blink 5 times then crash
        //  p = (int *)0x800000;                // this address doesn't exist
        //  *p = 1234;                          // attempt to write data to invalid address
        //}

        // uncomment following four lines to cause a prefetch abort (two blinks)
        //if  (IdleCount >= 10) {               // let it blink 5 times then crash
        //  pFnPtr = (FnPtr)0x800000;           // this address doesn't exist
        //  pFnPtr();                           // attempt to call a function at a illegal address
        //}
    }
}
```

If you specify that it should break when IdleCount == 50000, you will essentially wait forever. The way this works, the debugger breaks on the selected source line every pass through that source line and then queries via JTAG for the current value of the variable IdleCount. When IdleCount==50000, the debugger will stop. Obviously, that requires a lot of serial communication at a very slow baud rate. Still, you may find some use for this feature.

In the Breakpoint Summary view, you can see all the breakpoints you have created and the right-click menu lets you change the properties, remove or disable any of the breakpoints, etc.

Single Stepping

Single-stepping is the single most useful feature in any debugging environment. The debug view has three buttons to support this.



Step Into   Step Over   Step Out Of

Step Into

If the cursor is at a function call, this will step **into** the function. It will stop at the first instruction inside the function.

If cursor is on any other line, this will execute one instruction.

Step Over

If the cursor is at a function call, this will step **over** the function. It will execute the entire function and stop on the next instruction after the function call.

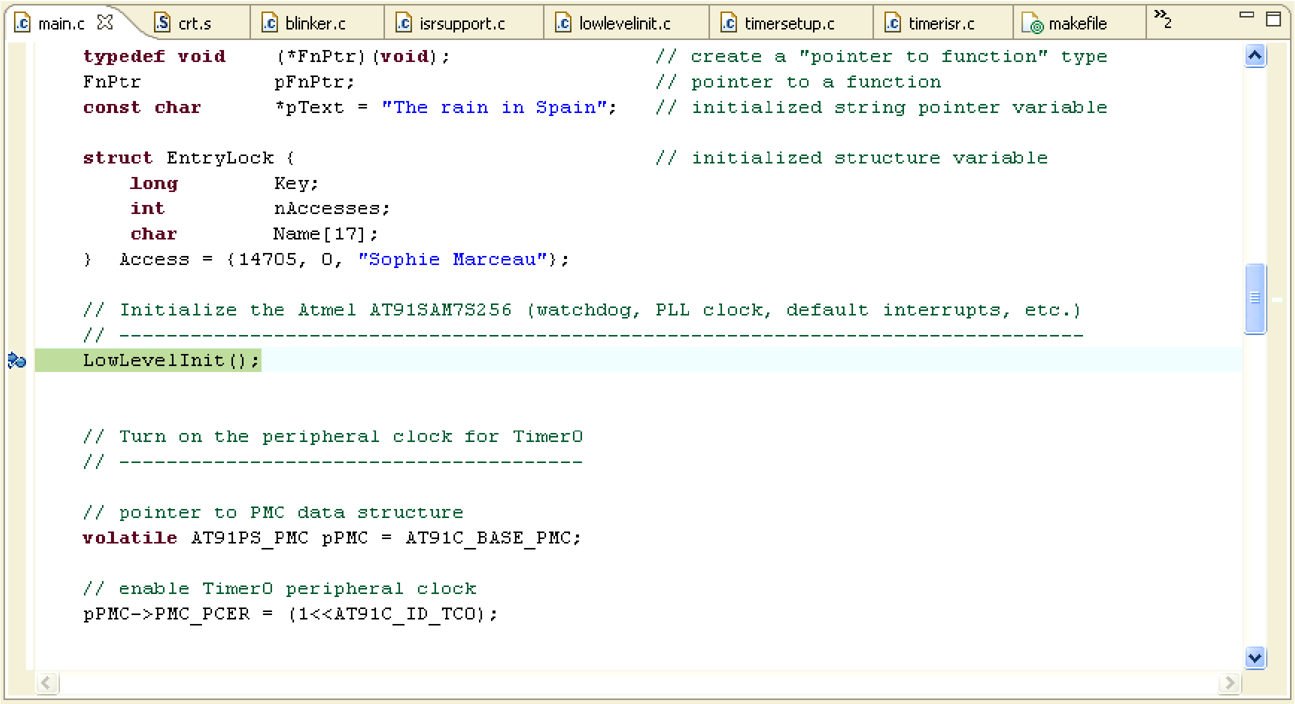
If cursor is on any other line, this will execute one instruction

Step Out Of

If the cursor is within a function, this will execute the remaining instructions in the function and stop on the next instruction after the function call.

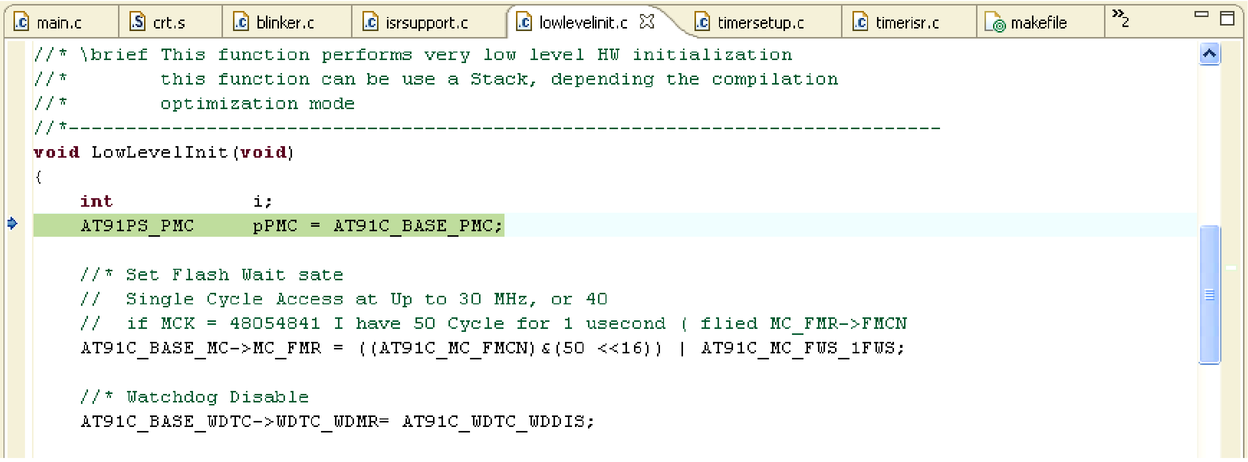
This button will be "grayed-out" if cursor is not within a function.

As a simple example, restart the debugger and set a breakpoint on the line that calls the **LowLevelInit( )** function. Hit the **Start** button to go to that breakpoint.



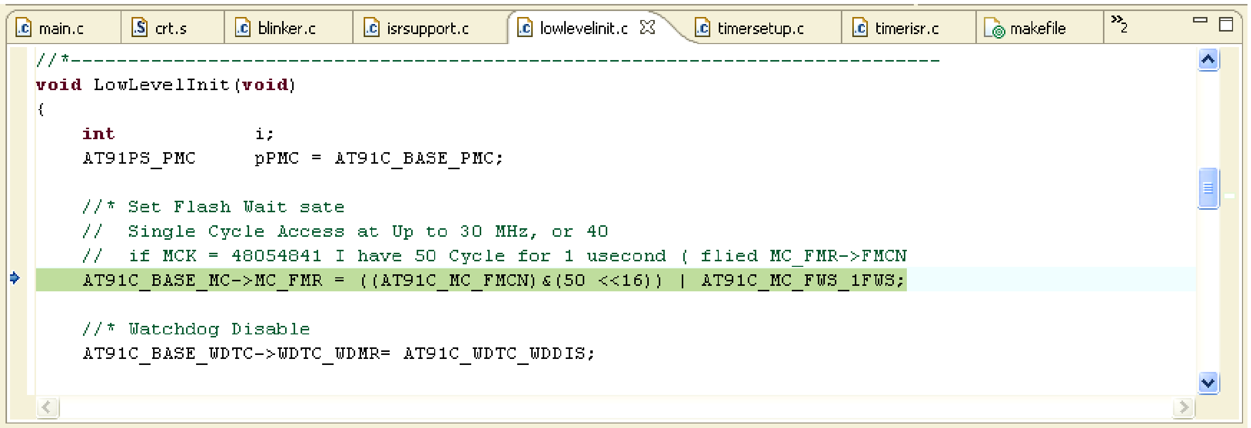
```
typedef void     (*FnPtr)(void);              // create a "pointer to function" type
FnPtr            pFnPtr;                      // pointer to a function
const char       *pText = "The rain in Spain"; // initialized string pointer variable

struct EntryLock {                            // initialized structure variable
    long         Key;
    int          nAccesses;
    char         Name[17];
}  Access = {14705, 0, "Sophie Marceau"};

// Initialize the Atmel AT91SAM7S256 (watchdog, PLL clock, default interrupts, etc.)
// ---------------------------------------------------------------------------------
LowLevelInit();


// Turn on the peripheral clock for Timer0
// ---------------------------------------

// pointer to PMC data structure
volatile AT91PS_PMC pPMC = AT91C_BASE_PMC;

// enable Timer0 peripheral clock
pPMC->PMC_PCER = (1<<AT91C_ID_TC0);
```

Click the "**Step Into**" button The debugger will enter the LowLevelInit( ) function.



```
//* \brief This function performs very low level HW initialization
//*        this function can be use a Stack, depending the compilation
//*        optimization mode
//*----------------------------------------------------------------------------
void LowLevelInit(void)
{
    int            i;
    AT91PS_PMC     pPMC = AT91C_BASE_PMC;

    //* Set Flash Wait sate
    //  Single Cycle Access at Up to 30 MHz, or 40
    //  if MCK = 48054841 I have 50 Cycle for 1 usecond ( flied MC_FMR->FMCN
    AT91C_BASE_MC->MC_FMR = ((AT91C_MC_FMCN)&(50 <<16)) | AT91C_MC_FWS_1FWS;

    //* Watchdog Disable
    AT91C_BASE_WDTC->WDTC_WDMR= AT91C_WDTC_WDDIS;
```

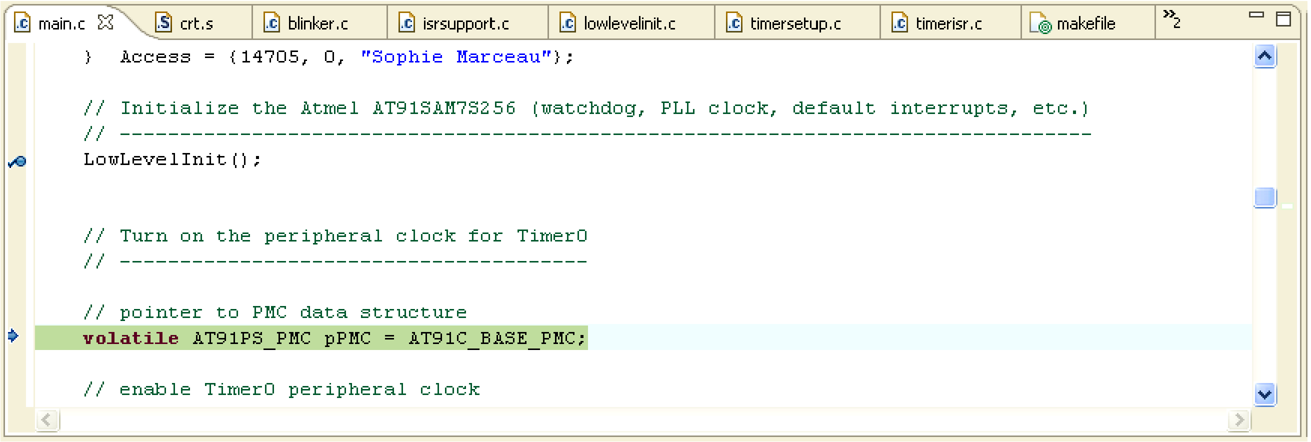
Click the “**Step Over**” button The debugger will execute one instruction.



```
//*----------------------------------------------------------------------------
void LowLevelInit(void)
{
    int            i;
    AT91PS_PMC     pPMC = AT91C_BASE_PMC;

    //* Set Flash Wait sate
    //  Single Cycle Access at Up to 30 MHz, or 40
    //  if MCK = 48054841 I have 50 Cycle for 1 usecond ( flied MC_FMR->FMCN
    AT91C_BASE_MC->MC_FMR = ((AT91C_MC_FMCN)&(50 <<16)) | AT91C_MC_FWS_1FWS;

    //* Watchdog Disable
    AT91C_BASE_WDTC->WDTC_WDMR= AT91C_WDTC_WDDIS;
```

Notice that the “**Step Out Of**” button is illuminated. Click the “**Step Out Of**” button

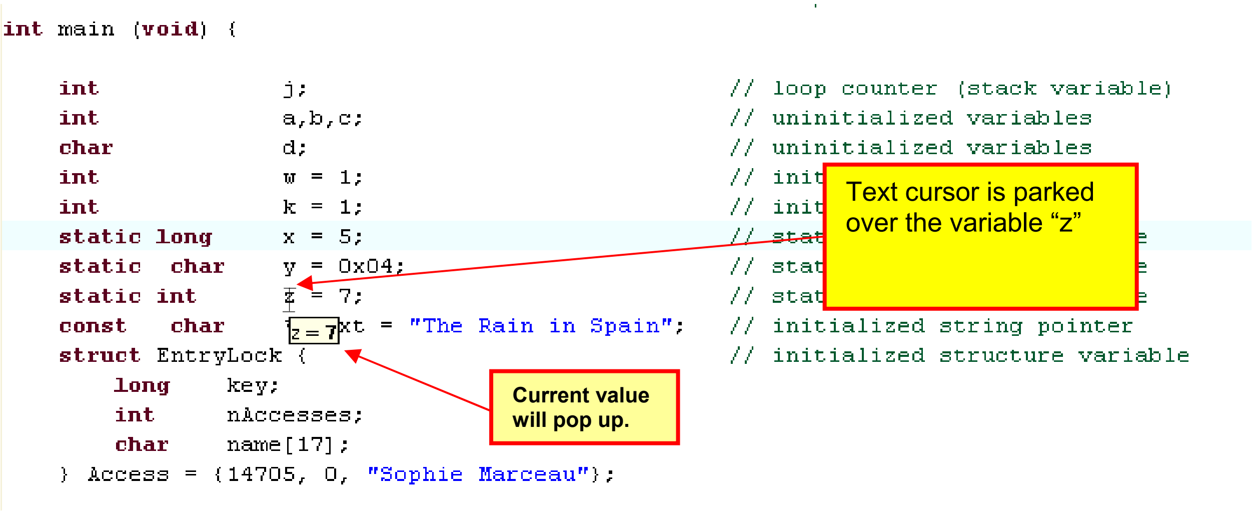
The debugger will execute the remaining instructions in LowLevelInit( ) and return to just after the function call.



```
    } Access = {14705, 0, "Sophie Marceau"};

    // Initialize the Atmel AT91SAM7S256 (watchdog, PLL clock, default interrupts, etc.)
    // ---------------------------------------------------------------------------------
    LowLevelInit();


    // Turn on the peripheral clock for Timer0
    // ----------------------------------------

    // pointer to PMC data structure
    volatile AT91PS_PMC pPMC = AT91C_BASE_PMC;

    // enable Timer0 peripheral clock
```

Inspecting and Modifying Variables

The simple way to inspect variables is to just park the cursor over the variable name in the source window; the current value will pop up in a tiny text box. Execution must be stopped for this to work; either by breakpoint or pause. In this operation, try to position the text cursor within the variable name.

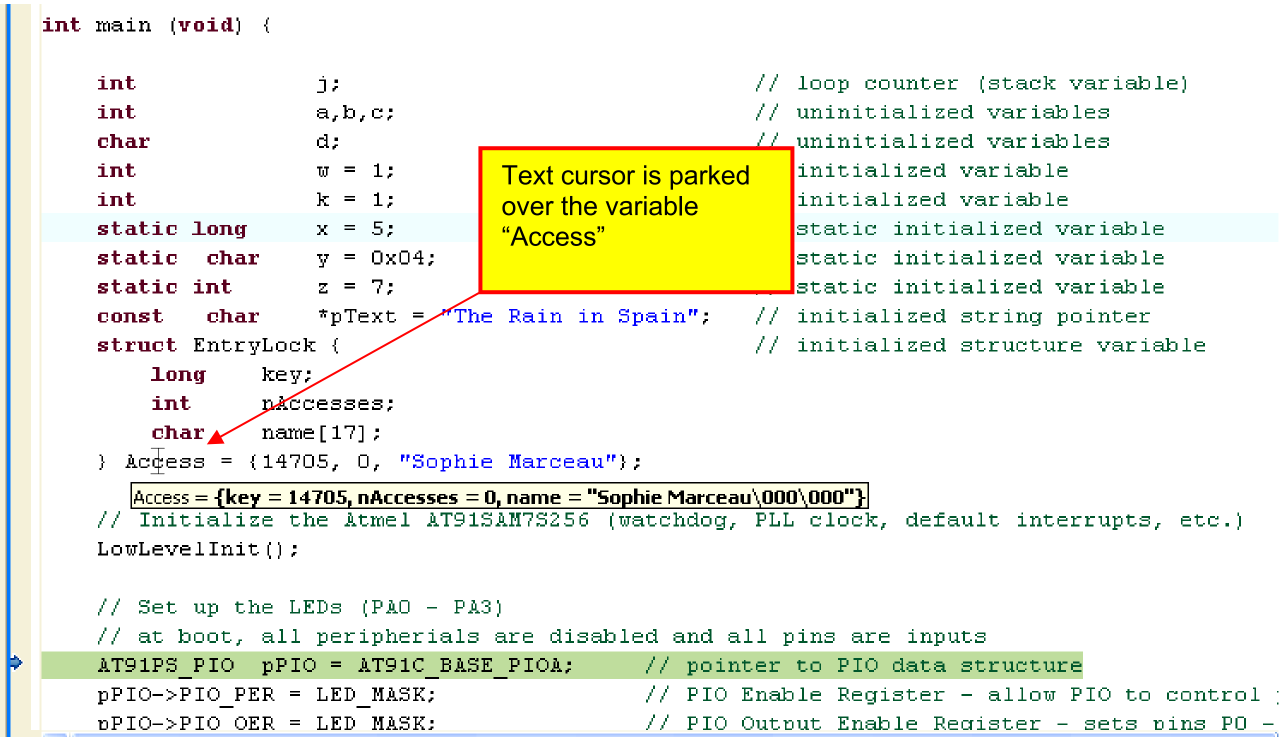


```
int main (void) {

    int             j;                              // loop counter (stack variable)
    int             a,b,c;                          // uninitialized variables
    char            d;                              // uninitialized variables
    int             w = 1;                          // init
    int             k = 1;                          // init
    static long     x = 5;                          // stat
    static  char    y = 0x04;                       // stat
    static int      z = 7;                          // stat
    const   char    z = 7xt = "The Rain in Spain";  // initialized string pointer
    struct EntryLock {                              // initialized structure variable
        long    key;
        int     nAccesses;
        char    name[17];
    } Access = {14705, 0, "Sophie Marceau"};
```

Text cursor is parked over the variable “z”

Current value will pop up.

For a structured variable, parking the cursor over the variable name will show the values of all the internal component parts.

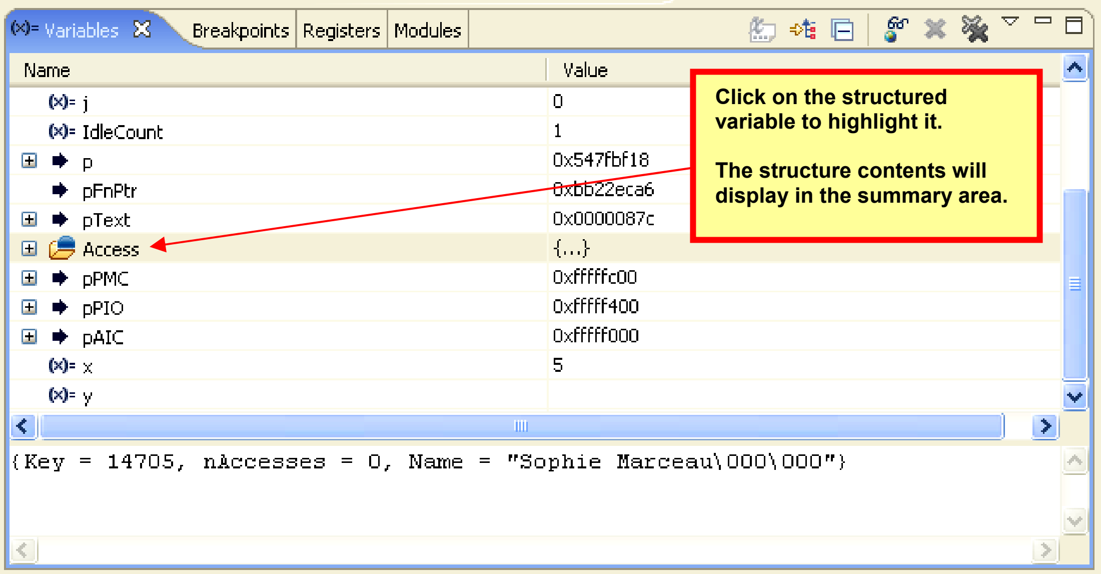


```
int main (void) {

    int             j;                              // loop counter (stack variable)
    int             a,b,c;                          // uninitialized variables
    char            d;                              // uninitialized variables
    int             w = 1;                          initialized variable
    int             k = 1;                          initialized variable
    static long     x = 5;                          static initialized variable
    static  char    y = 0x04;                       static initialized variable
    static int      z = 7;                          static initialized variable
    const   char    *pText = "The Rain in Spain";   // initialized string pointer
    struct EntryLock {                              // initialized structure variable
        long    key;
        int     nAccesses;
        char    name[17];
    } Access = {14705, 0, "Sophie Marceau"};
      Access = {key = 14705, nAccesses = 0, name = "Sophie Marceau\000\000"}
    // Initialize the Atmel AT91SAM7S256 (watchdog, PLL clock, default interrupts, etc.)
    LowLevelInit();

    // Set up the LEDs (PA0 - PA3)
    // at boot, all peripherials are disabled and all pins are inputs
    AT91PS_PIO  pPIO = AT91C_BASE_PIOA;     // pointer to PIO data structure
    pPIO->PIO_PER = LED_MASK;               // PIO Enable Register - allow PIO to control :
    pPIO->PIO_OER = LED_MASK;               // PIO Output Enable Register - sets pins P0 -
```

Text cursor is parked over the variable “Access”

Another way to look at the local variables is to inspect the “**Variables**” view. This will automatically display all automatic variables in the current stack frame. It can also display any global variables that you choose. For simple scalar variables, the value is printed next to the variable name.

If you click on a variable, its value appears in the summary area at the bottom. This is handy for a structured variable or a pointer; wherein the debugger will expand the variable in the summary area.



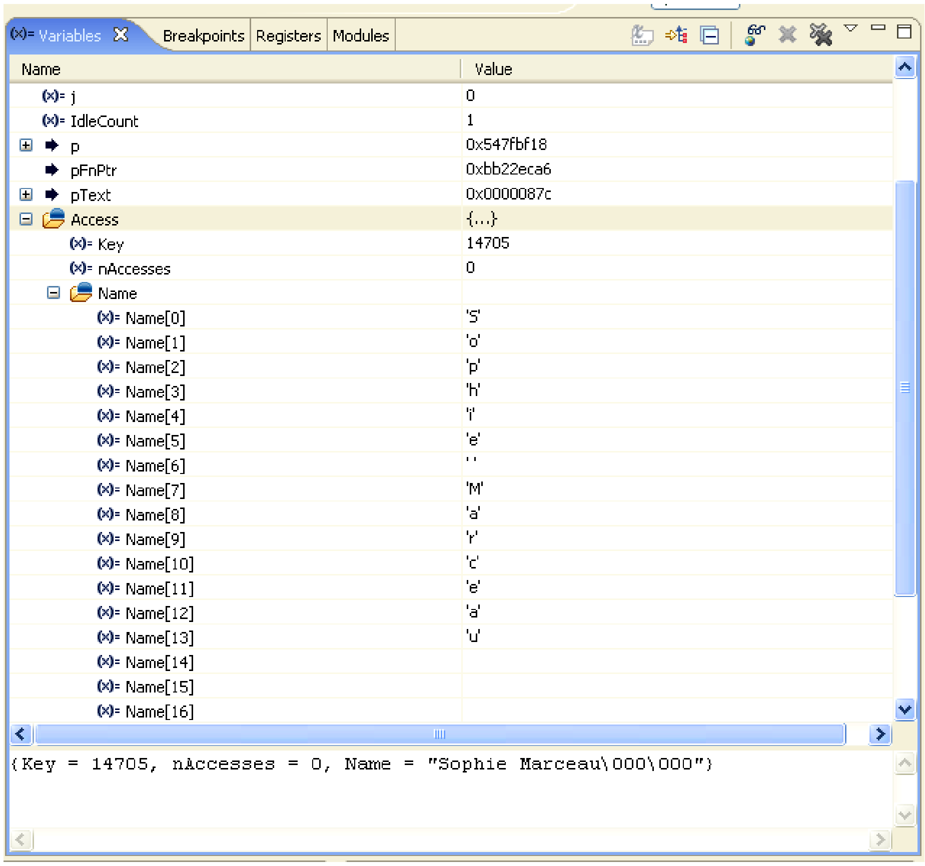
Name	Value
j	0
IdleCount	1
p	0x547fbf18
pFnPtr	0xbb22eca6
pText	0x0000087c
Access	{...}
pPMC	0xfffffc00
pPIO	0xfffff400
pAIC	0xfffff000
x	5
y	

Click on the structured variable to highlight it.

The structure contents will display in the summary area.

```
{Key = 14705, nAccesses = 0, Name = "Sophie Marceau\000\000"}
```

The Variables view can also expand structures. Just click on any "+" signs you see to expand the structure and view its contents.



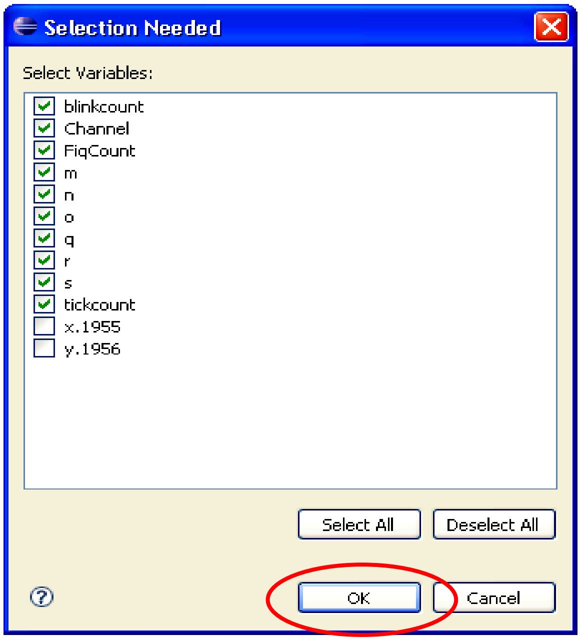
Name	Value
j	0
IdleCount	1
p	0x547fbf18
pFnPtr	0xbb22eca6
pText	0x0000087c
Access	{...}
Key	14705
nAccesses	0
Name	
Name[0]	'S'
Name[1]	'o'
Name[2]	'p'
Name[3]	'h'
Name[4]	'i'
Name[5]	'e'
Name[6]	' '
Name[7]	'M'
Name[8]	'a'
Name[9]	'r'
Name[10]	'c'
Name[11]	'e'
Name[12]	'a'
Name[13]	'u'
Name[14]	
Name[15]	
Name[16]	

```
{Key = 14705, nAccesses = 0, Name = "Sophie Marceau\000\000")
```

Global variables have to be individually selected for display within the "Variables" view.

Use the "Add Global Variables" button to open the selection dialog.

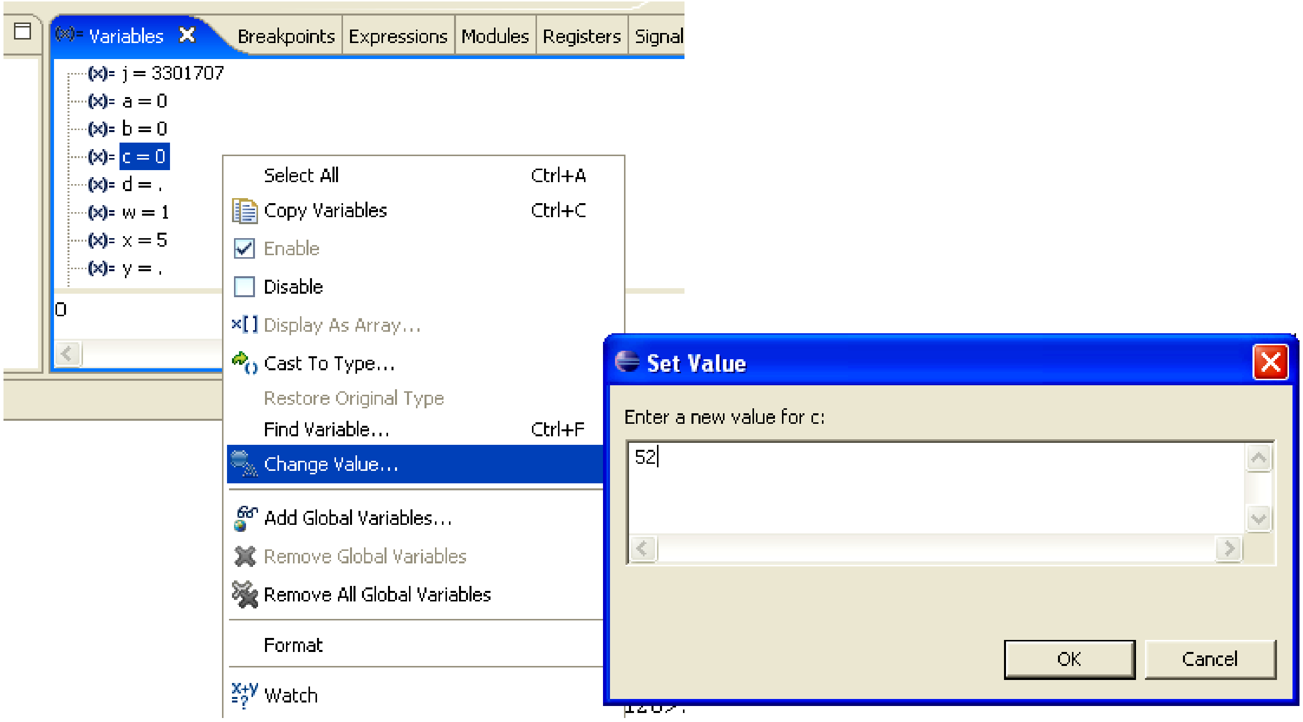
Check the variables you want to display and then click “**OK**” to add them to the **Variables** view,



Select Variables:

☑ blinkcount
☑ Channel
☑ FiqCount
☑ m
☑ n
☑ o
☑ q
☑ r
☑ s
☑ tickcount
□ x.1955
□ y.1956

You can easily change the value of a variable at any time. Assuming that the debugger has stopped, click on the variable you wish to change and right click. In the right-click menu, select “**Change Value…**” and enter the new value into the pop-up window as shown below. In this example, we change the variable “c” to 52. Resist the temptation to hit the “Enter” key on your keyboard to signal completion of the new value; doing so will invalidate your entry. You must click the “**OK**” button to register your change.



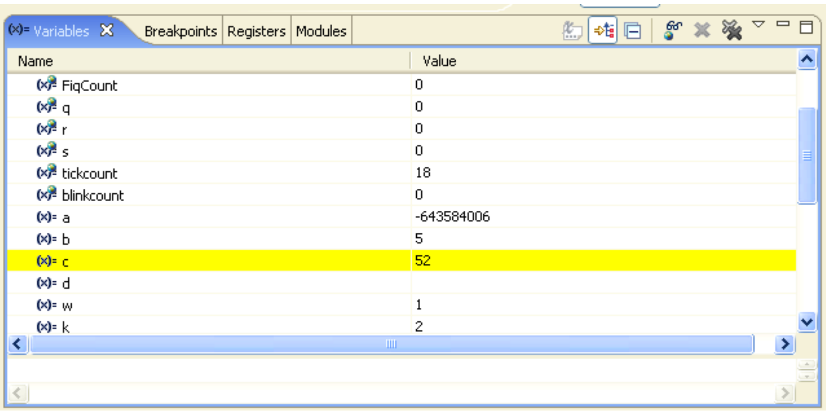
j = 3301707
a = 0
b = 0
c = 0
d = .
w = 1
x = 5
y = .

0

Enter a new value for c:

52

Now the “**Variables**” view should show the new value for the variable “c”. Note that it has been back lighted to the color yellow to indicate that it has been changed.

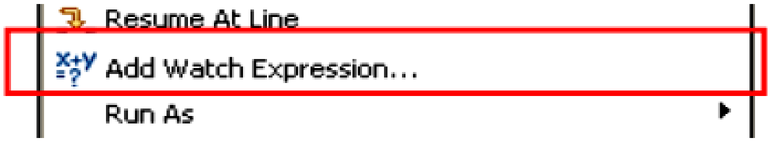


Name	Value
FiqCount	0
q	0
r	0
s	0
tickcount	18
blinkcount	0
a	-643584006
b	5
c	52
d	
w	1
k	2

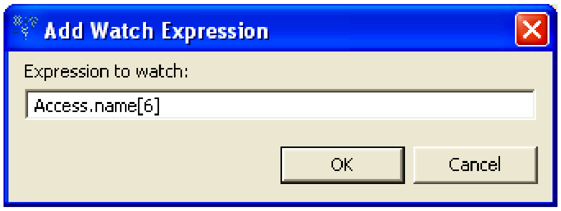
Watch Expressions

The “Expressions” view can display the results of expressions (any legal C Language expression). Since it can pick any local or global variable, it forms the basis of a customizable variable display; showing only the information you want.

For example, to display the 6th character of the name in the structured variable “**Access**”, bring up the right-click menu and select “**Add Watch Expression…**”.

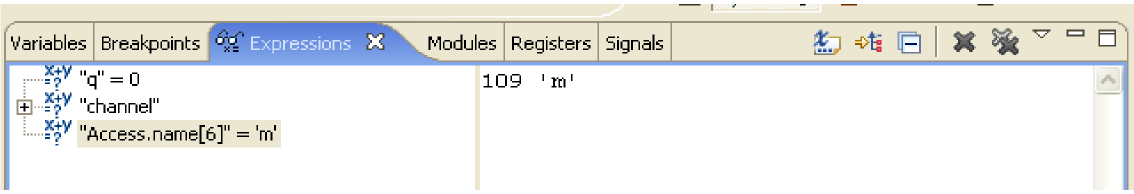


Enter the fully qualified name of the 6th character of the name[] array.



Expression to watch: Access.name[6]

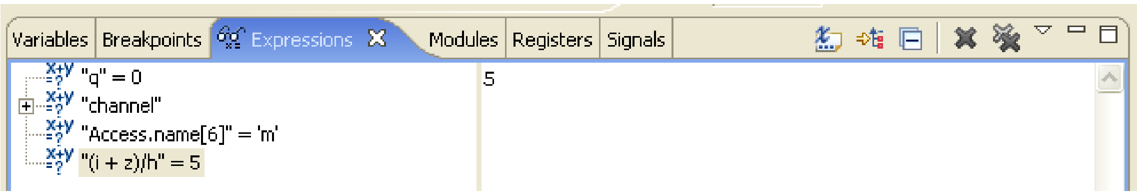
Note that it now appears in the “Expressions” view.



"q" = 0
"channel"
"Access.name[6]" = 'm'

109 'm'

You can type in very complicated expressions. Here we defined the expression (i + z)/h



"q" = 0
"channel"
"Access.name[6]" = 'm'
"(i + z)/h" = 5

5

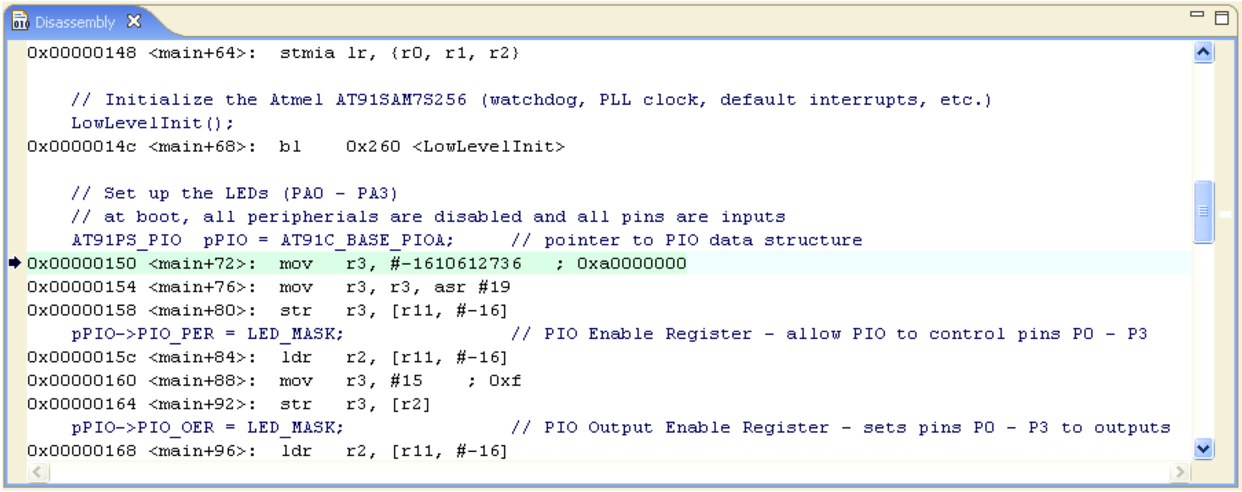
Assembly Language Debugging

The Debug perspective includes an Assembly Language view.

If you click on the Instruction Stepping Mode toggle button in the Debug view,

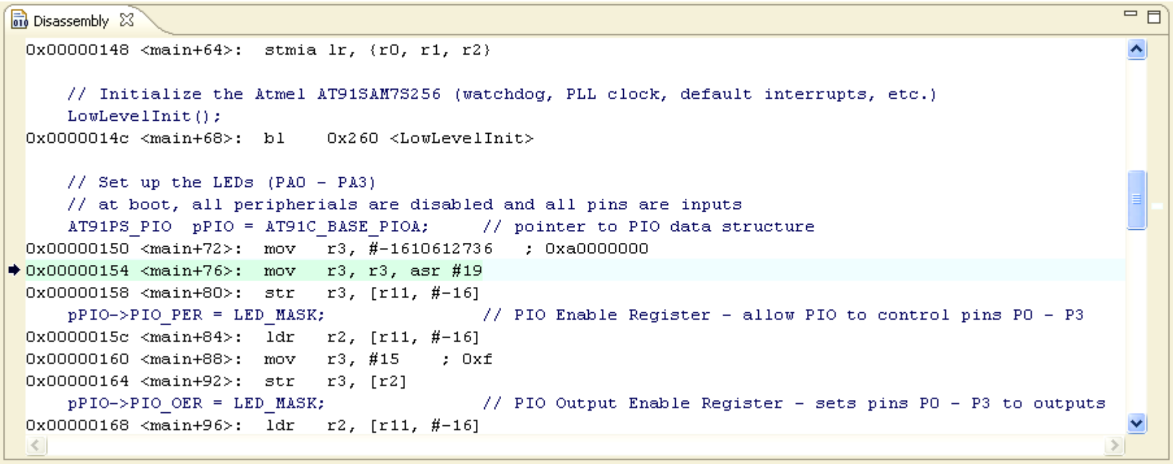
the assembly language window becomes active and the single-step buttons apply to the assembler window. The single-step buttons will advance the program by a single assembler instruction. Note that the “Disassembly” tab lights up when the assembler view has control.

Note that the debugger is currently stopped at the assembler line at address 0x0000150.



```
0x00000148 <main+64>:  stmia lr, {r0, r1, r2}

    // Initialize the Atmel AT91SAM7S256 (watchdog, PLL clock, default interrupts, etc.)
    LowLevelInit();
0x0000014c <main+68>:  bl    0x260 <LowLevelInit>

    // Set up the LEDs (PA0 - PA3)
    // at boot, all peripherials are disabled and all pins are inputs
    AT91PS_PIO  pPIO = AT91C_BASE_PIOA;     // pointer to PIO data structure
0x00000150 <main+72>:  mov   r3, #-1610612736    ; 0xa0000000
0x00000154 <main+76>:  mov   r3, r3, asr #19
0x00000158 <main+80>:  str   r3, [r11, #-16]
    pPIO->PIO_PER = LED_MASK;               // PIO Enable Register - allow PIO to control pins P0 - P3
0x0000015c <main+84>:  ldr   r2, [r11, #-16]
0x00000160 <main+88>:  mov   r3, #15    ; 0xf
0x00000164 <main+92>:  str   r3, [r2]
    pPIO->PIO_OER = LED_MASK;               // PIO Output Enable Register - sets pins P0 - P3 to outputs
0x00000168 <main+96>:  ldr   r2, [r11, #-16]
```

If we click the “**Step Over**” button in the Debug view, the debugger will execute one assembler line.



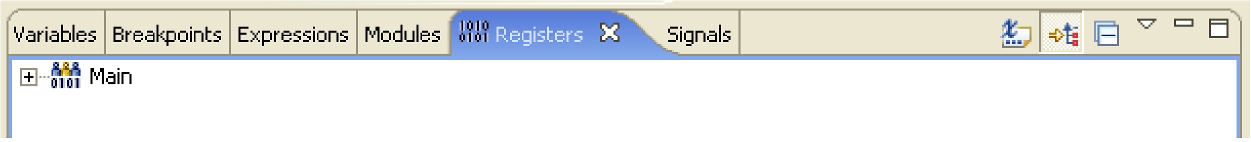
```
0x00000148 <main+64>:  stmia lr, {r0, r1, r2}

    // Initialize the Atmel AT91SAM7S256 (watchdog, PLL clock, default interrupts, etc.)
    LowLevelInit();
0x0000014c <main+68>:  bl    0x260 <LowLevelInit>

    // Set up the LEDs (PA0 - PA3)
    // at boot, all peripherials are disabled and all pins are inputs
    AT91PS_PIO  pPIO = AT91C_BASE_PIOA;     // pointer to PIO data structure
0x00000150 <main+72>:  mov   r3, #-1610612736    ; 0xa0000000
0x00000154 <main+76>:  mov   r3, r3, asr #19
0x00000158 <main+80>:  str   r3, [r11, #-16]
    pPIO->PIO_PER = LED_MASK;               // PIO Enable Register - allow PIO to control pins P0 - P3
0x0000015c <main+84>:  ldr   r2, [r11, #-16]
0x00000160 <main+88>:  mov   r3, #15    ; 0xf
0x00000164 <main+92>:  str   r3, [r2]
    pPIO->PIO_OER = LED_MASK;               // PIO Output Enable Register - sets pins P0 - P3 to outputs
0x00000168 <main+96>:  ldr   r2, [r11, #-16]
```

The “**Step Into**” and “**Step Out Of**” buttons work in the same way as for C code.

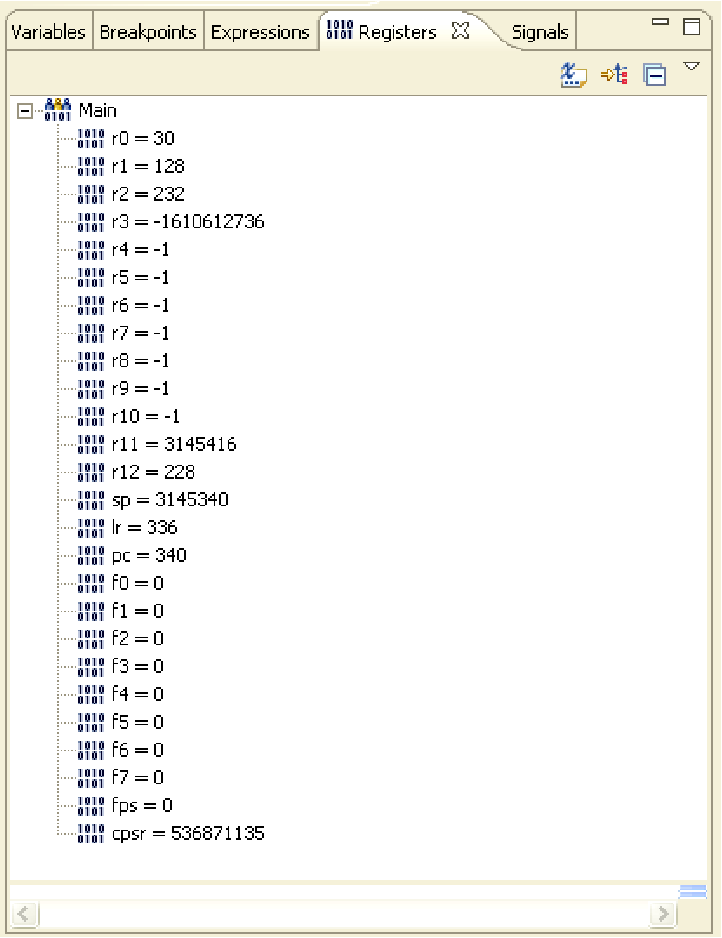
Note: It pains the author greatly to report that the Eclipse 3.2 release has a bug wherein assembly language breakpoints do not function. Monitor the chat boards to see when this is resolved. Truthfully, you shouldn’t be programming in assembly language anyway!

Inspecting Registers

Unfortunately, parking the cursor over a register name (R3 e.g.) does not pop up its current value. For that, you can refer to the “Registers” view.

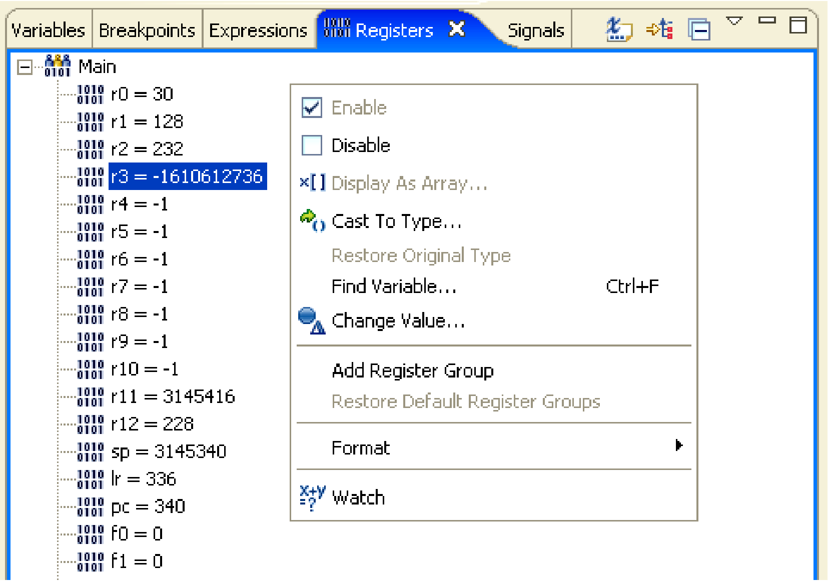


Click on the “**+**” symbol next to Main and the registers will appear. The Atmel AT91SAM7S256 doesn’t have any floating point registers so registers F0 through FPS are not applicable.



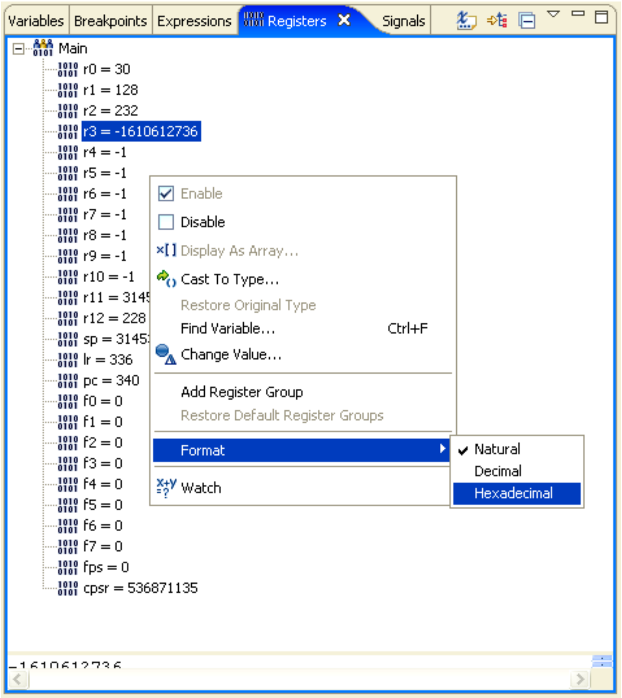
```
Main
  r0 = 30
  r1 = 128
  r2 = 232
  r3 = -1610612736
  r4 = -1
  r5 = -1
  r6 = -1
  r7 = -1
  r8 = -1
  r9 = -1
  r10 = -1
  r11 = 3145416
  r12 = 228
  sp = 3145340
  lr = 336
  pc = 340
  f0 = 0
  f1 = 0
  f2 = 0
  f3 = 0
  f4 = 0
  f5 = 0
  f6 = 0
  f7 = 0
  fps = 0
  cpsr = 536871135
```

If you don’t like a particular register’s numeric format, you can click to highlight it and then bring up the right-click menu. You can, of course, drag the mouse cursor to highlight them all if desired.



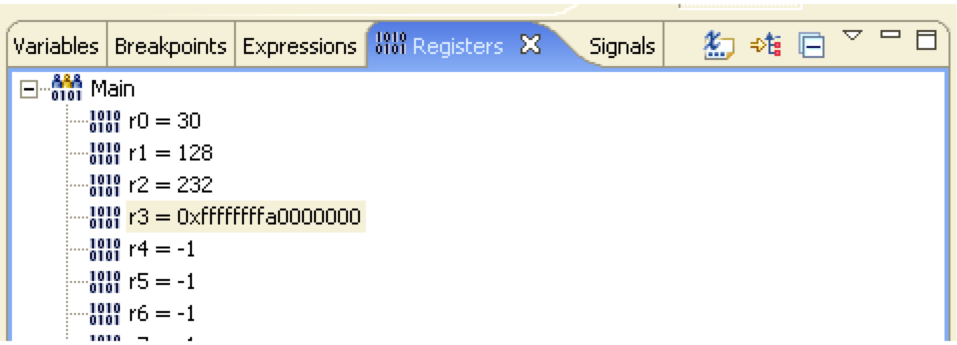
```
r0 = 30
r1 = 128
r2 = 232
r3 = -1610612736
r4 = -1
r5 = -1
r6 = -1
r7 = -1
r8 = -1
r9 = -1
r10 = -1
r11 = 3145416
r12 = 228
sp = 3145340
lr = 336
pc = 340
f0 = 0
f1 = 0
```

The “**Format**” option permits you to change the numeric format to hexadecimal, for example.



```
r0 = 30
r1 = 128
r2 = 232
r3 = -1610612736
r4 = -1
r5 = -1
r6 = -1
r7 = -1
r8 = -1
r9 = -1
r10 = -1
r11 = 314
r12 = 228
sp = 3145
lr = 336
pc = 340
f0 = 0
f1 = 0
f2 = 0
f3 = 0
f4 = 0
f5 = 0
f6 = 0
f7 = 0
fps = 0
cpsr = 536871135

-1610612736
```

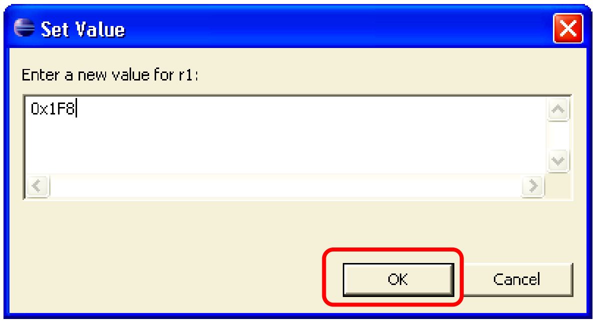
Now R3 is displayed in hexadecimal.



```
r0 = 30
r1 = 128
r2 = 232
r3 = 0xffffffffa0000000
r4 = -1
r5 = -1
r6 = -1
```

Of course, the right click menu lets you change the value of any register. For example, to change **r1** from **128** to **0x1F8**, just select the register, right-click and select "**Change Value…**"

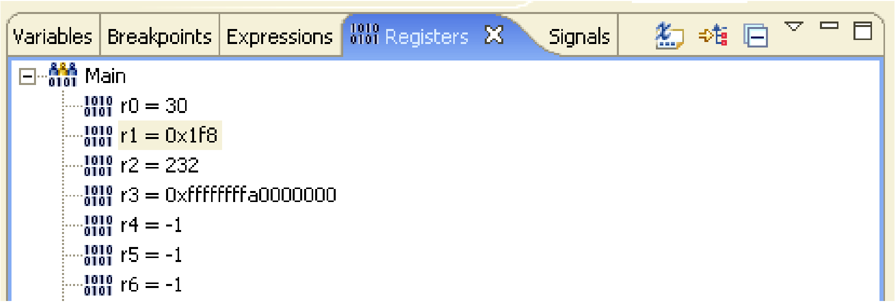
In the "Set Value" dialog box, enter the hexadecimal value **0x1F8** and click "**OK**" to accept.



Enter a new value for r1:

0x1F8

The register display now shows the new value for R1 (we also changed the display format to hexadecimal using the right-click menu).

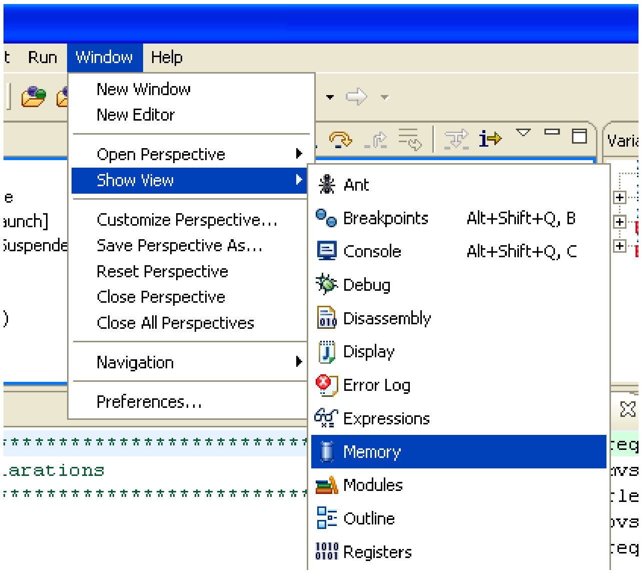


```
Main
  r0 = 30
  r1 = 0x1f8
  r2 = 232
  r3 = 0xffffffffa0000000
  r4 = -1
  r5 = -1
  r6 = -1
```

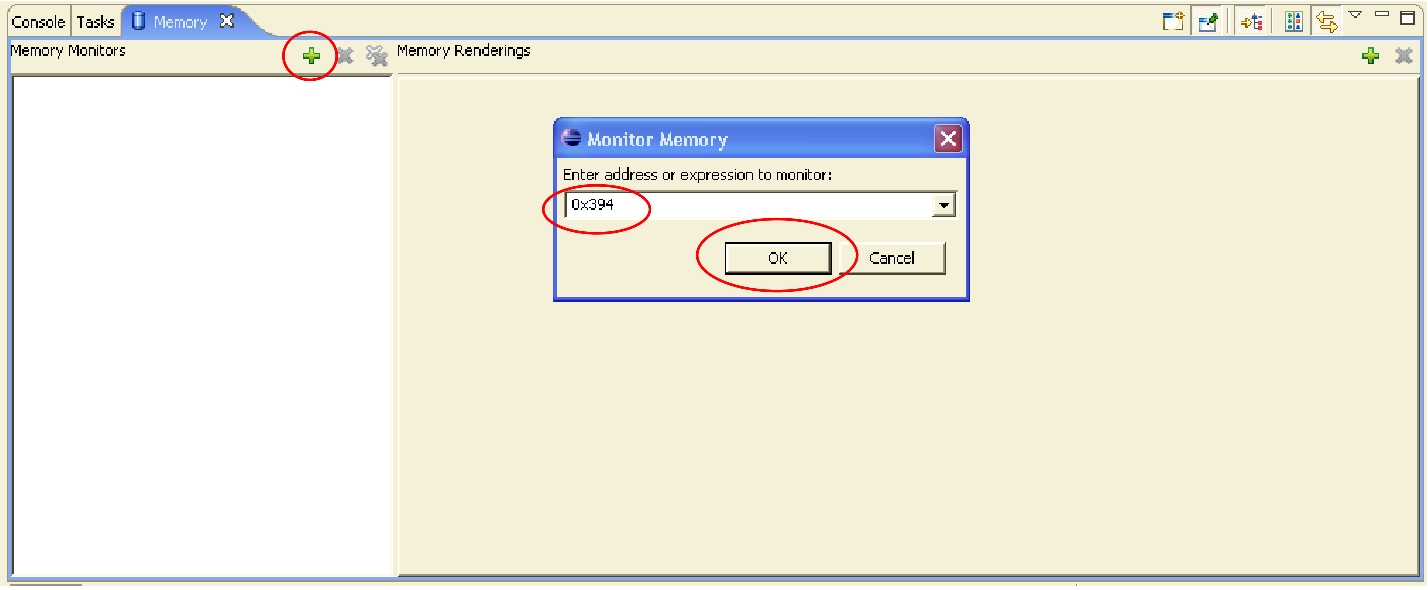
It goes without saying that you had better use this feature with great care! Make sure you know what you are doing before tampering with the ARM registers.

Inspecting Memory

Viewing memory is a bit complex in Eclipse. First, the memory view is not part of the default debug launch configuration. You can add it by clicking "**Window – Show View – Memory**" as shown below.



The memory view appears with the "**Console**" view at the bottom of the Debug perspective. At this point, nothing has been defined. Memory is displayed as one or more "**memory monitors**". You can create a memory monitor by clicking on the "+" symbol. Enter the address **0x394** (address of the string "The Rain in Spain") in the dialog box and click "**OK**".

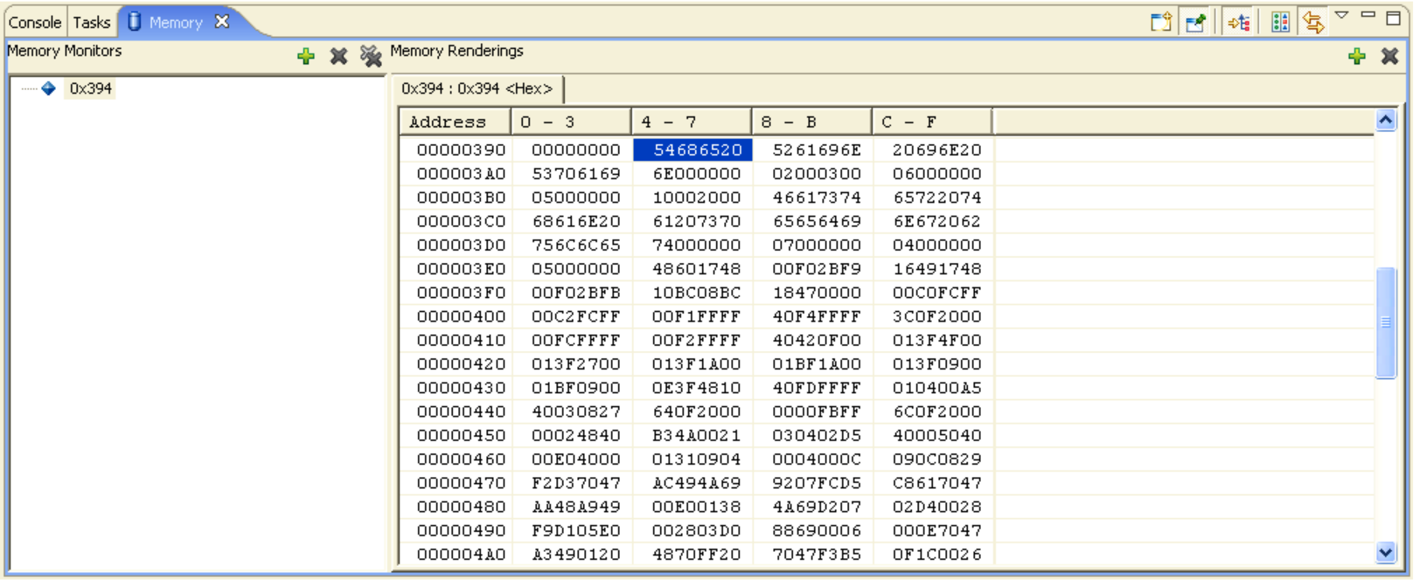


Enter address or expression to monitor:

0x394

The memory monitor is created, although it defaults to 4-byte display mode. The display of the address columns and the associated memory contents is called a "**Rendering**".

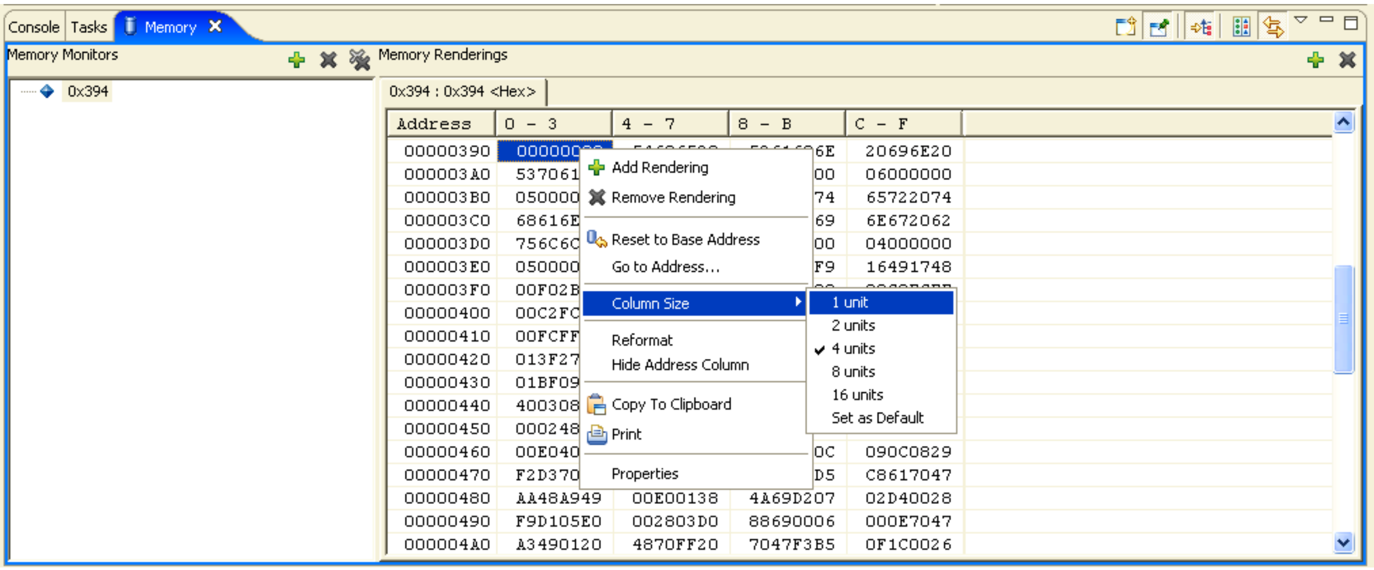
The address **0x394** is called the Base Address; there's a right-click menu option "**Reset to Base Address**" that will automatically return you to this address if you scroll the memory display.



```
Address   0 - 3     4 - 7     8 - B     C - F
00000390  00000000  54686520  5261696E  20696E20
000003A0  53706169  6E000000  02000300  06000000
000003B0  05000000  10002000  46617374  65722074
000003C0  68616E20  61207370  65656469  6E672062
000003D0  756C6C65  74000000  07000000  04000000
000003E0  05000000  48601748  00F02BF9  16491748
000003F0  00F02BFB  10BC08BC  18470000  00C0FCFF
00000400  00C2FCFF  00F1FFFF  40F4FFFF  3C0F2000
00000410  00FCFFFF  00F2FFFF  40420F00  013F4F00
00000420  013F2700  013F1A00  01BF1A00  013F0900
00000430  01BF0900  0E3F4810  40FDFFFF  010400A5
00000440  40030827  640F2000  0000FBFF  6C0F2000
00000450  00024840  B34A0021  030402D5  40005040
00000460  00E04000  01310904  0004000C  090C0829
00000470  F2D37047  AC494A69  9207FCD5  C8617047
00000480  AA48A949  00E00138  4A69D207  02D40028
00000490  F9D105E0  002803D0  88690006  000E7047
000004A0  A3490120  4870FF20  7047F3B5  0F1C0026
```

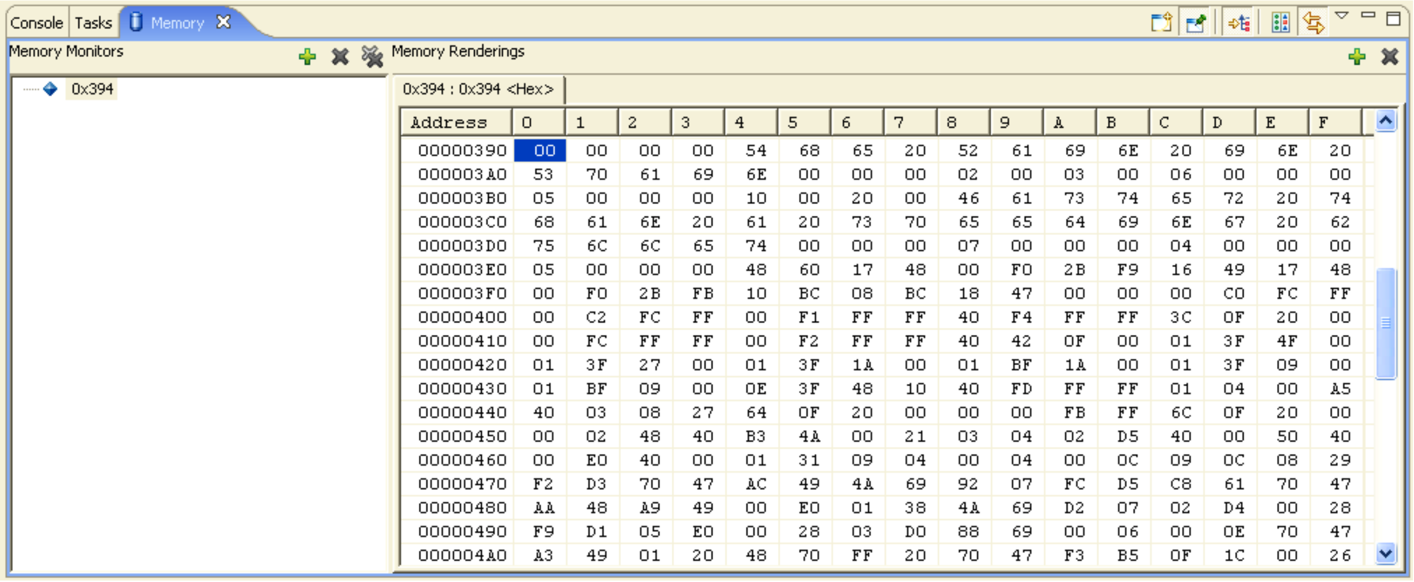
There's also a "**Go to Address…**" right-click menu option that will jump all over memory for you.

By right-clicking anywhere within the memory rendering (display area), you can select “**Column Size – 1 unit**”.



Address	0 - 3	4 - 7	8 - B	C - F
00000390	000000		6E	20696E20
000003A0	537061		00	06000000
000003B0	050000		74	65722074
000003C0	68616E		69	6E672062
000003D0	756C6C		00	04000000
000003E0	050000		F9	16491748
000003F0	00F02B			
00000400	00C2FC			
00000410	00FCFF			
00000420	013F27			
00000430	01BF09			
00000440	400308			
00000450	000248			
00000460	00E040		0C	090C0829
00000470	F2D370		D5	C8617047
00000480	AA48A949	00E00138	4A69D207	02D40028
00000490	F9D105E0	002803D0	88690006	000E7047
000004A0	A3490120	4870FF20	7047F3B5	0F1C0026

This will repaint the memory rendering in Byte format as shown below.



Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000390	00	00	00	00	54	68	65	20	52	61	69	6E	20	69	6E	20
000003A0	53	70	61	69	6E	00	00	00	02	00	03	00	06	00	00	00
000003B0	05	00	00	00	10	00	20	00	46	61	73	74	65	72	20	74
000003C0	68	61	6E	20	61	20	73	70	65	65	64	69	6E	67	20	62
000003D0	75	6C	6C	65	74	00	00	00	07	00	00	00	04	00	00	00
000003E0	05	00	00	00	48	60	17	48	00	F0	2B	F9	16	49	17	48
000003F0	00	F0	2B	FB	10	BC	08	BC	18	47	00	00	00	C0	FC	FF
00000400	00	C2	FC	FF	00	F1	FF	FF	40	F4	FF	FF	3C	0F	20	00
00000410	00	FC	FF	FF	00	F2	FF	FF	40	42	0F	00	01	3F	4F	00
00000420	01	3F	27	00	01	3F	1A	00	01	BF	1A	00	01	3F	09	00
00000430	01	BF	09	00	0E	3F	48	10	40	FD	FF	FF	01	04	00	A5
00000440	40	03	08	27	64	0F	20	00	00	00	FB	FF	6C	0F	20	00
00000450	00	02	48	40	B3	4A	00	21	03	04	02	D5	40	00	50	40
00000460	00	E0	40	00	01	31	09	04	00	04	00	0C	09	0C	08	29
00000470	F2	D3	70	47	AC	49	4A	69	92	07	FC	D5	C8	61	70	47
00000480	AA	48	A9	49	00	E0	01	38	4A	69	D2	07	02	D4	00	28
00000490	F9	D1	05	E0	00	28	03	D0	88	69	00	06	00	0E	70	47
000004A0	A3	49	01	20	48	70	FF	20	70	47	F3	B5	0F	1C	00	26

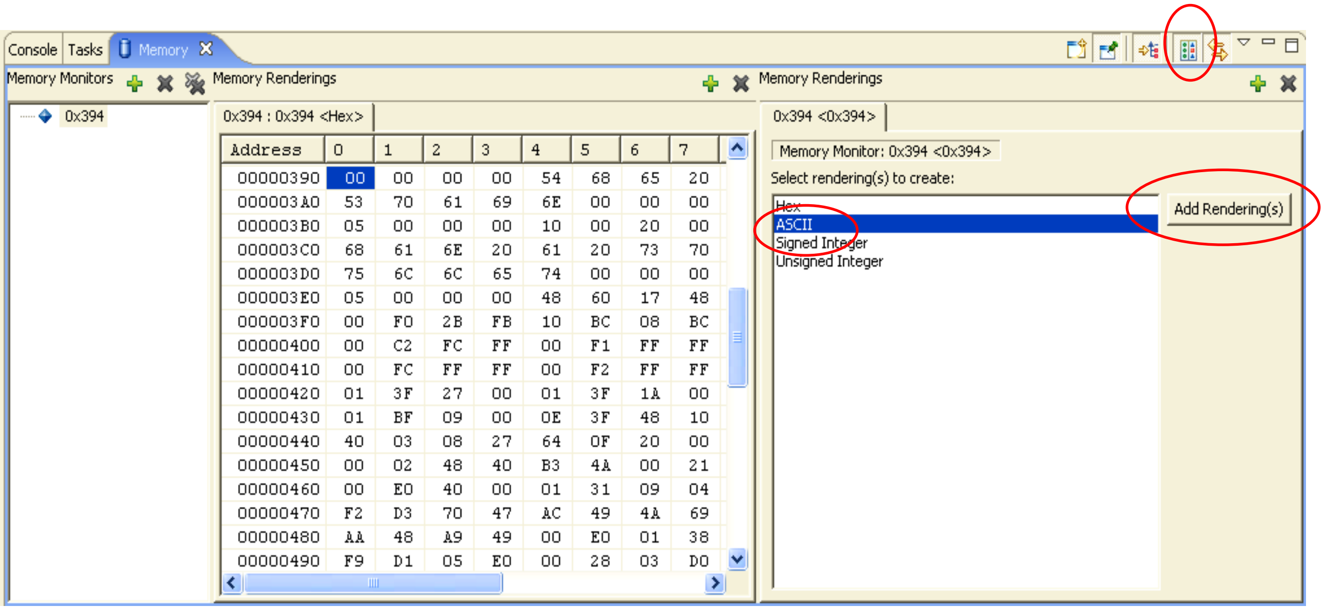
The Eclipse memory display allows you to simply type new values into the displayed cells. Of course, this example is in FLASH and that wouldn’t work. Memory displays in the RAM area can be edited.

Now we will add a second rendering that will display the memory monitor in ASCII.

Click on the “**Toggle Split Pane**” button to create a second rendering pane.

Pick “**ASCII**” display for the new rendering.

Click on the “**Add Rendering(s)**” button to complete the specification of an additional ASCII memory display.



0x394 : 0x394 <Hex>

```
Address   0  1  2  3  4  5  6  7
00000390  00 00 00 00 54 68 65 20
000003A0  53 70 61 69 6E 00 00 00
000003B0  05 00 00 00 10 00 20 00
000003C0  68 61 6E 20 61 20 73 70
000003D0  75 6C 6C 65 74 00 00 00
000003E0  05 00 00 00 48 60 17 48
000003F0  00 F0 2B FB 10 BC 08 BC
00000400  00 C2 FC FF 00 F1 FF FF
00000410  00 FC FF FF 00 F2 FF FF
00000420  01 3F 27 00 01 3F 1A 00
00000430  01 BF 09 00 0E 3F 48 10
00000440  40 03 08 27 64 0F 20 00
00000450  00 02 48 40 B3 4A 00 21
00000460  00 E0 40 00 01 31 09 04
00000470  F2 D3 70 47 AC 49 4A 69
00000480  AA 48 A9 49 00 E0 01 38
00000490  F9 D1 05 E0 00 28 03 D0
```

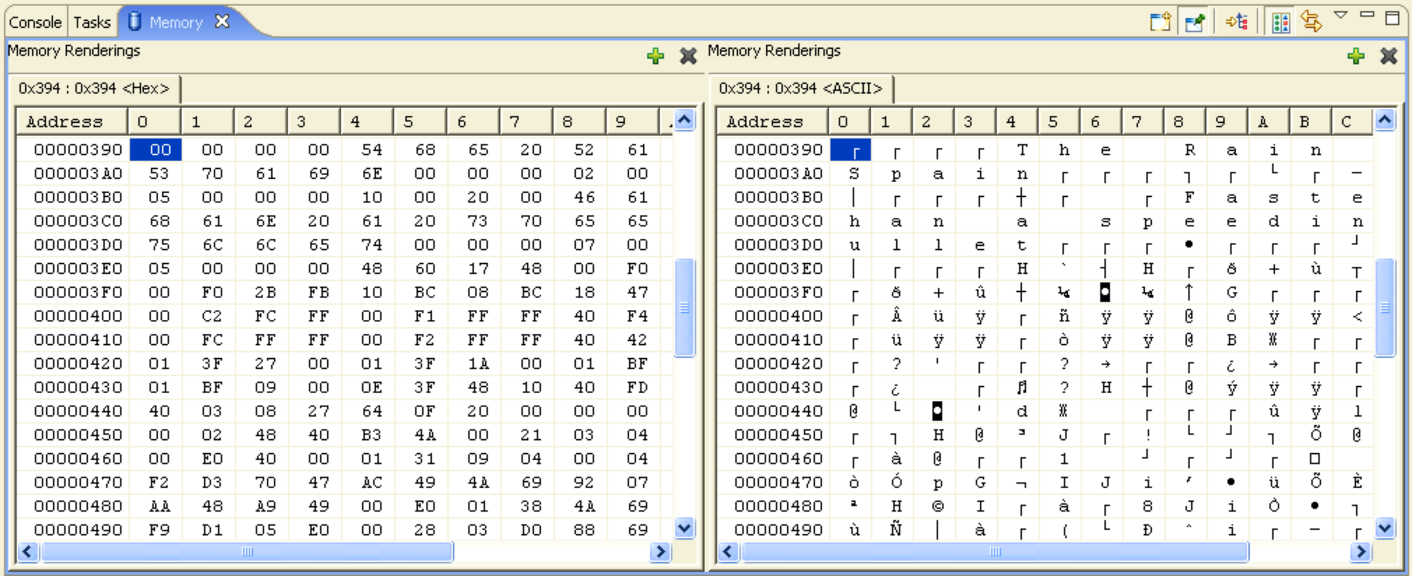
Now we have an additional display of the hex values and the corresponding ASCII characters.

Click on the “**Link Memory Rendering Panes**” button.

This means that scrolling one memory rendering will automatically scroll the other one in synchronism.

Click on the “**Toggle Memory Monitors Pane**” button.

This will expand the display erasing the “memory monitors” list on the left.



0x394 : 0x394 <Hex>

```
Address   0  1  2  3  4  5  6  7  8  9
00000390  00 00 00 00 54 68 65 20 52 61
000003A0  53 70 61 69 6E 00 00 00 02 00
000003B0  05 00 00 00 10 00 20 00 46 61
000003C0  68 61 6E 20 61 20 73 70 65 65
000003D0  75 6C 6C 65 74 00 00 00 07 00
000003E0  05 00 00 00 48 60 17 48 00 F0
000003F0  00 F0 2B FB 10 BC 08 BC 18 47
00000400  00 C2 FC FF 00 F1 FF FF 40 F4
00000410  00 FC FF FF 00 F2 FF FF 40 42
00000420  01 3F 27 00 01 3F 1A 00 01 BF
00000430  01 BF 09 00 0E 3F 48 10 40 FD
00000440  40 03 08 27 64 0F 20 00 00 00
00000450  00 02 48 40 B3 4A 00 21 03 04
00000460  00 E0 40 00 01 31 09 04 00 04
00000470  F2 D3 70 47 AC 49 4A 69 92 07
00000480  AA 48 A9 49 00 E0 01 38 4A 69
00000490  F9 D1 05 E0 00 28 03 D0 88 69
```

0x394 : 0x394 <ASCII>

```
Address   0 1 2 3 4 5 6 7 8 9 A B C
00000390  ┌ ┌ ┌ ┌ T h e   R a i n
000003A0  S p a i n ┌ ┌ ┌ ┐ ┌ └ ┌ ─
000003B0  │ ┌ ┌ ┌ ┼ ┌   ┌ F a s t e
000003C0  h a n   a   s p e e d i n
000003D0  u l l e t ┌ ┌ ┌ • ┌ ┌ ┌ ┘
000003E0  │ ┌ ┌ ┌ H ` ┤ H ┌ ð + ù ┬
000003F0  ┌ ð + û ┼ ¼ ◘ ¼ ↑ G ┌ ┌ ┌
00000400  ┌ Â ü ÿ ┌ ñ ÿ ÿ @ ô ÿ ÿ <
00000410  ┌ ü ÿ ÿ ┌ ò ÿ ÿ @ B ‼ ┌ ┌
00000420  ┌ ? ' ┌ ┌ ? → ┌ ┌ ¿ → ┌ ┌
00000430  ┌ ¿   ┌ ♫ ? H ┼ @ ý ÿ ÿ ┌
00000440  @ └ ◙ ' d ‼   ┌ ┌ ┌ û ÿ l
00000450  ┌ ┐ H @ ³ J ┌ ! └ ┘ ┐ Õ @
00000460  ┌ à @ ┌ ┌ 1   ┘ ┌ ┘ ┌ □
00000470  ò Ó p G ¬ I J i ' • ü Õ È
00000480  ª H © I ┌ à ┌ 8 J i Ò • ┐
00000490  ù Ñ │ à ┌ ( └ Ð ˆ i ┌ ─ ┌
```

Admittedly, this Eclipse memory display is a bit complex. However, it allows you to define many “memory monitors” and clicking on any one of them pops up the renderings instantly. It’s like so many things in life, once you learn how to do it; it seems easy!

Create an Eclipse Project to Run in RAM

There are two reasons to run an application entirely within onboard RAM memory; to gain a speed advantage and to be able to set an unlimited number of software breakpoints.

Execution within RAM is about two times faster than execution within FLASH memory. Many programmers will just copy the routines that need the increased execution speed from FLASH to RAM at run-time and thenceforth call the routines resident in RAM. This is not the subject of this tutorial so we will not address this idea any further.

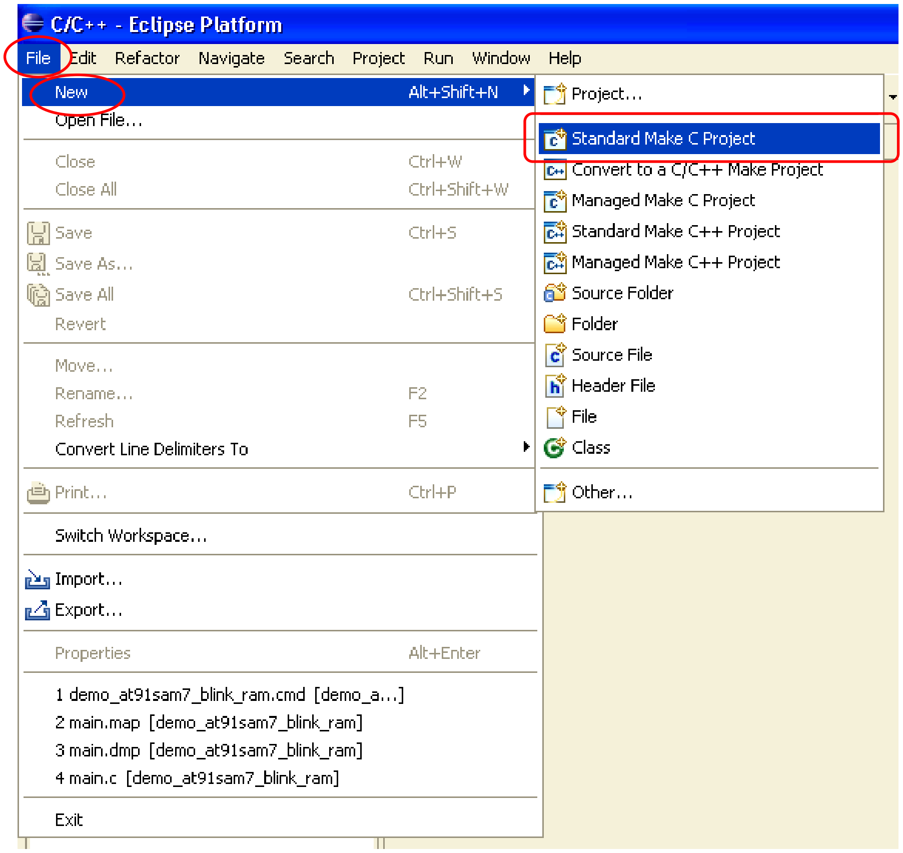
In the FLASH example shown previously, the OpenOCD and J-Link GDB Server utilities permitted the Eclipse debugger to use the two on-chip breakpoint units; thus allowing a breakpoint to be set in FLASH. This limits us to just two breakpoints. Note also that the OpenOCD and J-Link setup converted every Eclipse breakpoint specification into a hardware-assisted breakpoint. This works great but there may be occasions where the two-breakpoint limit is not satisfactory.

Creating an Eclipse project that runs entirely out of on-chip RAM is simple if a bit counter-intuitive. We use the Linker command script to place the code (.text), initialized variables (.data) and uninitialized variables (.bss) all into FLASH at address 0x00000000. When the debugger starts up, we toggle the MC Memory Remap Control Register to place the RAM memory at address 0x000000. We then use our JTAG hardware interface to load the main.out file (containing the executable code) into RAM now at address 0x00000000 and away we go! It’s almost as if Flash memory has become read/write.

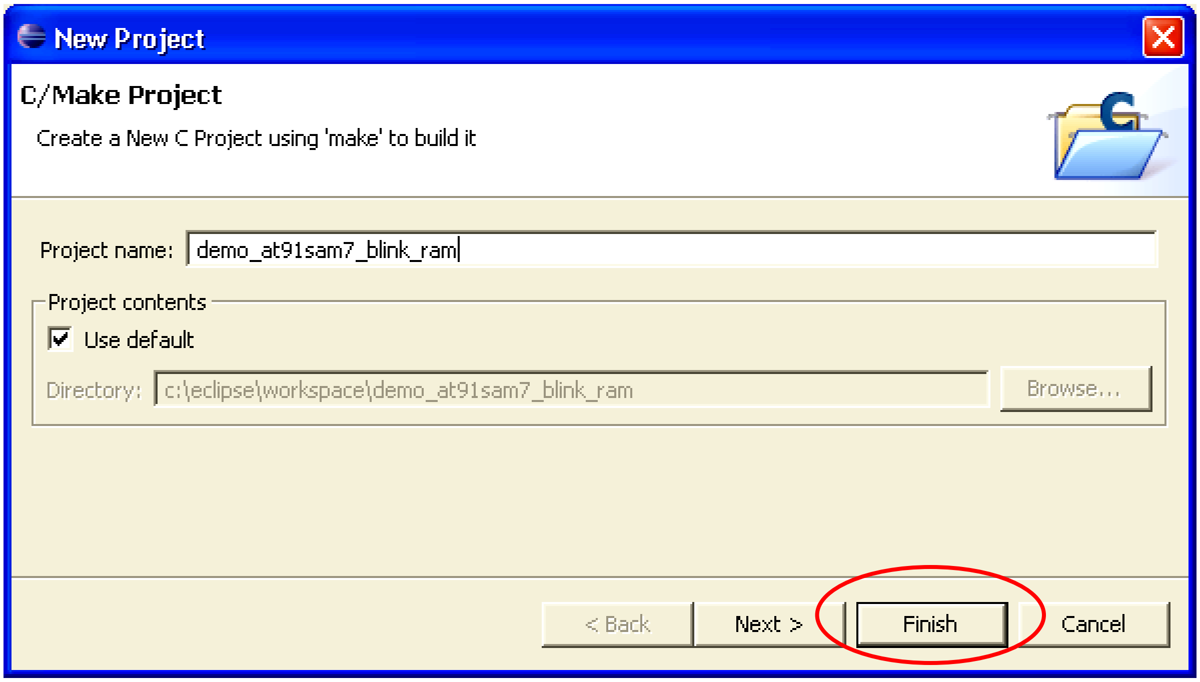
With this approach, we get an unlimited number of software breakpoints and can use the JTAG debugger interface to download the code (we don’t have to use the OpenOCD or SAM-BA flash programming facility). The disadvantage, of course, is that the application is limited to 64 Kbytes.

Close the current Eclipse project using the “**Project**” pull-down menu and then selecting “**Close Project**”.

Click on “**File – New – Standard Make C Project**” as shown below.



Give the new project the name “**demo_at91sam7_blink_ram**” and click “**Finish**”.



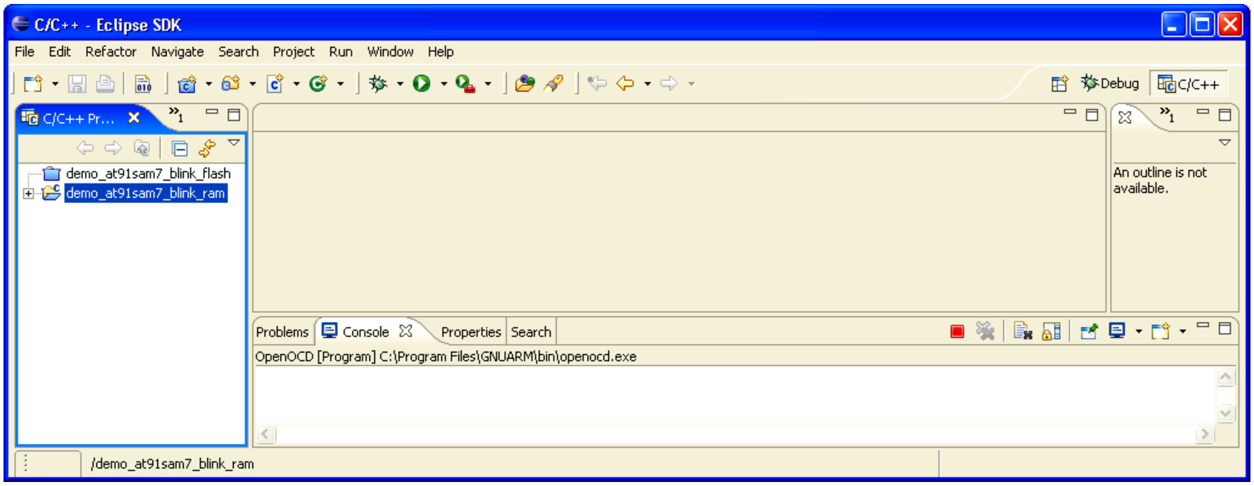
Project name: demo_at91sam7_blink_ram

Project contents

☑ Use default

Directory: c:\eclipse\workspace\demo_at91sam7_blink_ram

Now we have a project that has no files.



OpenOCD [Program] C:\Program Files\GNUARM\bin\openocd.exe

Now import the source files from the c:\download\atmel_tutorial_source\demo_at91sam7_blink_ram\ folder for the project **demo_at91sam7_blink_ram** using the techniques learned earlier.

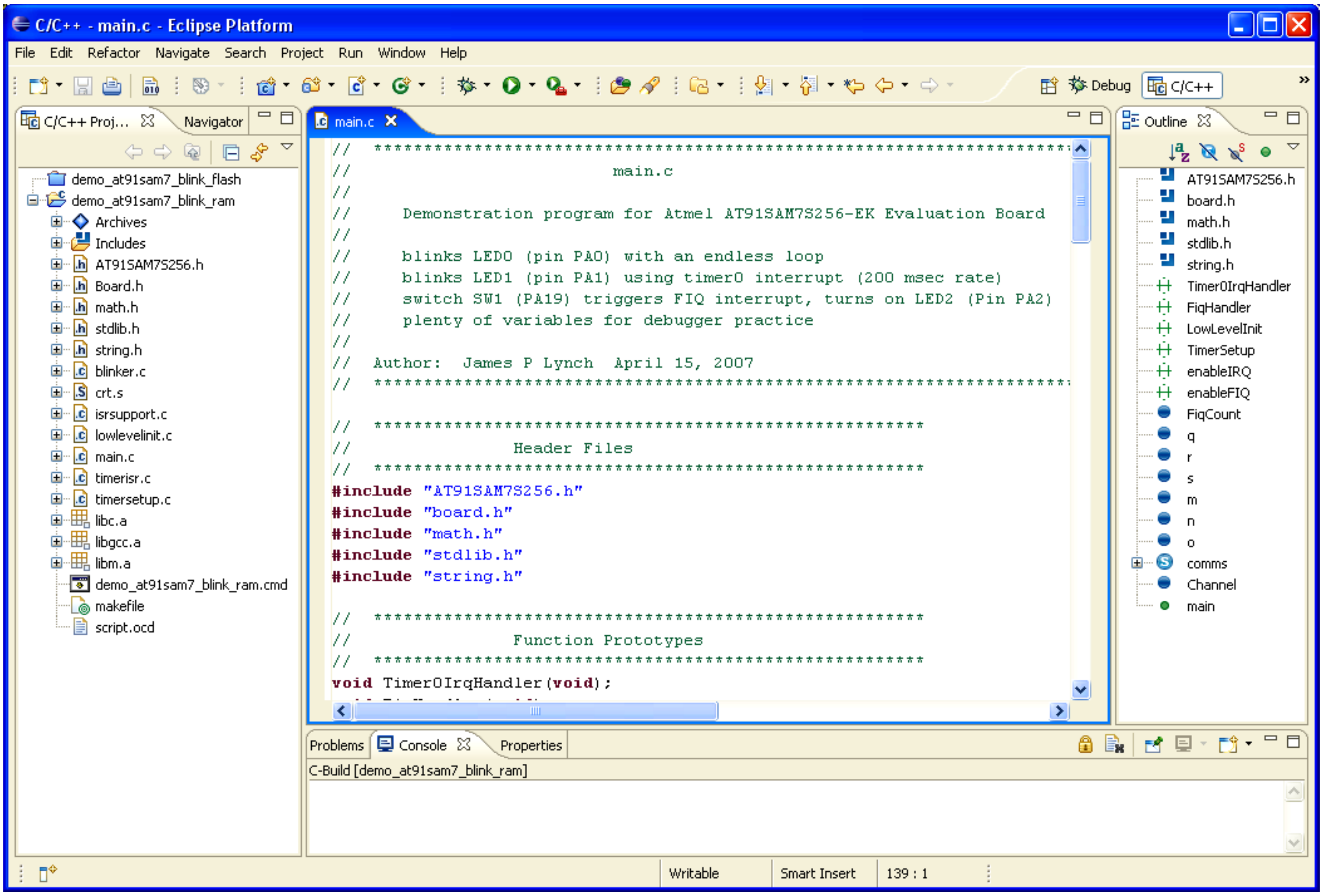
Only two files are different from the previous FLASH version:

demo_at91sam7_blink_ram.cmd - This file is different in that all code and variables are linked and loaded into address 0x00000000.

makefile.mak - this file references the file above (demo_at91sam7_blink_ram.cmd) so there are some minor edits therein.

All other files are exactly the same as the FLASH example.

Now we have a project with the proper files imported.



```
//   *****************************************************************************
//                               main.c
//
//      Demonstration program for Atmel AT91SAM7S256-EK Evaluation Board
//
//      blinks LED0 (pin PA0) with an endless loop
//      blinks LED1 (pin PA1) using timer0 interrupt (200 msec rate)
//      switch SW1 (PA19) triggers FIQ interrupt, turns on LED2 (Pin PA2)
//      plenty of variables for debugger practice
//
//   Author:  James P Lynch  April 15, 2007
//   *****************************************************************************

//   *******************************************************
//                  Header Files
//   *******************************************************
#include "AT91SAM7S256.h"
#include "board.h"
#include "math.h"
#include "stdlib.h"
#include "string.h"

//   *******************************************************
//                  Function Prototypes
//   *******************************************************
void Timer0IrqHandler(void);
```

C-Build [demo_at91sam7_blink_ram]

Only two files have changes and they are shown below. The few things that have been changed for RAM execution are colored in blue.

DEMO_AT91SAM7_BLINK_RAM.CMD

```
/* ***************************************************************************************************** */
/*   demo_at91sam7_blink_ram.cmd        LINKER  SCRIPT                                                   */
/*                                                                                                       */
/*                                                                                                       */
/*   The Linker Script defines how the code and data emitted by the GNU C compiler and assembler are     */
/*   to be loaded into memory (code goes into RAM, variables go into RAM).                               */
/*                                                                                                       */
/*   Any symbols defined in the Linker Script are automatically global and available to the rest of the  */
/*   program.                                                                                            */
/*                                                                                                       */
/*   To force the linker to use this LINKER SCRIPT, just add the -T demo_at91sam7_blink_ram.cmd          */
/*   directive to the linker flags in the makefile. For example,                                         */
/*                                                                                                       */
/*             LFLAGS  =  -Map main.map -nostartfiles -T demo_at91sam7_blink_ram.cmd                     */
/*                                                                                                       */
/*                                                                                                       */
/*   The order that the object files are listed in the makefile determines what .text section is         */
/*   placed first.                                                                                       */
/*                                                                                                       */
/*   For example:  $(LD) $(LFLAGS) -o main.out  crt.o main.o lowlevelinit.o                              */
/*                                                                                                       */
/*                crt.o is first in the list of objects, so it will be placed at address 0x00000000      */
/*                                                                                                       */
/*                                                                                                       */
/*   The top of the stack (_stack_end) is (last_byte_of_ram +1) - 4                                      */
/*                                                                                                       */
/*   Therefore:   _stack_end = (0x0000FFFF + 1) - 4  =  0x00010000 - 4  =  0x0000FFFC                    */
/*                                                                                                       */
/*                                                                                                         */
/*   Note that this symbol (_stack_end) is automatically GLOBAL and will be used by the crt.s              */
/*   startup assembler routine to specify all stacks for the various ARM modes                             */
/*                                                                                                         */
/*                              MEMORY MAP                                                                 */
/*                      |                                 |                                                */
/*            .-------->|---------------------------------|0x00010000                                      */
/*            .         |                                 |0x0000FFFC  <---------- _stack_end              */
/*            .         |    UDF Stack  16 bytes          |                                                */
/*            .         |                                 |                                                */
/*            .         |---------------------------------|0x0000FFEC                                      */
/*            .         |                                 |                                                */
/*            .         |    ABT Stack  16 bytes          |                                                */
/*            .         |                                 |                                                */
/*            .         |---------------------------------|0x0000FFDC                                      */
/*            .         |                                 |                                                */
/*            .         |                                 |                                                */
/*            .         |    FIQ Stack  128 bytes         |                                                */
/*            .         |                                 |                                                */
/*            .         |                                 |                                                */
/*            .         |---------------------------------|0x0000FF5C                                      */
/*            .         |                                 |                                                */
/*            .         |                                 |                                                */
/*            .         |    IRQ Stack  128 bytes         |                                                */
/*            .         |                                 |                                                */
/*            .         |                                 |                                                */
/*            .         |---------------------------------|0x0000FEDC                                      */
/*            .         |                                 |                                                */
/*            .         |    SVC Stack  16 bytes          |                                                */
/*            .         |                                 |                                                */
/*            .         |---------------------------------|0x0000FECC                                      */
/*            .         |                                 |                                                */
/*            .         |     stack area for user program |                                                */
/*            .         |                                 |                                                */
/*            .         |                                 |                                                */
/*            .         |                                 |                                                */
/*            .         |                                 |                                                */
/*            .         |                                 |                                                */
/*            .         |                                 |                                                */
/*            .         |                                 |                                                */
/*            .         |          free ram               |                                                */
/*           ram        |                                 |                                                */
/*            .         |                                 |                                                */
/*            .         |                                 |                                                */
/*            .         |.................................|0x00001398 <---------- _bss_end                 */
/*            .         |                                 |                                                */
/*            .         |  .bss   uninitialized variables |                                                */
/*            .         |.................................|0x00001380 <---------- _bss_start, _edata        */
/*            .         |                                 |                                                */
/*            .         |  .data  initialized variables   |                                                */
/*            .         |                                 |                                                */
/*           .          |---------------------------------|0x00000F3C <----------- _etext                  */
/*           .          |                                 |                                                */
/*           .          |                                 |                                                */
/*           .          |                                 |                                                */
/*           .          |            C code               |                                                */
/*           .          |                                 |                                                */
/*           .          |                                 |                                                */
/*           .          |                                 |                                                */
/*           .          |---------------------------------|0x0000015C  main()                              */
/*           .          |                                 |                                                */
/*           .          |    Startup Code  (crt.s)        |                                                */
/*           .          |         (assembler)             |                                                */
/*           .          |                                 |                                                */
/*           .          |---------------------------------|0x00000020                                      */
/*           .          |                                 |                                                */
/*           .          | Interrupt Vector Table          |                                                */
/*           .          |          32 bytes               |                                                */
/*           .--------->|---------------------------------|0x00000000 _vec_reset                           */
/*                                                                                                         */
/*                                                                                                         */
/*  Author:  James P. Lynch      May 12, 2007                                                              */
/*                                                                                                         */
/* ******************************************************************************************************* */


/* identify the Entry Point  (_vec_reset is defined in file crt.s)  */
ENTRY(_vec_reset)

/* specify the AT91SAM7S256 memory areas  */
MEMORY
{
    flash  : ORIGIN = 0,          LENGTH = 256K     /* FLASH EPROM    */
    ram    : ORIGIN = 0x00200000, LENGTH = 64K      /* static RAM area*/
}


/* define a global symbol _stack_end  (see analysis in annotation above) */
_stack_end = 0xFFFC;
```

```
/* now define the output sections  */
SECTIONS
{
    . = 0;                              /* set location counter to address zero  */

    .text :                             /* collect all sections that should go into FLASH after startup  */
    {
        *(.text)                        /* all .text sections (code)  */
        *(.rodata)                      /* all .rodata sections (constants, strings, etc.)  */
        *(.rodata*)                     /* all .rodata* sections (constants, strings, etc.)  */
        *(.glue_7)                      /* all .glue_7 sections  (no idea what these are) */
        *(.glue_7t)                     /* all .glue_7t sections (no idea what these are) */
        _etext = .;                     /* define a global symbol _etext just after the last code byte */
    } >ram                              /* put all the above into RAM */

    .data :                             /* collect all initialized .data sections that go into RAM  */
    {
        _data = .;                      /* create a global symbol marking the start of the .data section  */
        *(.data)                        /* all .data sections  */
        _edata = .;                     /* define a global symbol marking the end of the .data section  */
    } >ram                              /* put all the above into RAM */

    .bss :                              /* collect all uninitialized .bss sections that go into RAM  */
    {
        _bss_start = .;                 /* define a global symbol marking the start of the .bss section */
        *(.bss)                         /* all .bss sections  */
    } >ram                              /* put all the above in RAM (it will be cleared in the startup code */

    . = ALIGN(4);                       /* advance location counter to the next 32-bit boundary */
    _bss_end = . ;                      /* define a global symbol marking the end of the .bss section */
}
    _end = .;                           /* define a global symbol marking the end of application RAM */
```

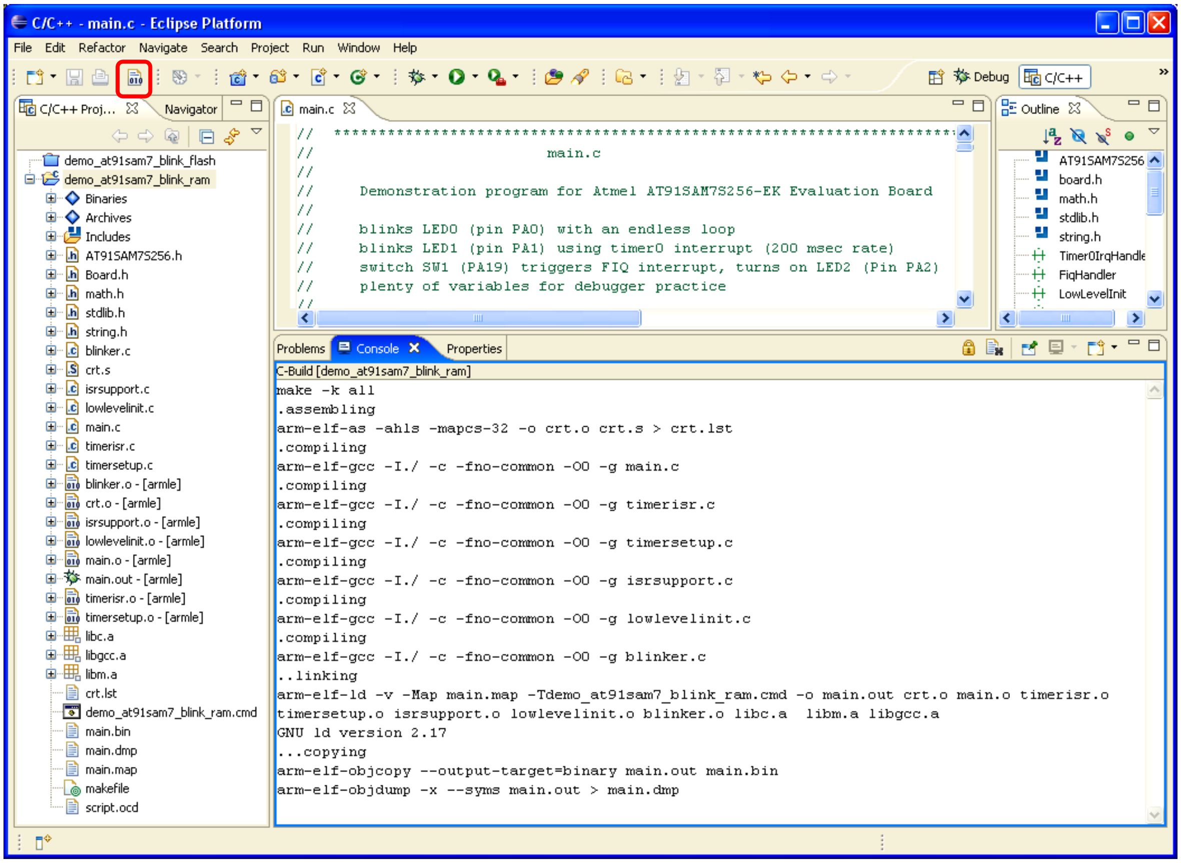
MAKEFILE.MAK

```
# ***************************************************************
# *     Makefile for Atmel AT91SAM7S256 - ram execution         *
# *                                                             *
# *                                                             *
# * James P Lynch  May 12, 2007                                 *
# ***************************************************************

NAME   = demo_at91sam7_blink_ram

# variables
CC      = arm-elf-gcc
LD      = arm-elf-ld -v
AR      = arm-elf-ar
AS      = arm-elf-as
CP      = arm-elf-objcopy
OD     = arm-elf-objdump

CFLAGS  = -I./ -c -fno-common -O0 -g
AFLAGS  = -ahls -mapcs-32 -o crt.o
LFLAGS  =  -Map main.map -Tdemo_at91sam7_blink_ram.cmd
CPFLAGS = --output-target=binary
ODFLAGS= -x --syms

OBJECTS = crt.o   main.o timerisr.o timersetup.o isrsupport.o lowlevelinit.o blinker.o


# make target called by Eclipse (Project -> Clean ...)
clean:
    -rm $(OBJECTS) crt.lst main.lst main.out main.bin main.hex main.map main.dmp


#make target called by Eclipse  (Project -> Build Project)
all:  main.out
    @ echo "...copying"
    $(CP) $(CPFLAGS) main.out main.bin
    $(OD) $(ODFLAGS) main.out > main.dmp
```

```
main.out: $(OBJECTS) demo_at91sam7_blink_ram.cmd
    @ echo "..linking"
    $(LD) $(LFLAGS) -o main.out $(OBJECTS) libc.a  libm.a libgcc.a

crt.o: crt.s
    @ echo ".assembling"
    $(AS) $(AFLAGS) crt.s > crt.lst

main.o: main.c
    @ echo ".compiling"
    $(CC) $(CFLAGS) main.c

timerisr.o: timerisr.c
    @ echo ".compiling"
    $(CC) $(CFLAGS) timerisr.c

lowlevelinit.o: lowlevelinit.c
    @ echo ".compiling"
    $(CC) $(CFLAGS) lowlevelinit.c

timersetup.o: timersetup.c
    @ echo ".compiling"
    $(CC) $(CFLAGS) timersetup.c

isrsupport.o: isrsupport.c
    @ echo ".compiling"
    $(CC) $(CFLAGS) isrsupport.c

blinker.o: blinker.c
    @ echo ".compiling"
    $(CC) $(CFLAGS) blinker.c
```

Build the RAM Project

Using the “**Build All**” button, build the new RAM Project.



```
//  ***********************************************************************
//                              main.c
//
//     Demonstration program for Atmel AT91SAM7S256-EK Evaluation Board
//
//     blinks LED0 (pin PA0) with an endless loop
//     blinks LED1 (pin PA1) using timer0 interrupt (200 msec rate)
//     switch SW1 (PA19) triggers FIQ interrupt, turns on LED2 (Pin PA2)
//     plenty of variables for debugger practice
//
```

C-Build [demo_at91sam7_blink_ram]

```
make -k all
.assembling
arm-elf-as -ahls -mapcs-32 -o crt.o crt.s > crt.lst
.compiling
arm-elf-gcc -I./ -c -fno-common -O0 -g main.c
.compiling
arm-elf-gcc -I./ -c -fno-common -O0 -g timerisr.c
.compiling
arm-elf-gcc -I./ -c -fno-common -O0 -g timersetup.c
.compiling
arm-elf-gcc -I./ -c -fno-common -O0 -g isrsupport.c
.compiling
arm-elf-gcc -I./ -c -fno-common -O0 -g lowlevelinit.c
.compiling
arm-elf-gcc -I./ -c -fno-common -O0 -g blinker.c
..linking
arm-elf-ld -v -Map main.map -Tdemo_at91sam7_blink_ram.cmd -o main.out crt.o main.o timerisr.o
timersetup.o isrsupport.o lowlevelinit.o blinker.o libc.a  libm.a libgcc.a
GNU ld version 2.17
...copying
arm-elf-objcopy --output-target=binary main.out main.bin
arm-elf-objdump -x --syms main.out > main.dmp
```

In this version, we will be using the “**main.out**” file to download the executable into RAM via the JTAG.

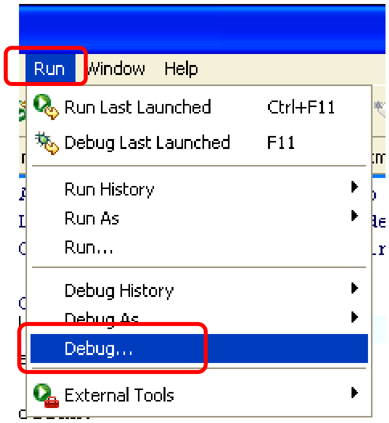
Debugging the RAM Application

Debugging an application loaded entirely into RAM is very similar to debugging in Flash. The advantage is that you have an unlimited number of software breakpoints and the application is automatically loaded into RAM at debugger startup.

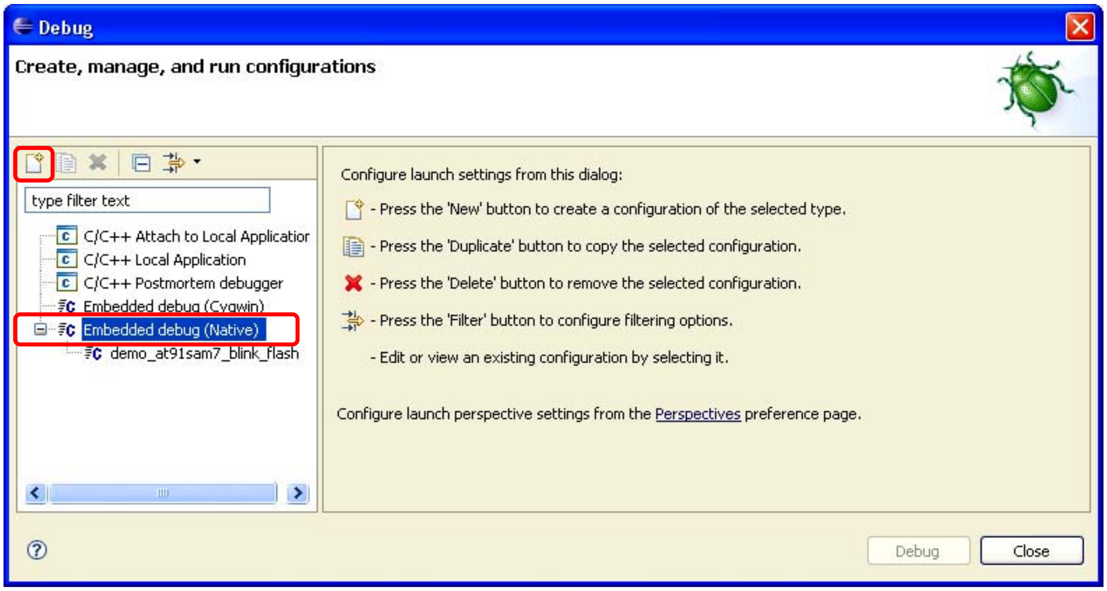
Create an Embedded Debug Launch Configuration for RAM

A separate Debug Launch Configuration is appropriate since the debugger startup script will be different and the downloading of executable code into RAM will be performed by the JTAG hardware interface.

Click on “**Run**” followed by “**Debug…**”.



When the Debug “Create, manage, and run configurations” screen appears, click on “**Embedded debug (Native)**” followed by the “**New**” button.

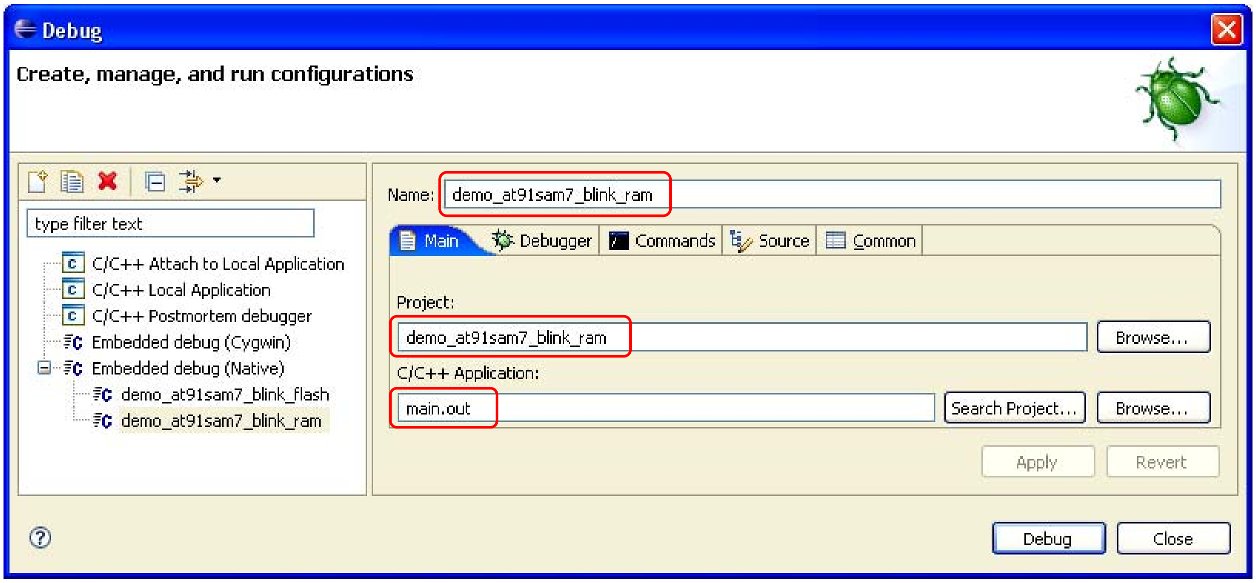

Debug
Create, manage, and run configurations
type filter text
C/C++ Attach to Local Applicatior
C/C++ Local Application
C/C++ Postmortem debugger
Embedded debug (Cygwin)
Embedded debug (Native)
demo_at91sam7_blink_flash
Debug
Close

Configure launch settings from this dialog:

- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.

Configure launch perspective settings from the Perspectives preference page.

A new and empty "Embedded debug launch" configuration screen will appear. Under the "Main" tab, fill out the new configuration screen as shown below. Once again, I selected the project name "**demo_at91sam7_blink_ram**" as the debug launch configuration name. Use the "**Browse**" buttons to find the project and the C/C++ Application file as shown below.



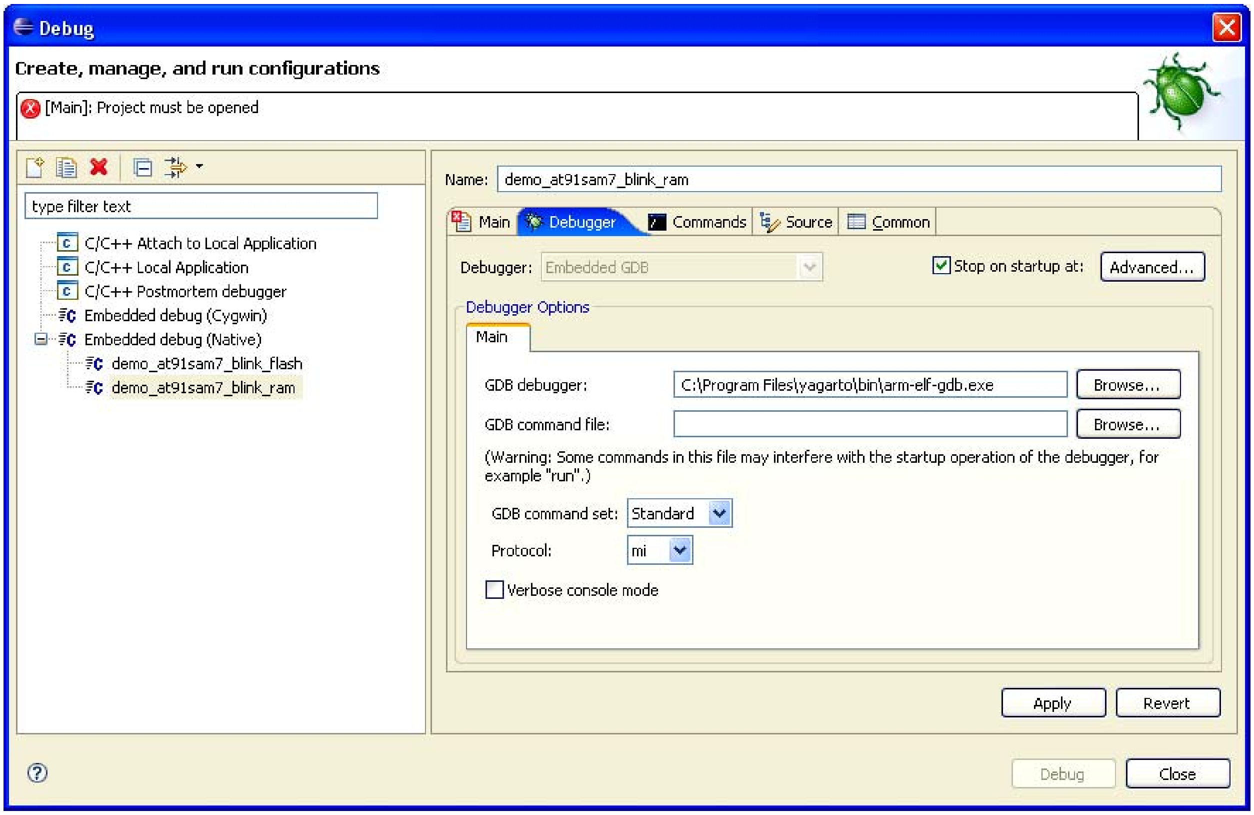
Name: demo_at91sam7_blink_ram

Project: demo_at91sam7_blink_ram

C/C++ Application: main.out

Under the "Debugger" tab, fill out the screen as shown below. Note that we checked the "**Stop on startup at:**" check box so that the debugger will stop at the entry point of main( ).

Also use the "**Browse**" button to find the GDB debugger (it is the file: **c:\Program Files\yagarto\bin\arm-elf-gdb.exe**).



[Main]: Project must be opened

Name: demo_at91sam7_blink_ram

Debugger: Embedded GDB

☑ Stop on startup at:

GDB debugger: C:\Program Files\yagarto\bin\arm-elf-gdb.exe

GDB command file:

(Warning: Some commands in this file may interfere with the startup operation of the debugger, for example "run".)

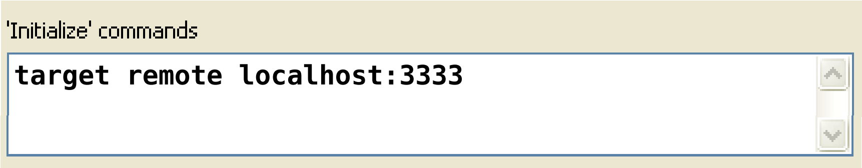
GDB command set: Standard

Protocol: mi

☐ Verbose console mode

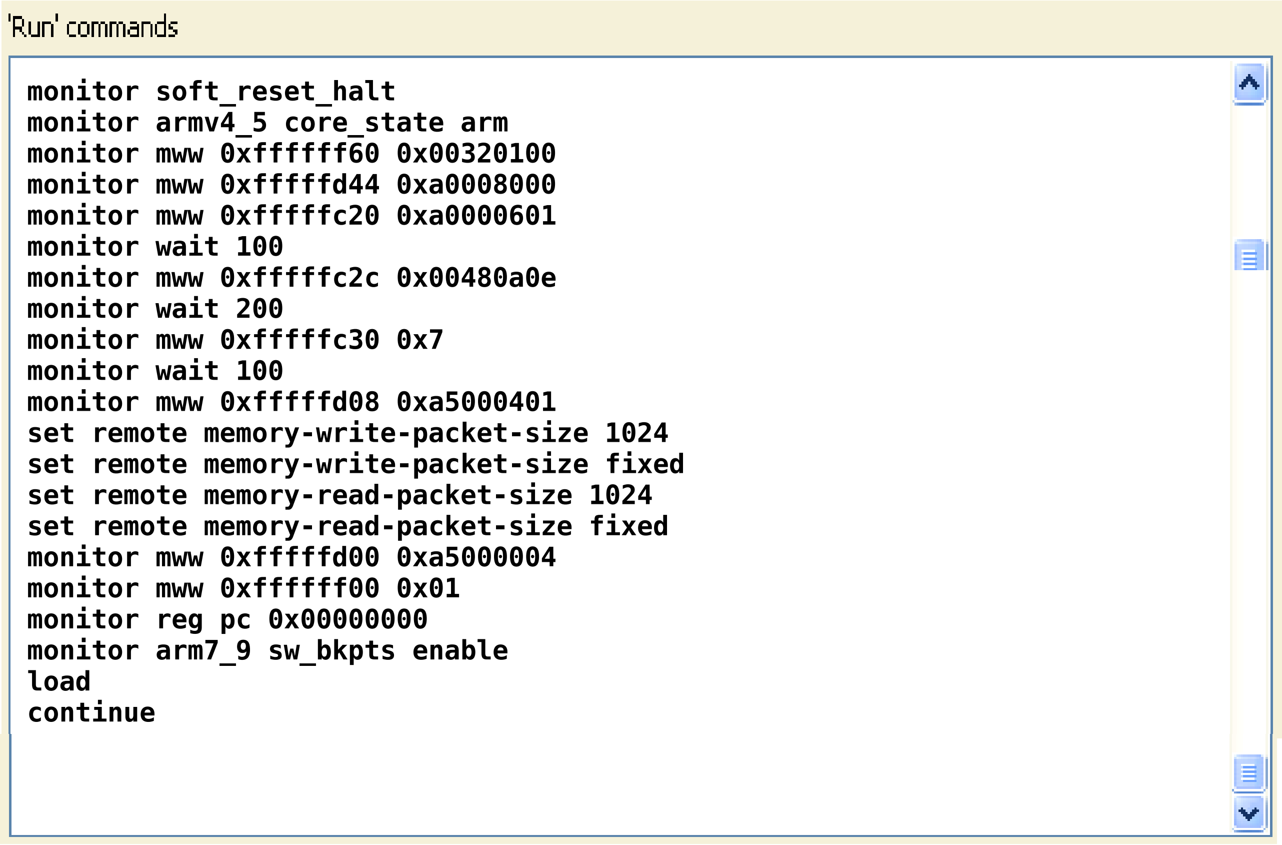
Now select the “Comands” tab as shown below.

If you are using OpenOCD, enter the single GDB command “**target remote localhost:3333**” in the “Initialize commands” text window exactly as shown below. This command tells the GDB debugger to emit commands in RSP format to the TCP port “localhost:3333” (the port OpenOCD will be listening to).



```
target remote localhost:3333
```

If you are using OpenOCD, enter the following GDB and OpenOCD commands into the “Run commands” text window, exactly as shown below. The “Source” and “Common” tabs can be left in their default state.

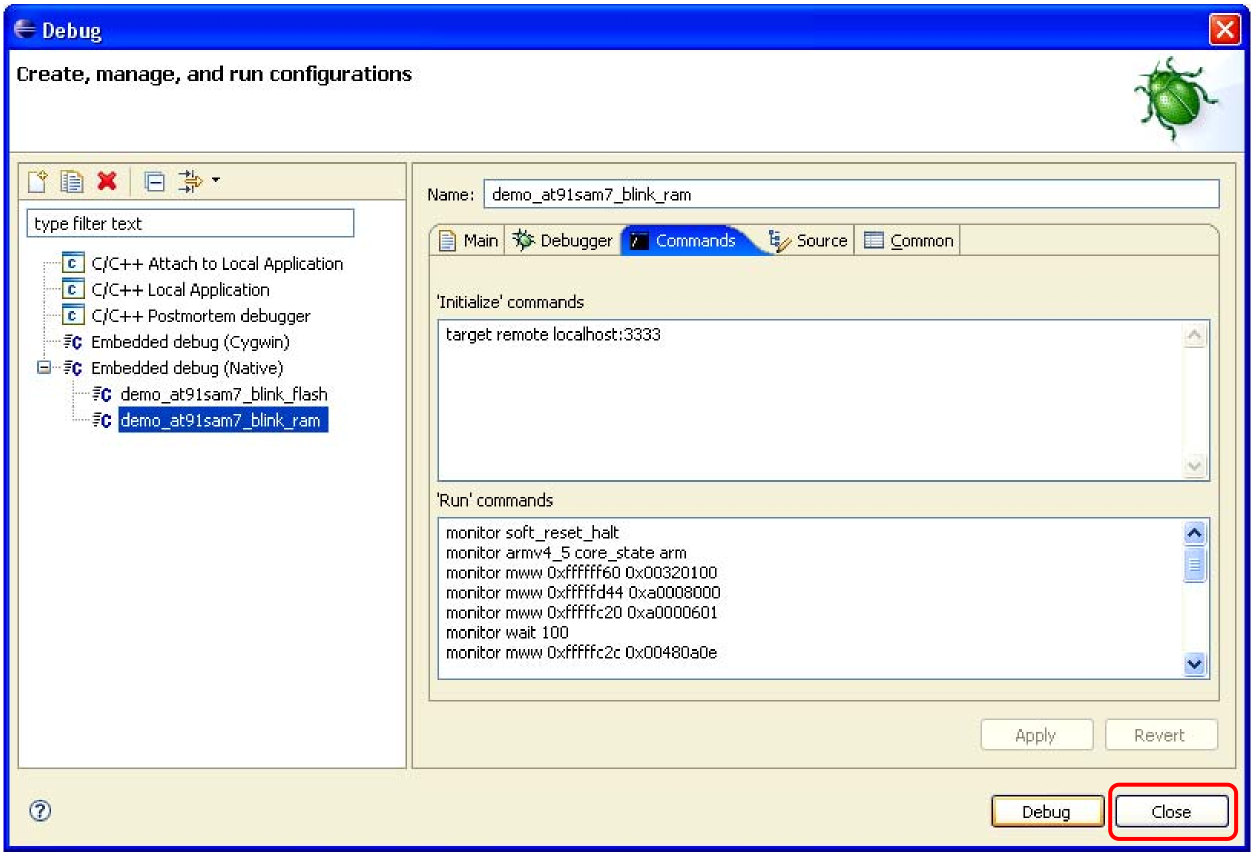


```
monitor soft_reset_halt
monitor armv4_5 core_state arm
monitor mww 0xffffff60 0x00320100
monitor mww 0xfffffd44 0xa0008000
monitor mww 0xfffffc20 0xa0000601
monitor wait 100
monitor mww 0xfffffc2c 0x00480a0e
monitor wait 200
monitor mww 0xfffffc30 0x7
monitor wait 100
monitor mww 0xfffffd08 0xa5000401
set remote memory-write-packet-size 1024
set remote memory-write-packet-size fixed
set remote memory-read-packet-size 1024
set remote memory-read-packet-size fixed
monitor mww 0xfffffd00 0xa5000004
monitor mww 0xffffff00 0x01
monitor reg pc 0x00000000
monitor arm7_9 sw_bkpts enable
load
continue
```

Below is the Debug Launch Configuration “Commands” tab for use with OpenOCD and flash execution. Note that the ‘Run’ commands window below only shows a portion of the commands that were entered. Be sure to enter all the commands as shown above.

The “Source” and “Common” tabs can be left in their default condition. Click on “**Close**” to complete definition of the Debug Launch Configuration for flash debugging with OpenOCD.

Author’s Note: GDB manual states “Any text from a # to the end of a line is a comment; it does nothing”. Unfortunately, I’ve noted that these systems get tripped up occasionally by these comments so they have been left out of all debug windows.



Name: demo_at91sam7_blink_ram

'Initialize' commands

```
target remote localhost:3333
```

'Run' commands

```
monitor soft_reset_halt
monitor armv4_5 core_state arm
monitor mww 0xffffff60 0x00320100
monitor mww 0xfffffd44 0xa0008000
monitor mww 0xfffffc20 0xa0000601
monitor wait 100
monitor mww 0xfffffc2c 0x00480a0e
```

To make entry of the 'Run' commands more convenient, here is a list of the commands that can be used for "cut-and-paste" transfer to Eclipse.

```
monitor soft_reset_halt
monitor armv4_5 core_state arm
monitor mww 0xffffff60 0x00320100
monitor mww 0xfffffd44 0xa0008000
monitor mww 0xfffffc20 0xa0000601
monitor wait 100
monitor mww 0xfffffc2c 0x00480a0e
monitor wait 200
monitor mww 0xfffffc30 0x7
monitor wait 100
monitor mww 0xfffffd08 0xa5000401
set remote memory-write-packet-size 1024
set remote memory-write-packet-size fixed
set remote memory-read-packet-size 1024
set remote memory-read-packet-size fixed
monitor mww 0xfffffd00 0xa5000004
monitor mww 0xffffff00 0x01
monitor reg pc 0x00000000
monitor arm7_9 sw_bkpts enable
load
continue
```

Copy these commands into the "Run Commands" window.

The GDB startup commands for OpenOCD operation shown above require some explanation. If the command line starts with the word "monitor", then that command is an OpenOCD command. Otherwise, it is a legacy GDB command.

OpenOCD commands are described in the OpenOCD documentation which can be downloaded from:

http://developer.berlios.de/docman/display_doc.php?docid=1367&group_id=4148

GDB commands are described in several books and in the official document that can be downloaded from:
http://dsl.ee.unsw.edu.au/dsl-cdrom/gnutools/doc/gnu-debugger.pdf

First, we have to halt the processor.

```
monitor soft_reset_halt                   # OpenOCD command to halt the processor and wait
```

Next, we identify the ARM core being used

```
monitor armv4_5 core_state arm            # OpenOCD command to select the core state
```

Now we set up the processor's clocks, etc. using the register settings in the lowlevelinit.c function. These are OpenOCD memory write commands used to set the various AT91SAM7S256 clock registers. This guarantees that the processor will be running at full speed when the "continue" command is asserted.

```
monitor mww 0xffffff60 0x00320100         # set flash wait state (AT91C_MC_FMR)
monitor mww 0xfffffd44 0xa0008000         # watchdog disable (AT91C_WDTC_WDMR)
monitor mww 0xfffffc20 0xa0000601         # enable main oscillator (AT91C_PMC_MOR)
monitor wait 100                          # wait 100 ms
monitor mww 0xfffffc2c 0x00480a0e         # set PLL register (AT91C_PMC_PLLR)
monitor wait 200                          # wait 200 ms
monitor mww 0xfffffc30 0x7                # set master clock to PLL (AT91C_PMC_MCKR)
monitor wait 100                          # wait 100 ms
```

Enable the Reset button in the AT91SAM7S-EK board.

```
monitor mww 0xfffffd08 0xa5000401         # enable user reset AT91C_RSTC_RMR
```

Now increase the GDB packet size to 1024. This will have a slight improvement on FLASH debugging as reads of large data structures, etc. may be speeded up. These are legacy GDB commands.

```
set remote memory-write-packet-size 1024  # Setup GDB for faster downloads
set remote memory-write-packet-size fixed # Setup GDB for faster downloads
set remote memory-read-packet-size 1024   # Setup GDB for faster downloads
set remote memory-read-packet-size fixed  # Setup GDB for faster downloads
```

This is an OpenOCD command to force a peripheral reset. This guarantees that the next command (set MC Remap Control register to 1) starts from a known initial state (MC Remap Control Register is a "toggle" action).

```
monitor mww 0xfffffd00 0xa5000004         #  force a peripheral RESET  AT91C_RSTC_RCR
```

This OpenOCD command sets the AT91SAM7S256 MC Remap Control register to 1 which toggles the remap state. This action effectively overlays RAM memory on top of low memory at address 0x00000000.

```
monitor mww 0xffffff00 0x01               # toggle the remap register to place RAM at 0x00000000
```

This OpenOCD command sets the PC to the reset vector address 0x00000000

```
monitor reg pc 0x00000000                 # set the PC to 0x00000000
```

This is an OpenOCD command to enable software breakpoints.

```
monitor arm7_9 sw_bkpts enable            # enable use of software breakpoints
```

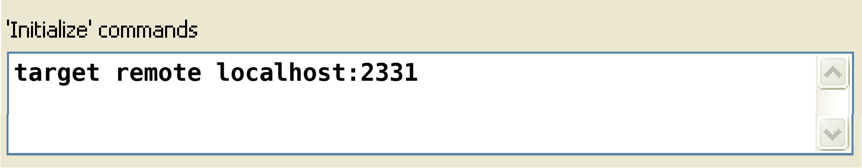
Now we load the application into RAM. This is a legacy GDB command.

```
load                                      # download the application using file main.out
```

Finally we emit the legacy GDB command "continue". The processor was already halted at the Reset vector and will thus start executing until it hits the breakpoint set at main( ).

```
continue                                  # resume execution from reset vector - will break at main( )
```

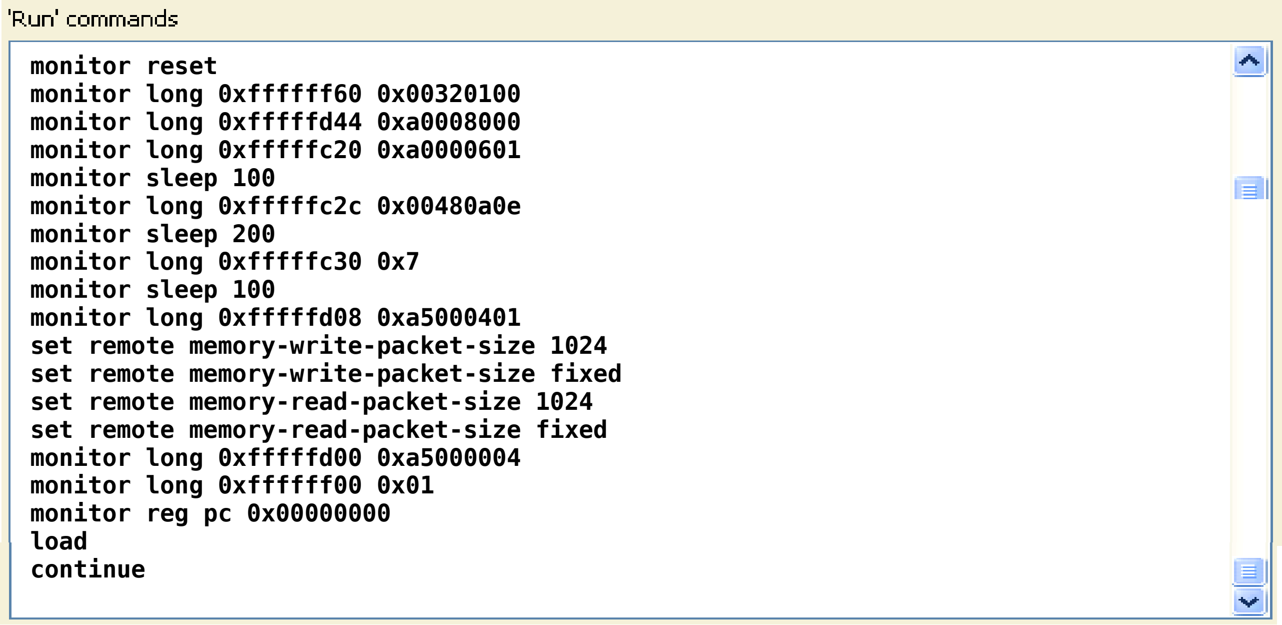
If you are using the J-Link GDB Server, enter the single GDB command “**target remote localhost:2331**” in the “Initialize commands” text window exactly as shown below. This command tells the GDB debugger to emit commands in RSP format to the TCP port “localhost:2331” (the port the J-Link GDB Server will be listening to).



'Initialize' commands

```
target remote localhost:2331
```

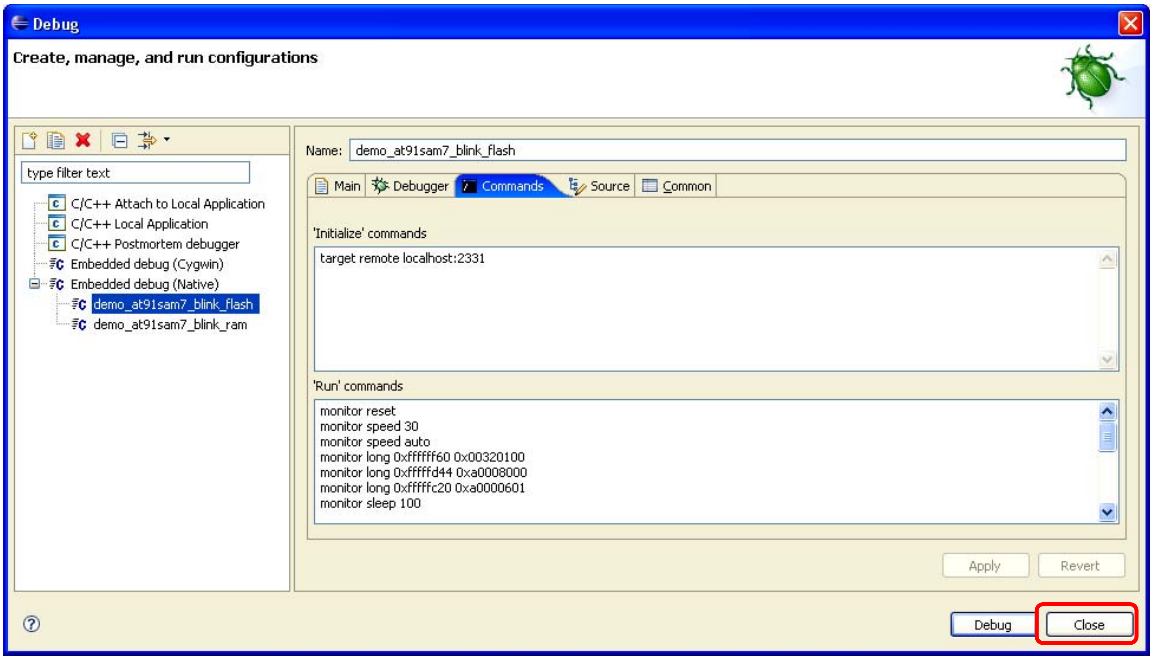
If you are using the J-Link GDB Server, enter the following GDB and J-Link GDB Server commands into the “Run commands” text window, exactly as shown below. The “Source” and “Common” tabs can be left in their default state.



'Run' commands

```
monitor reset
monitor long 0xffffff60 0x00320100
monitor long 0xfffffd44 0xa0008000
monitor long 0xfffffc20 0xa0000601
monitor sleep 100
monitor long 0xfffffc2c 0x00480a0e
monitor sleep 200
monitor long 0xfffffc30 0x7
monitor sleep 100
monitor long 0xfffffd08 0xa5000401
set remote memory-write-packet-size 1024
set remote memory-write-packet-size fixed
set remote memory-read-packet-size 1024
set remote memory-read-packet-size fixed
monitor long 0xfffffd00 0xa5000004
monitor long 0xffffff00 0x01
monitor reg pc 0x00000000
load
continue
```

Below is the Debug Launch Configuration “Commands” tab for use with the J-Link GDB Server and FLASH execution. Note that the ‘Run’ commands window only shows a portion of the commands that were entered. Be sure to enter all the commands as shown above.



Name: demo_at91sam7_blink_flash

'Initialize' commands

```
target remote localhost:2331
```

'Run' commands

```
monitor reset
monitor speed 30
monitor speed auto
monitor long 0xffffff60 0x00320100
monitor long 0xfffffd44 0xa0008000
monitor long 0xfffffc20 0xa0000601
monitor sleep 100
```

Click on “**Close**” above to complete definition of the Debug Launch Configuration for flash debugging with the J-Link GDB Server.

To make entry of the ‘Run’ commands more convenient, here is a list of them for “cut-and-paste” transfer to Eclipse.

```
monitor reset
monitor long 0xffffff60 0x00320100
monitor long 0xfffffd44 0xa0008000
monitor long 0xfffffc20 0xa0000601
monitor sleep 100
monitor long 0xfffffc2c 0x00480a0e
monitor sleep 200
monitor long 0xfffffc30 0x7
monitor sleep 100
monitor long 0xfffffd08 0xa5000401
set remote memory-write-packet-size 1024
set remote memory-write-packet-size fixed
set remote memory-read-packet-size 1024
set remote memory-read-packet-size fixed
monitor long 0xfffffd00 0xa5000004
monitor long 0xffffff00 0x01
monitor reg pc 0x00000000
load
continue
```

Copy these commands into the “Run Commands” window.

The GDB startup commands for the J-Link GDB Server operation shown above require some explanation. If the command line starts with the word “monitor”, then that command is a J-Link GDB Server command. Otherwise, it is a legacy GDB command.

J-Link GDB Server commands are described in the document “JLinkGDBServer.pdf” which is in the Segger documentation folder that you downloaded (“c:\Program Files\SEGGER\JLinkARM_V368b\Doc\Manuals\”)

GDB commands are described in several books and in the official document that can be downloaded from:
http://dsl.ee.unsw.edu.au/dsl-cdrom/gnutools/doc/gnu-debugger.pdf

First, we have to halt the processor.

```
monitor reset                              # Reset the chip to get to a known state.
```

Next, we set up the JTAG speed

```
monitor speed 30                           # Set JTAG speed to 30 kHz
monitor speed auto                         # Set auto JTAG speed
```

Now we set up the processor’s clocks, etc. using the register settings in the lowlevelinit.c function. These are J-Link GDB Server memory write commands used to set the various AT91SAM7S256 clock registers. This guarantees that the processor will be running at full speed when the “continue” command is asserted.

```
monitor long 0xffffff60 0x00320100         # set flash wait state (AT91C_MC_FMR)
monitor long 0xfffffd44 0xa0008000         # watchdog disable (AT91C_WDTC_WDMR)
monitor long 0xfffffc20 0xa0000601         # enable main oscillator (AT91C_PMC_MOR)
monitor sleep 100                          # wait 100 ms
monitor long 0xfffffc2c 0x00480a0e         # set PLL register (AT91C_PMC_PLLR)
monitor sleep 200                          # wait 200 ms
monitor long 0xfffffc30 0x7                # set master clock to PLL (AT91C_PMC_MCKR)
monitor sleep 100                          # wait 100 ms
```

Enable the Reset button in the AT91SAM7S-EK board.

```
    monitor long 0xfffffd08 0xa5000401          # enable user reset AT91C_RSTC_RMR
```

Now increase the GDB packet size to 1024. This will have a slight improvement on FLASH debugging as reads of large data structures, etc. may be speeded up. These are legacy GDB commands.

```
    set remote memory-write-packet-size 1024    # Setup GDB for faster downloads
    set remote memory-write-packet-size fixed   # Setup GDB for faster downloads
    set remote memory-read-packet-size 1024     # Setup GDB for faster downloads
    set remote memory-read-packet-size fixed    # Setup GDB for faster downloads
```

This is an OpenOCD command to force a peripheral reset. This guarantees that the next command (set MC Remap Control register to 1) starts from a known initial state (MC Remap Control Register is a “toggle” action).

```
    monitor long 0xfffffd00 0xa5000004          #  force a peripheral RESET  AT91C_RSTC_RCR
```

This OpenOCD command sets the AT91SAM7S256 MC Remap Control register to 1 which toggles the remap state. This action effectively overlays RAM memory on top of low memory at address 0x00000000.

```
    monitor long 0xffffff00 0x01                # toggle the remap register to place RAM at 0x00000000
```

This command sets the PC to the reset vector address 0x00000000

```
    monitor reg pc 0x00000000                   # set the PC to 0x00000000
```

Now we load the application into RAM. This is a legacy GDB command.

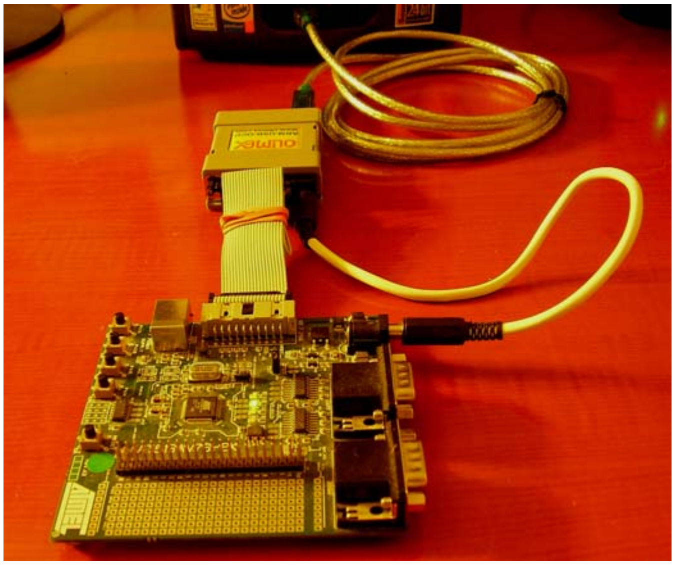
```
    load                                        # download the application using file main.out
```

Finally we emit the legacy GDB command “continue”. The processor was already halted at the Reset vector and will thus start executing until it hits the breakpoint set at main( ).

```
    continue                                    # resume execution from reset vector - will break at main( )
```

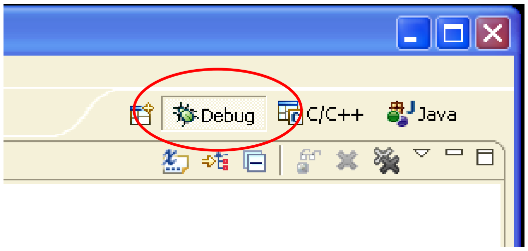
Set up the hardware

Whatever debugger you are using (SAM-ICE, wiggler, JTAGKey or ARM-USB-OCD), the same hardware setup used for FLASH programming and debugging will also apply to RAM-based applications. Shown below is the hardware setup for the Olimex ARM-USB-OCD JTAG debugger.

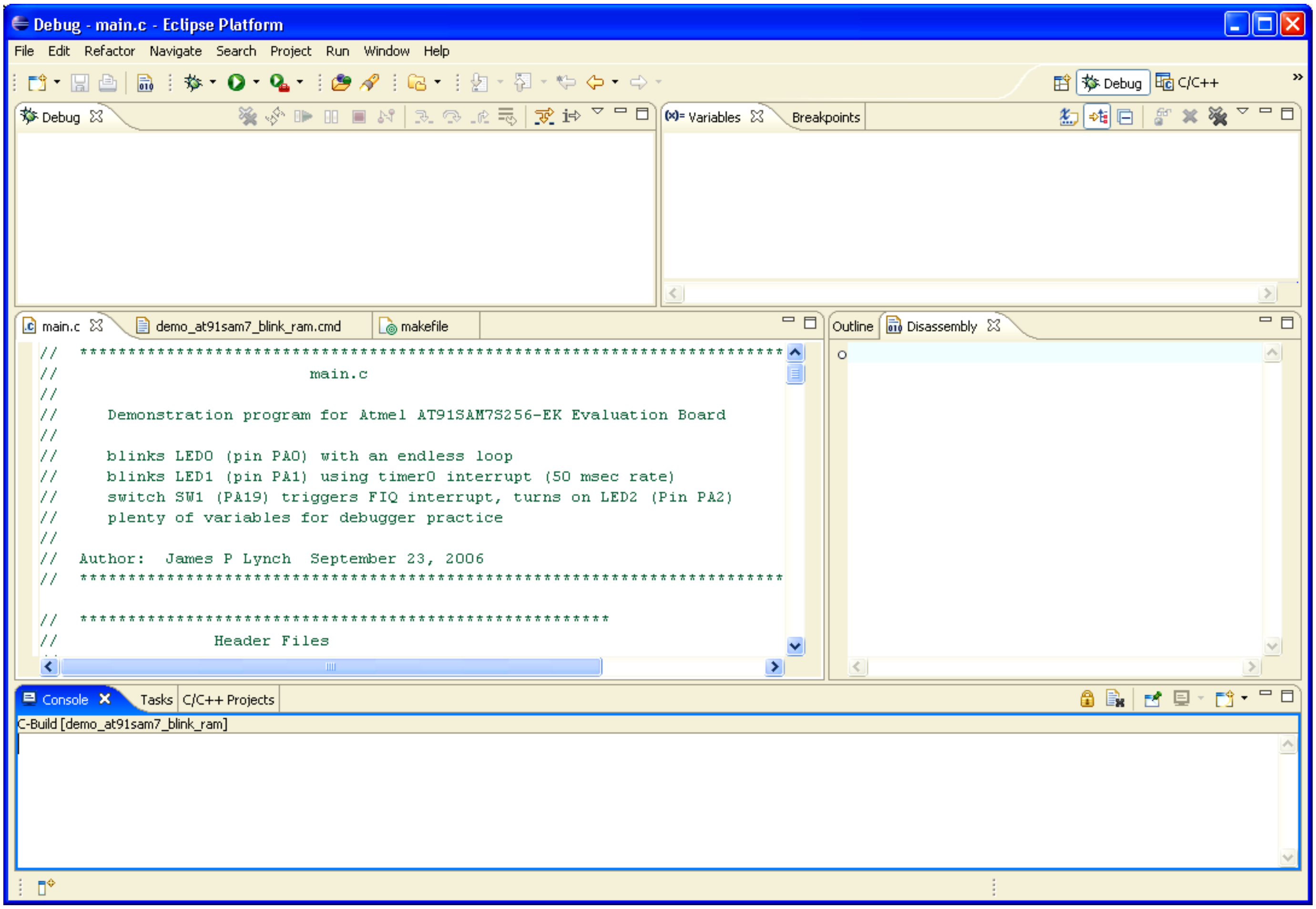


Open the Eclipse "Debug" Perspective

As shown earlier, click on the "**Debug**" perspective button located at the upper right part of the Eclipse screen.



Now the Debug perspective will appear, as shown below.


Debug - main.c - Eclipse Platform
File Edit Refactor Navigate Search Project Run Window Help
Debug
C/C++
Debug
(x)= Variables
Breakpoints
main.c
demo_at91sam7_blink_ram.cmd
makefile
Outline
Disassembly
Console
Tasks
C/C++ Projects

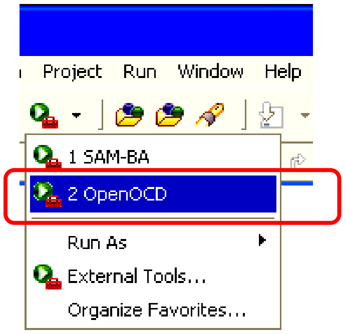
```
//   ****************************************************************************
//                              main.c
//
//     Demonstration program for Atmel AT91SAM7S256-EK Evaluation Board
//
//     blinks LED0 (pin PA0) with an endless loop
//     blinks LED1 (pin PA1) using timer0 interrupt (50 msec rate)
//     switch SW1 (PA19) triggers FIQ interrupt, turns on LED2 (Pin PA2)
//     plenty of variables for debugger practice
//
//   Author:  James P Lynch  September 23, 2006
//   ****************************************************************************

//   ***********************************************************
//                Header Files
```

```
C-Build [demo_at91sam7_blink_ram]
```

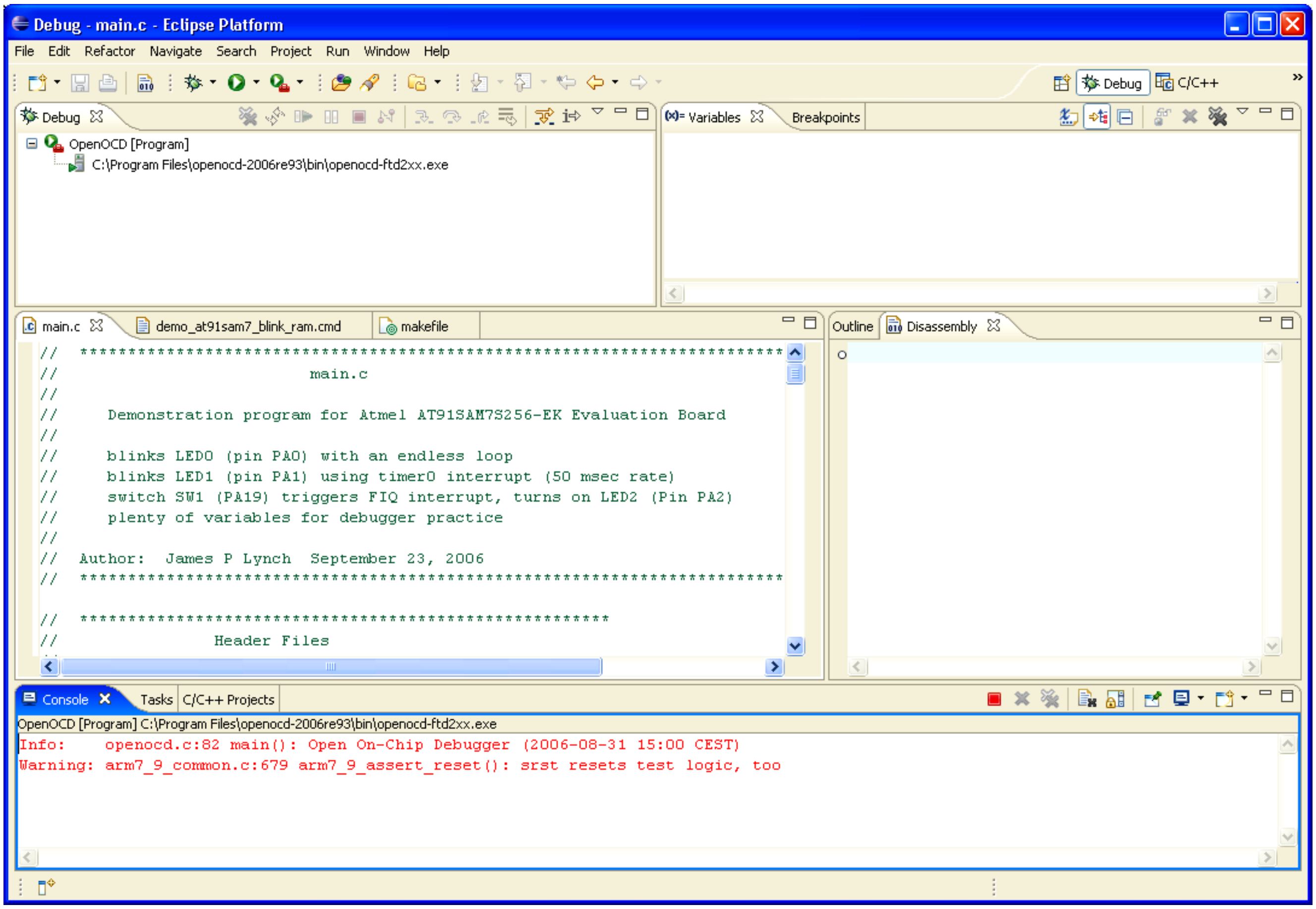
Start OpenOCD

If you have the Olimex or Amontec JTAG hardware interfaces, OpenOCD must be started before launching the Eclipse debugger.

To start OpenOCD, click on the "**External Tools**" toolbar button's down arrowhead and then select "**OpenOCD**". Alternatively, you can click on the "**Run**" pull-down menu and select "**External Tools**" followed by "**OpenOCD**".



The debug view will show that OpenOCD is running and the console view shows no errors (warnings are OK).



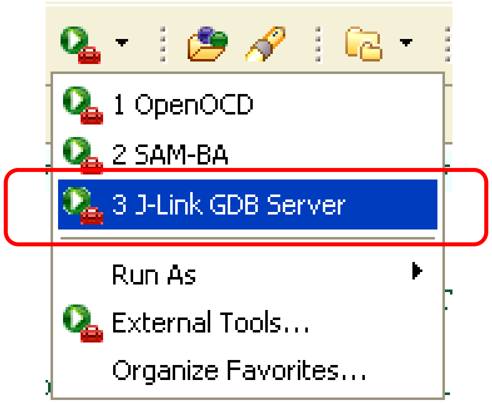
```
//  ************************************************************************
//                         main.c
//
//     Demonstration program for Atmel AT91SAM7S256-EK Evaluation Board
//
//     blinks LED0 (pin PA0) with an endless loop
//     blinks LED1 (pin PA1) using timer0 interrupt (50 msec rate)
//     switch SW1 (PA19) triggers FIQ interrupt, turns on LED2 (Pin PA2)
//     plenty of variables for debugger practice
//
//  Author:  James P Lynch  September 23, 2006
//  ************************************************************************

//  ******************************************************
//                Header Files
```

```
OpenOCD [Program] C:\Program Files\openocd-2006re93\bin\openocd-ftd2xx.exe
Info:    openocd.c:82 main(): Open On-Chip Debugger (2006-08-31 15:00 CEST)
Warning: arm7_9_common.c:679 arm7_9_assert_reset(): srst resets test logic, too
```

Start J-Link GDB Server

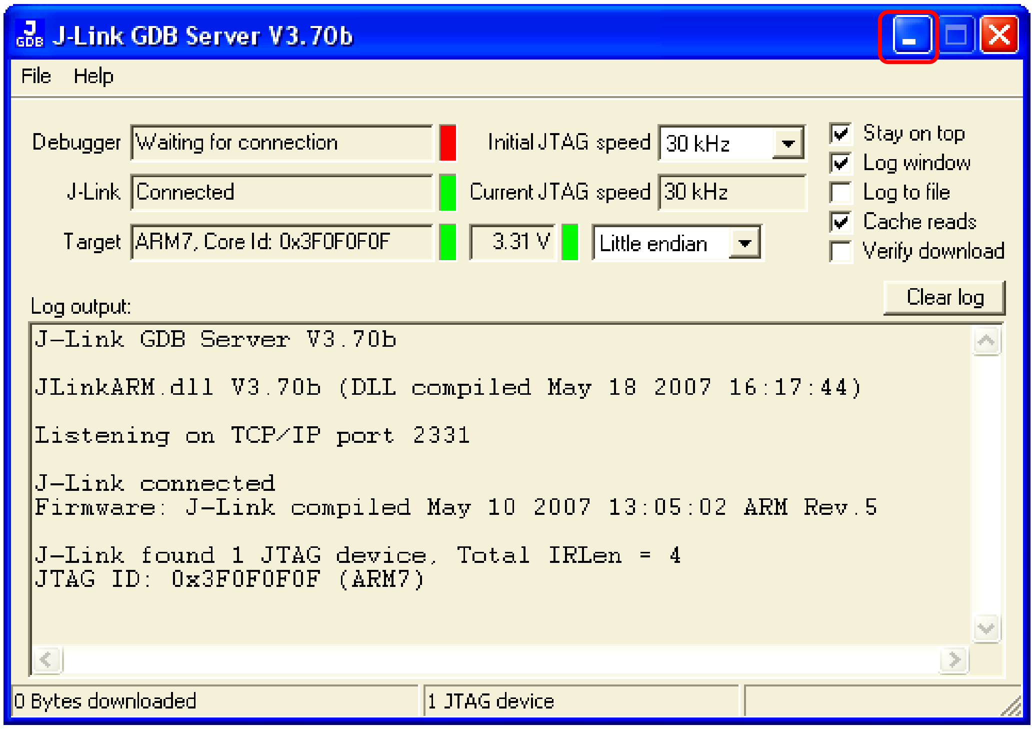
If you have the Atmel SAM-ICE JTAG hardware interface, the J-Link GDB Server must be started before launching the Eclipse debugger.

To start the J-Link GDB Server, click on the “**External Tools**” toolbar button’s down arrowhead and then select “**J-Link GDB Server**”. Alternatively, you can click on the “**Run**” pull-down menu and select “**External Tools**” followed by “**J-Link GDB Server**”.



First, a Segger J-Link GDB Server status window will appear as shown below. Notice that the green indicators show that the J-Link GDB Server is connected to your SAM-ICE and the target microprocessor core has been identified. The Debugger status light is indicating red; this is OK since we haven’t launched our Eclipse/GDB integrated graphical debugger yet. You should now minimize the Segger status display.

Whatever you do, don’t click the ☒ button; that will terminate the J-Link GDB Server!



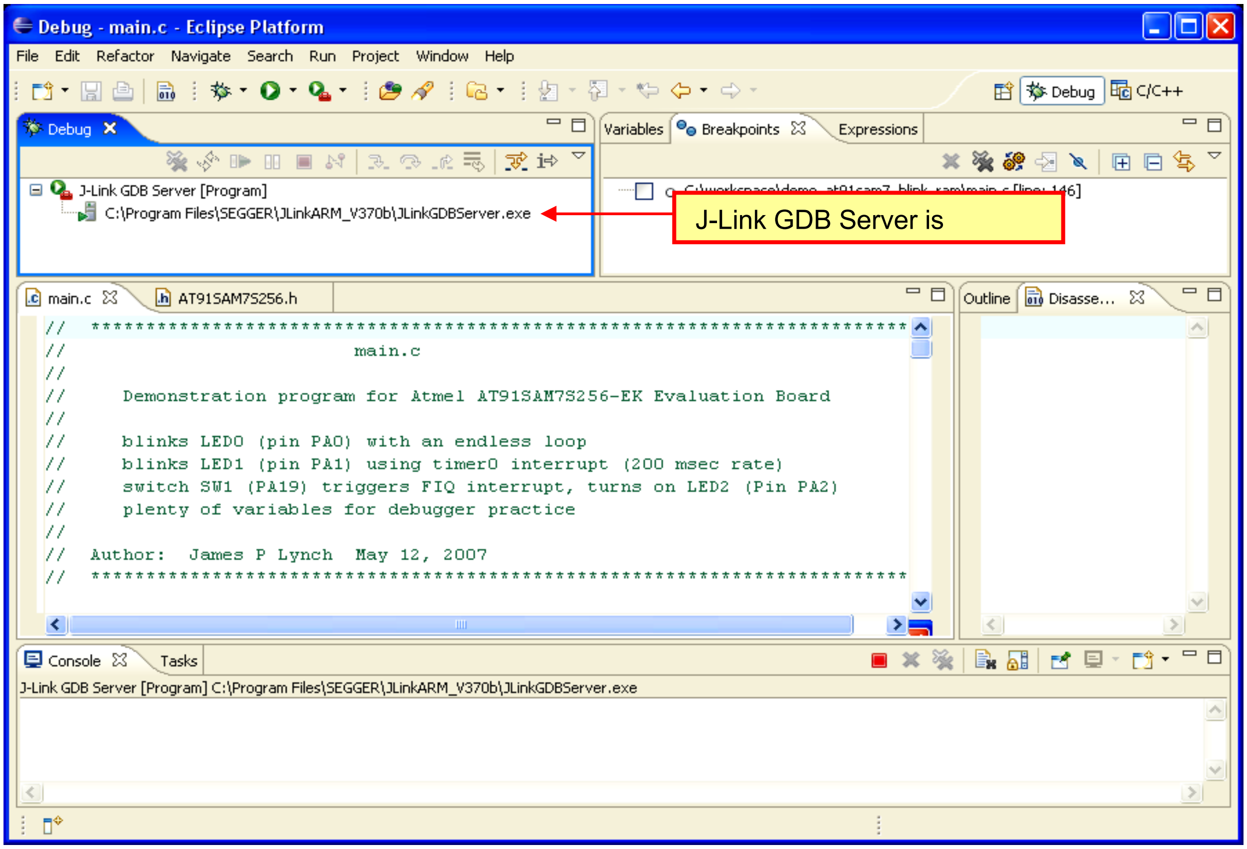
Debugger: Waiting for connection 🟥
J-Link: Connected 🟩
Target: ARM7, Core Id: 0x3F0F0F0F 🟩 3.31 V 🟩 Little endian
Initial JTAG speed: 30 kHz
Current JTAG speed: 30 kHz
☑ Stay on top
☑ Log window
☐ Log to file
☑ Cache reads
☐ Verify download

Log output:

```
J-Link GDB Server V3.70b

JLinkARM.dll V3.70b (DLL compiled May 18 2007 16:17:44)

Listening on TCP/IP port 2331

J-Link connected
Firmware: J-Link compiled May 10 2007 13:05:02 ARM Rev.5

J-Link found 1 JTAG device. Total IRLen = 4
JTAG ID: 0x3F0F0F0F (ARM7)
```

0 Bytes downloaded | 1 JTAG device

The debug view will show that J-Link GDB Server is running and the console view shows no errors (warnings are OK).



J-Link GDB Server [Program]
C:\Program Files\SEGGER\JLinkARM_V370b\JLinkGDBServer.exe

C:\workspace\demo_at91sam7_blink_ram\main.c [line: 146]

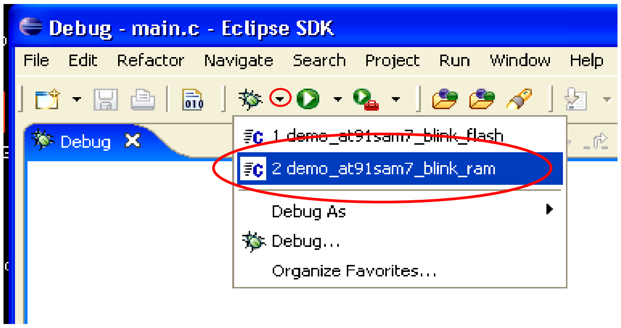
```
// ***************************************************************************
//                          main.c
//
//     Demonstration program for Atmel AT91SAM7S256-EK Evaluation Board
//
//     blinks LED0 (pin PA0) with an endless loop
//     blinks LED1 (pin PA1) using timer0 interrupt (200 msec rate)
//     switch SW1 (PA19) triggers FIQ interrupt, turns on LED2 (Pin PA2)
//     plenty of variables for debugger practice
//
//  Author:  James P Lynch  May 12, 2007
// ***************************************************************************
```

J-Link GDB Server [Program] C:\Program Files\SEGGER\JLinkARM_V370b\JLinkGDBServer.exe

Start the Eclipse Debugger

To start the Eclipse debugger, click on the “**Debug**” toolbar button’s down arrowhead and select the debug launch configuration “**demo_at91sam7_blink_ram**” as shown below.

Alternatively, you can start the debugger by clicking on “**Run – Debug…**” and then select the “**demo_at91sam7_blink_ram**” embedded launch configuration and then click “**debug**”. Obviously, the debug toolbar button is more convenient.

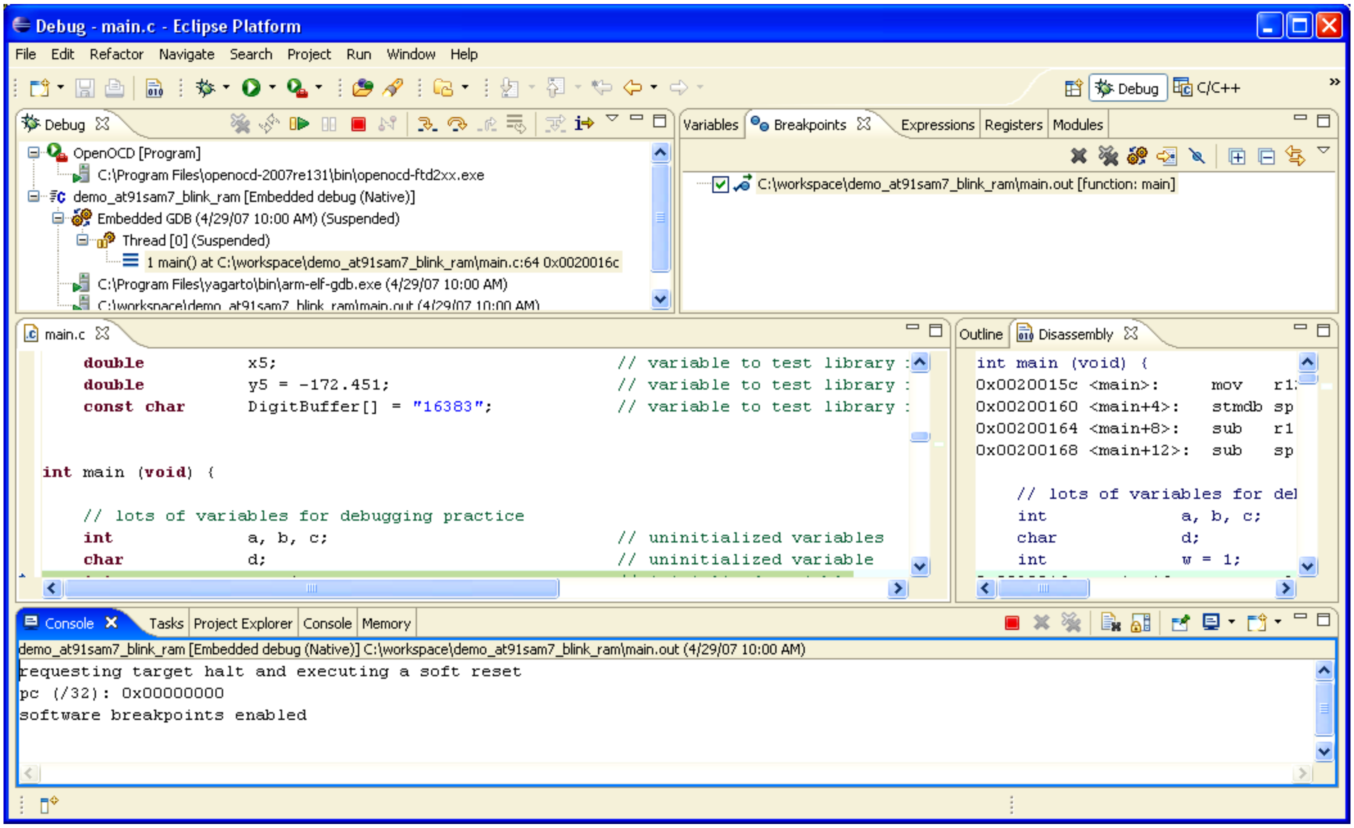

Debug - main.c - Eclipse SDK
File Edit Refactor Navigate Search Project Run Window Help
Debug
1 demo_at91sam7_blink_flash
2 demo_at91sam7_blink_ram
Debug As
Debug...
Organize Favorites...

The Eclipse debugger will run through the initializations you specified and then download the application into RAM. There will be a “progress bar” at the lower right corner of the Eclipse display showing the download in action. For these sample projects, this should only take a few seconds.

If the Eclipse debugger doesn't connect properly, then the progress bar at the bottom right status line will run forever. In this case, terminate everything, check your debug launch configuration very carefully and then start over again.

If the Eclipse debugger starts properly, the debug view (upper left) shows that the debugger has stopped at line 60 in main().

There is very little difference in starting up the Eclipse debugger between the OpenOCD and the J-Link GDB Server. The Eclipse debugger starting up using OpenOCD is shown below.



```
OpenOCD [Program]
    C:\Program Files\openocd-2007re131\bin\openocd-ftd2xx.exe
demo_at91sam7_blink_ram [Embedded debug (Native)]
    Embedded GDB (4/29/07 10:00 AM) (Suspended)
        Thread [0] (Suspended)
            1 main() at C:\workspace\demo_at91sam7_blink_ram\main.c:64 0x0020016c
    C:\Program Files\yagarto\bin\arm-elf-gdb.exe (4/29/07 10:00 AM)
    C:\workspace\demo_at91sam7_blink_ram\main.out (4/29/07 10:00 AM)
```

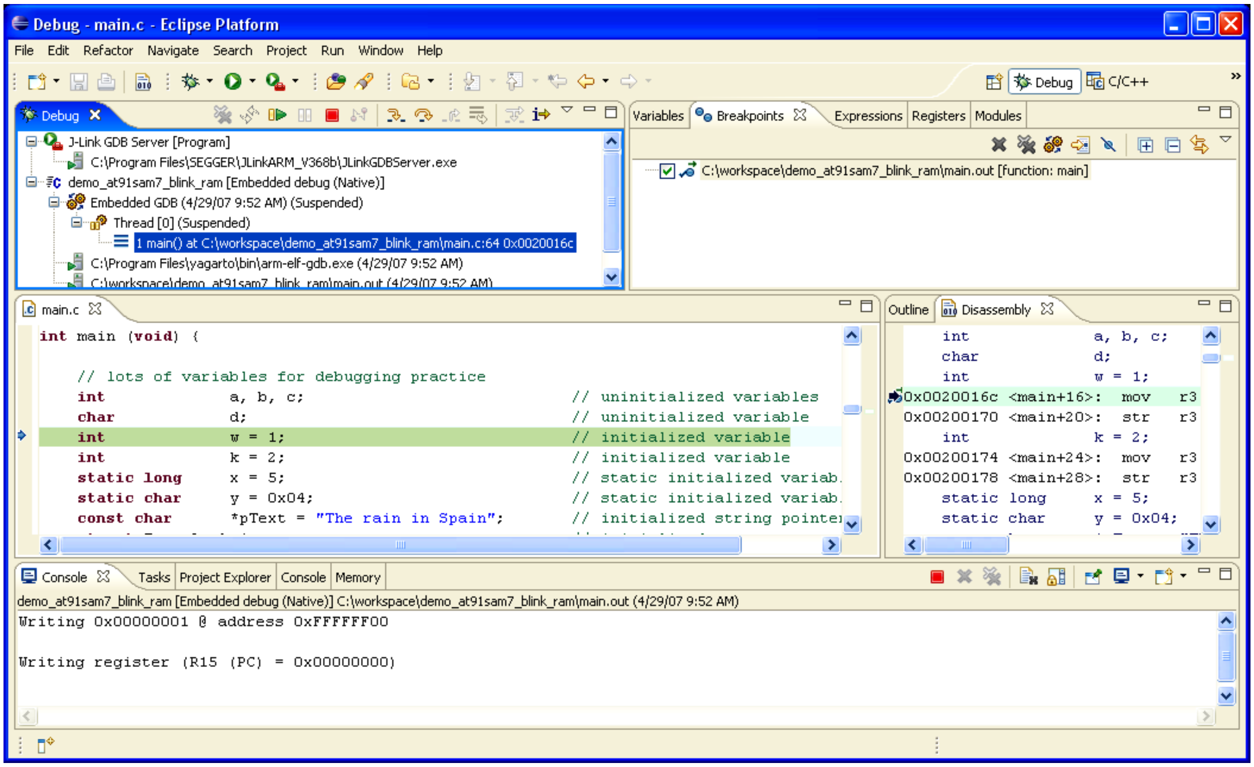
```
☑ C:\workspace\demo_at91sam7_blink_ram\main.out [function: main]
```

```
    double          x5;                          // variable to test library :
    double          y5 = -172.451;               // variable to test library :
    const char      DigitBuffer[] = "16383";     // variable to test library :


int main (void) {

    // lots of variables for debugging practice
    int             a, b, c;                     // uninitialized variables
    char            d;                           // uninitialized variable
```

```
int main (void) {
0x0020015c <main>:      mov    r1:
0x00200160 <main+4>:    stmdb  sp
0x00200164 <main+8>:    sub    r1
0x00200168 <main+12>:   sub    sp

    // lots of variables for del
    int             a, b, c;
    char            d;
    int             w = 1;
```

```
demo_at91sam7_blink_ram [Embedded debug (Native)] C:\workspace\demo_at91sam7_blink_ram\main.out (4/29/07 10:00 AM)
requesting target halt and executing a soft reset
pc (/32): 0x00000000
software breakpoints enabled
```

The Eclipse debugger starting up using J-Link GDB Server is shown below.



```
J-Link GDB Server [Program]
    C:\Program Files\SEGGER\JLinkARM_V368b\JLinkGDBServer.exe
demo_at91sam7_blink_ram [Embedded debug (Native)]
    Embedded GDB (4/29/07 9:52 AM) (Suspended)
        Thread [0] (Suspended)
            1 main() at C:\workspace\demo_at91sam7_blink_ram\main.c:64 0x0020016c
    C:\Program Files\yagarto\bin\arm-elf-gdb.exe (4/29/07 9:52 AM)
    C:\workspace\demo_at91sam7_blink_ram\main.out (4/29/07 9:52 AM)
```

☑ C:\workspace\demo_at91sam7_blink_ram\main.out [function: main]

```
int main (void) {

    // lots of variables for debugging practice
    int             a, b, c;                    // uninitialized variables
    char            d;                          // uninitialized variable
    int             w = 1;                      // initialized variable
    int             k = 2;                      // initialized variable
    static long     x = 5;                      // static initialized variab.
    static char     y = 0x04;                   // static initialized variab.
    const char      *pText = "The rain in Spain";  // initialized string pointe
```

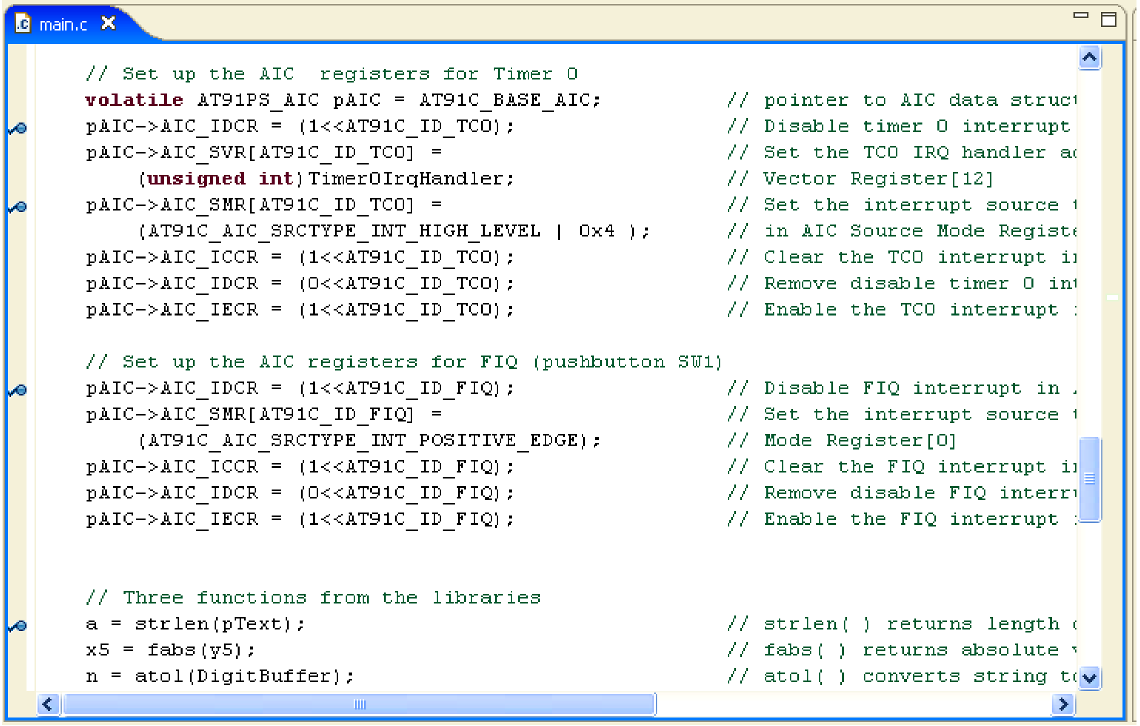
```
    int                 a, b, c;
    char                d;
    int                 w = 1;
0x0020016c <main+16>:  mov    r3
0x00200170 <main+20>:  str    r3
    int                 k = 2;
0x00200174 <main+24>:  mov    r3
0x00200178 <main+28>:  str    r3
    static long         x = 5;
    static char         y = 0x04;
```

demo_at91sam7_blink_ram [Embedded debug (Native)] C:\workspace\demo_at91sam7_blink_ram\main.out (4/29/07 9:52 AM)

```
Writing 0x00000001 @ address 0xFFFFFF00

Writing register (R15 (PC) = 0x00000000)
```

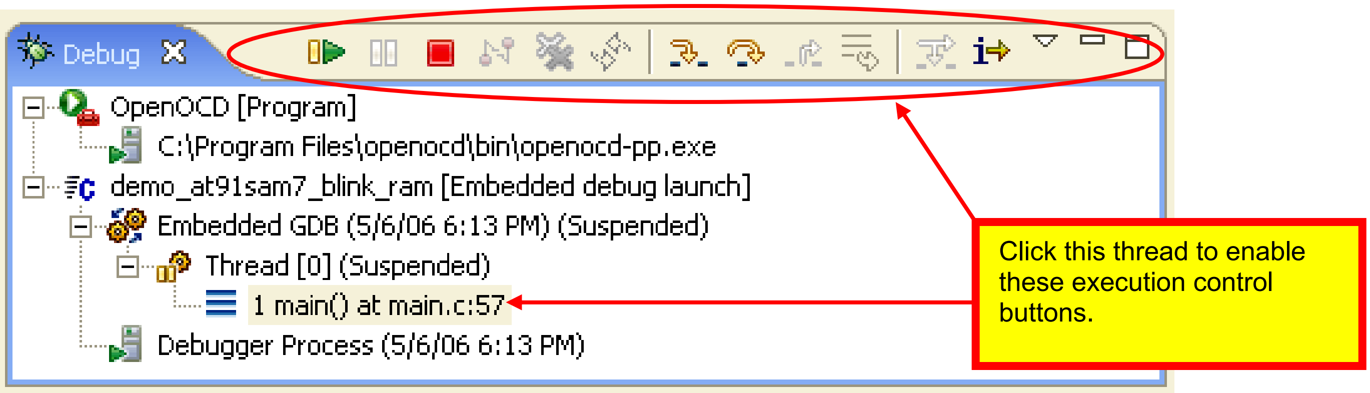
Setting Software Breakpoints

The big advantage of running entirely from on chip RAM is that you can set an unlimited number of software breakpoints. In the example below, we have set four breakpoints plus the breakpoint set at main( ).



```
    // Set up the AIC  registers for Timer 0
    volatile AT91PS_AIC pAIC = AT91C_BASE_AIC;          // pointer to AIC data struct
    pAIC->AIC_IDCR = (1<<AT91C_ID_TC0);                 // Disable timer 0 interrupt
    pAIC->AIC_SVR[AT91C_ID_TC0] =                       // Set the TC0 IRQ handler ad
        (unsigned int)Timer0IrqHandler;                 // Vector Register[12]
    pAIC->AIC_SMR[AT91C_ID_TC0] =                       // Set the interrupt source t
        (AT91C_AIC_SRCTYPE_INT_HIGH_LEVEL | 0x4 );      // in AIC Source Mode Registe
    pAIC->AIC_ICCR = (1<<AT91C_ID_TC0);                 // Clear the TC0 interrupt in
    pAIC->AIC_IDCR = (0<<AT91C_ID_TC0);                 // Remove disable timer 0 int
    pAIC->AIC_IECR = (1<<AT91C_ID_TC0);                 // Enable the TC0 interrupt i

    // Set up the AIC registers for FIQ (pushbutton SW1)
    pAIC->AIC_IDCR = (1<<AT91C_ID_FIQ);                 // Disable FIQ interrupt in A
    pAIC->AIC_SMR[AT91C_ID_FIQ] =                       // Set the interrupt source t
        (AT91C_AIC_SRCTYPE_INT_POSITIVE_EDGE);          // Mode Register[0]
    pAIC->AIC_ICCR = (1<<AT91C_ID_FIQ);                 // Clear the FIQ interrupt in
    pAIC->AIC_IDCR = (0<<AT91C_ID_FIQ);                 // Remove disable FIQ interru
    pAIC->AIC_IECR = (1<<AT91C_ID_FIQ);                 // Enable the FIQ interrupt i


    // Three functions from the libraries
    a = strlen(pText);                                  // strlen( ) returns length o
    x5 = fabs(y5);                                      // fabs( ) returns absolute v
    n = atol(DigitBuffer);                              // atol( ) converts string to
```

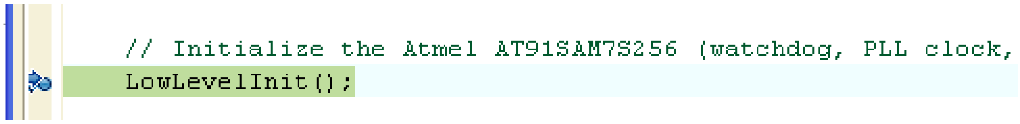
Let’s remind ourselves that the Eclipse debugger can handle multiple threads of execution. Since our ARM system only has one thread, you must click on it (highlight it) to enable the execution control commands to work. As shown below, the thread “1 main() at main.c:57” has been clicked and thus highlighted.



```
OpenOCD [Program]
    C:\Program Files\openocd\bin\openocd-pp.exe
demo_at91sam7_blink_ram [Embedded debug launch]
    Embedded GDB (5/6/06 6:13 PM) (Suspended)
        Thread [0] (Suspended)
            1 main() at main.c:57
    Debugger Process (5/6/06 6:13 PM)
```

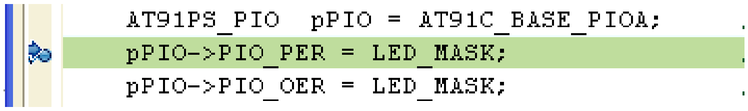
Click this thread to enable these execution control buttons.

Click on the “Resume” button ▶ and the debugger executes to our first breakpoint.



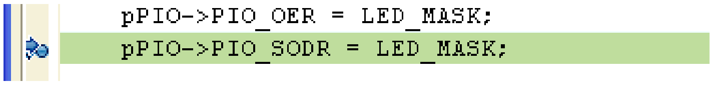
```
    // Initialize the Atmel AT91SAM7S256 (watchdog, PLL clock,
    LowLevelInit();
```

Click on the “Resume” ▶ button again and the debugger executes to our second breakpoint.



```
    AT91PS_PIO  pPIO = AT91C_BASE_PIOA;
    pPIO->PIO_PER = LED_MASK;
    pPIO->PIO_OER = LED_MASK;
```

Click on the “Resume” ▶ button again and the debugger executes to our third breakpoint.



```
    pPIO->PIO_OER = LED_MASK;
    pPIO->PIO_SODR = LED_MASK;
```

And so on, now we have an unlimited number of breakpoints available.

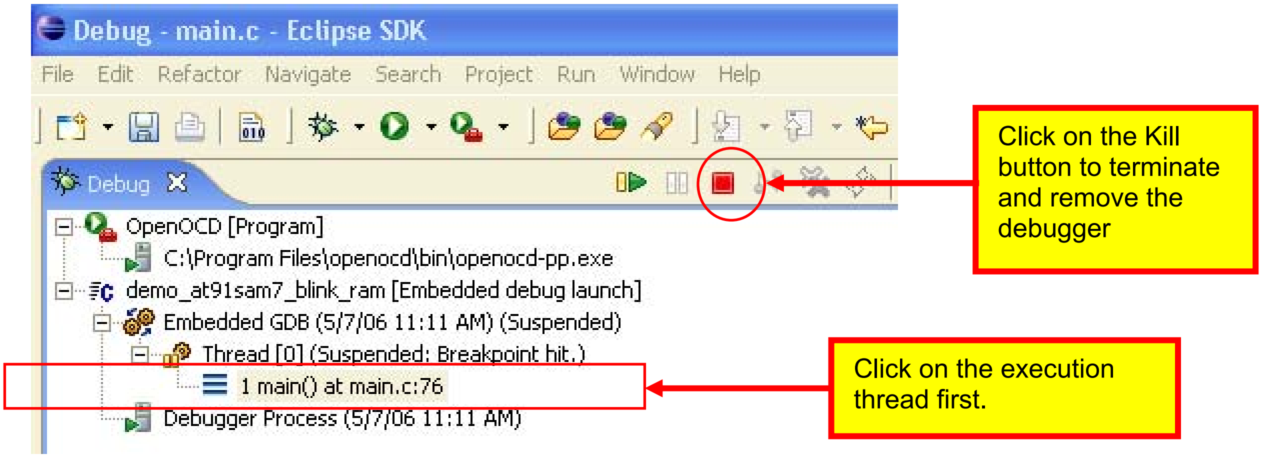
Now you can run through all the debugger operations covered earlier in this tutorial. Considering that modern desktop PCs and laptops are being manufactured without serial or parallel ports, a USB-based JTAG interface will soon be the only way to debug target boards.

Compiling from the Debug Perspective

You can conveniently stop the debugger and the OpenOCD or J-Link GDB Server, modify your source file and re-compile your application all within the Debug perspective. The following procedure is a safe way to do this.

- ### Stop the Eclipse Debugger

Click on the execution thread to highlight it and then click on the KILL button to terminate it.


Debug - main.c - Eclipse SDK
File Edit Refactor Navigate Search Project Run Window Help
Debug

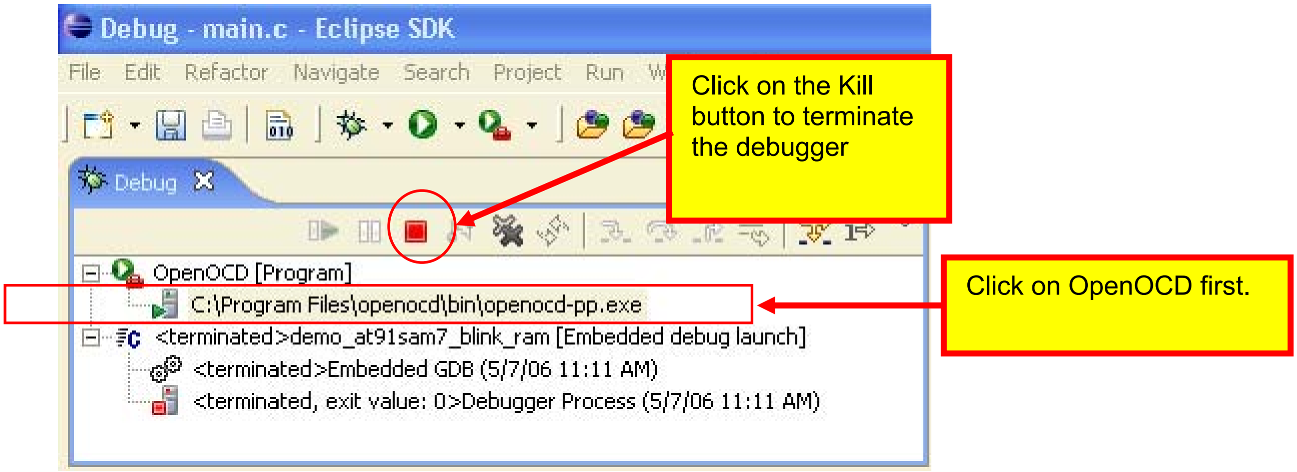
OpenOCD [Program]
C:\Program Files\openocd\bin\openocd-pp.exe
demo_at91sam7_blink_ram [Embedded debug launch]
Embedded GDB (5/7/06 11:11 AM) (Suspended)
Thread [0] (Suspended: Breakpoint hit.)
1 main() at main.c:76
Debugger Process (5/7/06 11:11 AM)

Click on the Kill button to terminate and remove the debugger

Click on the execution thread first.

- ### Stop OpenOCD or the J-Link GDB Server

Click on OpenOCD followed by clicking on the KILL button to terminate the OpenOCD or J-Link GDB Server. This operation may not be necessary, I often leave these daemons running and everything works OK.


Debug - main.c - Eclipse SDK
File Edit Refactor Navigate Search Project Run W
Debug

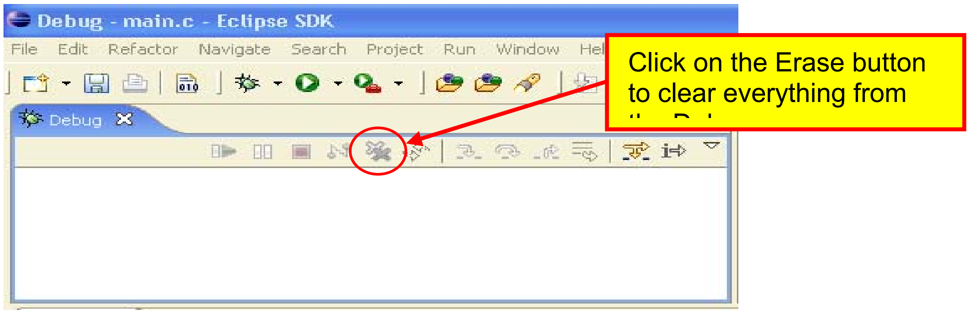
OpenOCD [Program]
C:\Program Files\openocd\bin\openocd-pp.exe
<terminated>demo_at91sam7_blink_ram [Embedded debug launch]
<terminated>Embedded GDB (5/7/06 11:11 AM)
<terminated, exit value: 0>Debugger Process (5/7/06 11:11 AM)

Click on the Kill button to terminate the debugger

Click on OpenOCD first.

- ## Erase the Debug Pane

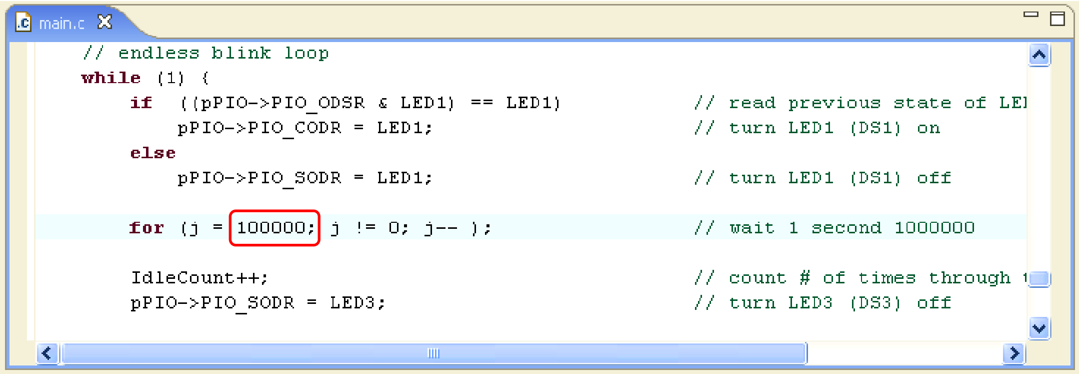
Click on the Erase button to clear everything from the Debug pane.


Debug - main.c - Eclipse SDK
File Edit Refactor Navigate Search Project Run Window Hel
Debug

Click on the Erase button to clear everything from

- ## Modify the Source File

Here we have changed the wait time by modifying the loop counts.

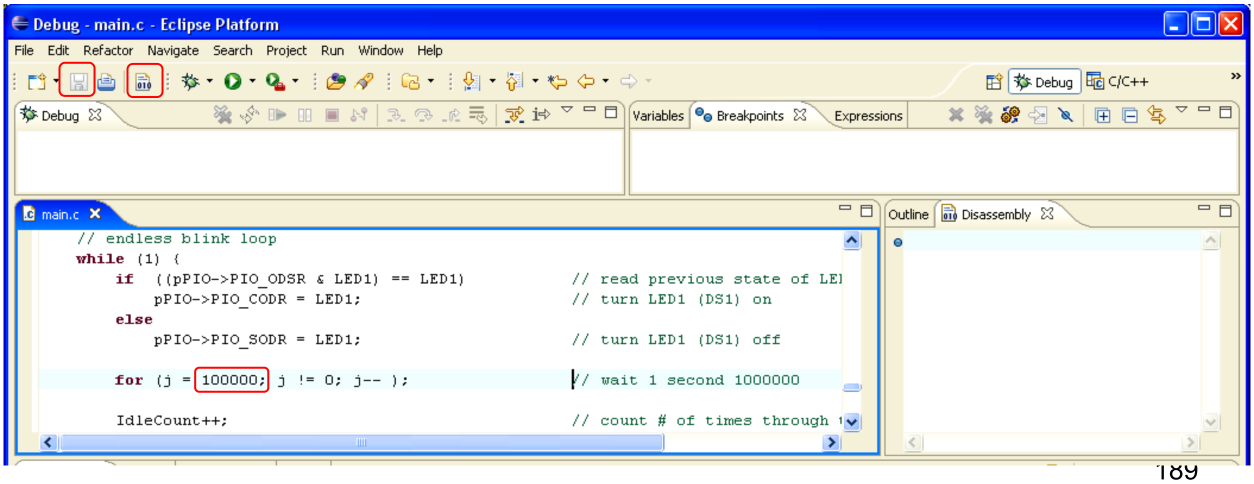


```
    // endless blink loop
    while (1) {
        if  ((pPIO->PIO_ODSR & LED1) == LED1)          // read previous state of LEI
            pPIO->PIO_CODR = LED1;                     // turn LED1 (DS1) on
        else
            pPIO->PIO_SODR = LED1;                     // turn LED1 (DS1) off

        for (j = 100000; j != 0; j-- );                // wait 1 second 1000000

        IdleCount++;                                   // count # of times through t
        pPIO->PIO_SODR = LED3;                         // turn LED3 (DS3) off
```

- ## Re-Compile and Link the Application

To change the blink rate, we modified the loop counts. We then saved the source file using the "**Save**" button.

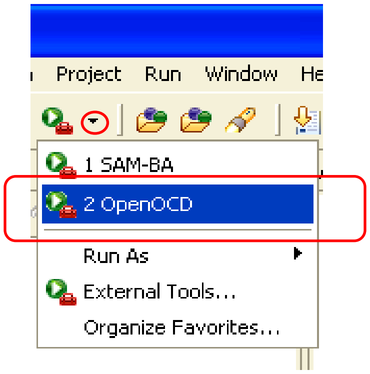
Next we re-built the application by clicking on the "**Build All**" button, as shown below. The Console view shows that the compile and link steps ran successfully. Note that it only compiled the source file main.c.


Debug - main.c - Eclipse Platform
File Edit Refactor Navigate Search Project Run Window Help
Debug
C/C++
Debug
Variables
Breakpoints
Expressions
main.c
Outline
Disassembly

```
    // endless blink loop
    while (1) {
        if  ((pPIO->PIO_ODSR & LED1) == LED1)          // read previous state of LEI
            pPIO->PIO_CODR = LED1;                     // turn LED1 (DS1) on
        else
            pPIO->PIO_SODR = LED1;                     // turn LED1 (DS1) off

        for (j = 100000; j != 0; j-- );                // wait 1 second 1000000

        IdleCount++;                                   // count # of times through t
```

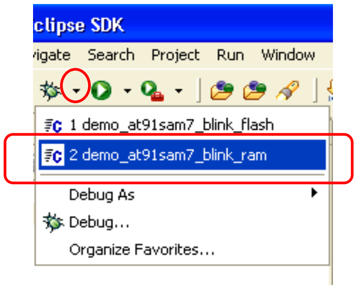
- ### Start OpenOCD or the J-Link GDB Server

Using the External Tools toolbar button, find and start the OpenOCD or J-Link GDB ServerJTAG utility.



- ### Start the Eclipse Debugger

Using the Debug toolbar button, find and start the at91sam7_blink_ram debug configuration.



- ### Repeat your Debugging Session

Now the Eclipse debugger is stopped at the function main( ), awaiting your next instructions. Once you have this procedure committed to memory, you will find RAM-based debugging a real pleasure.

Conclusions

Professional embedded software development packages from Rowley, IAR, Keil and ARM are complete, efficient, and easy-to-install and have telephone support if you encounter problems. For the professional programmer, they are worth the expense since “time is money”. Some of these companies offer “kick start” versions of their packages for free, albeit with some reduced functionality such as a 32K code limit, etc.

The Open Source tools described herein are an attractive alternative and are free. Thanks to the tireless contributions of open-source heroes such as Michael Fischer and Dominic Rath, the acquisition and installation of Open Source tools is becoming less complex and time-consuming. The reader needs a high speed internet connection to download the various components and a couple hours of time to install and test the lot.

Still, many thousands have managed successful application of Open Source tools for embedded software development. The GNU compilers are very close to the code efficiency of the professional compilers from Keil, IAR and ARM. The Eclipse and GNU Open Source tools bring the world of embedded software development to anyone on the planet that has imagination, skill and dedication but not the corporate bank account. Promoting the involvement of everyone in microprocessor development, not just an elite few, will allow us all to profit from their accomplishments.

About the Author

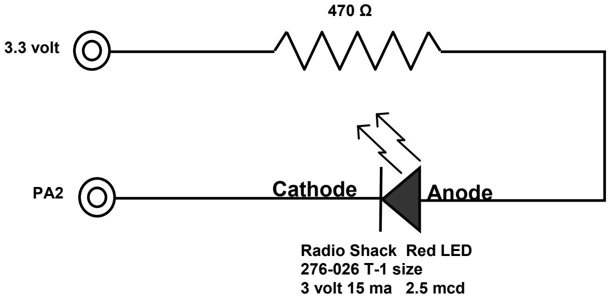
Jim Lynch lives in Grand Island, New York and is a software developer for Control Techniques, a subsidiary of Emerson Electric. He develops embedded software for the company’s industrial drives (high power motor controllers) which are sold all over the world.

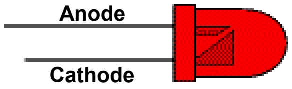
Mr. Lynch has previously worked for Mennen Medical, Calspan Corporation and the Boeing Company. He has a BSEE from Ohio University and a MSEE from State University of New York at Buffalo. Jim is a single Father and has two grown children who now live in Florida and Nevada. He has two brothers, one is a Viet Nam veteran in Hollywood, Florida and the other is the Bishop of St. Petersburg, also in Florida. Jim plays the guitar, enjoys woodworking and hopes to write a book very soon that will teach students and hobbyists how to use these high-powered ARM microcontrollers. Lynch can be reached via e-mail at: lynch007@gmail.com

Appendix 1. Olimex AT91SAM7- P64 Board

The Olimex AT91SAM7-P64 board has two LED's and two pushbutton switches while the Atmel AT91SAM7S-EK board has four LEDs and four pushbutton switches. The application described in this tutorial uses one switch and three LEDs. Fortunately, the pushbutton switches use the same PIO ports as the Atmel AT91SAM7S-EK board, so no change is required for the single switch. The Olimex board uses different PIO ports for the LEDs, so we are required to do two things; add a LED to the board and adjust the board.h file to specify the correct ports.

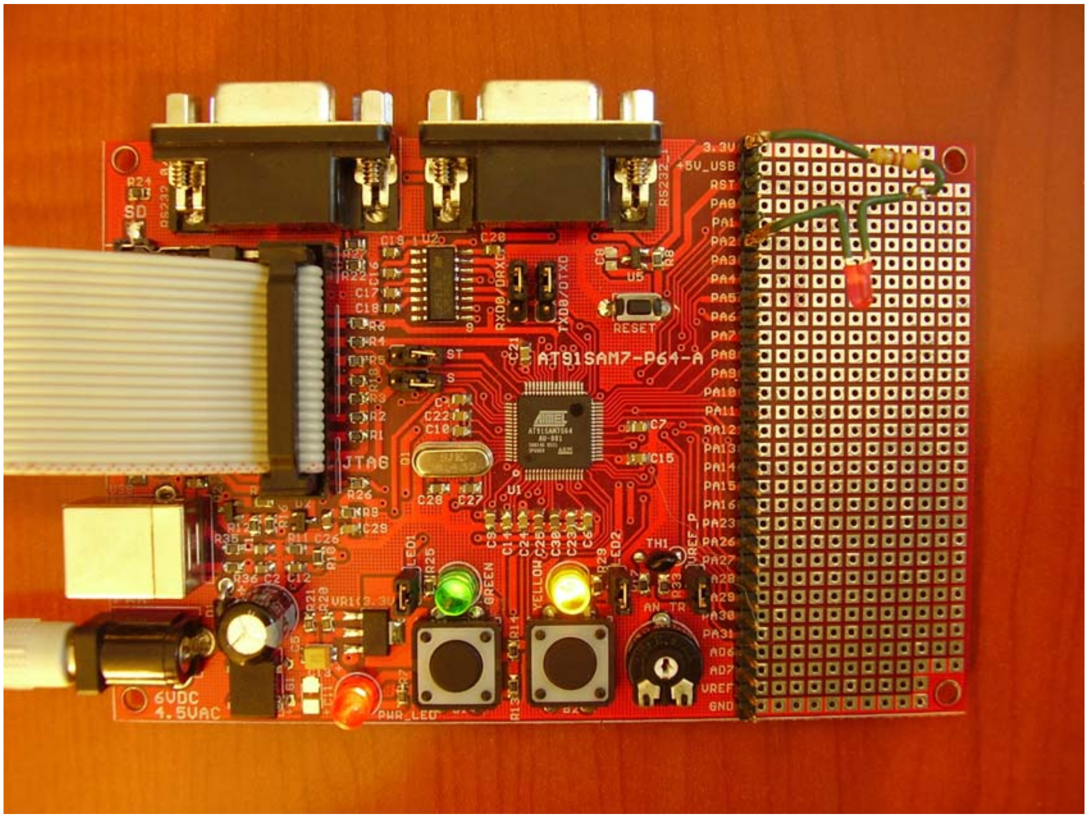
Since LED3 was port PA2 in the Atmel evaluation board, the author added the following simple circuit to the Olimex board.





Below is a photograph showing the added LED3. The board.h include file was modified to specify the correct ports for the LEDs and the switches. The major changes are indicated with bold-faced type.

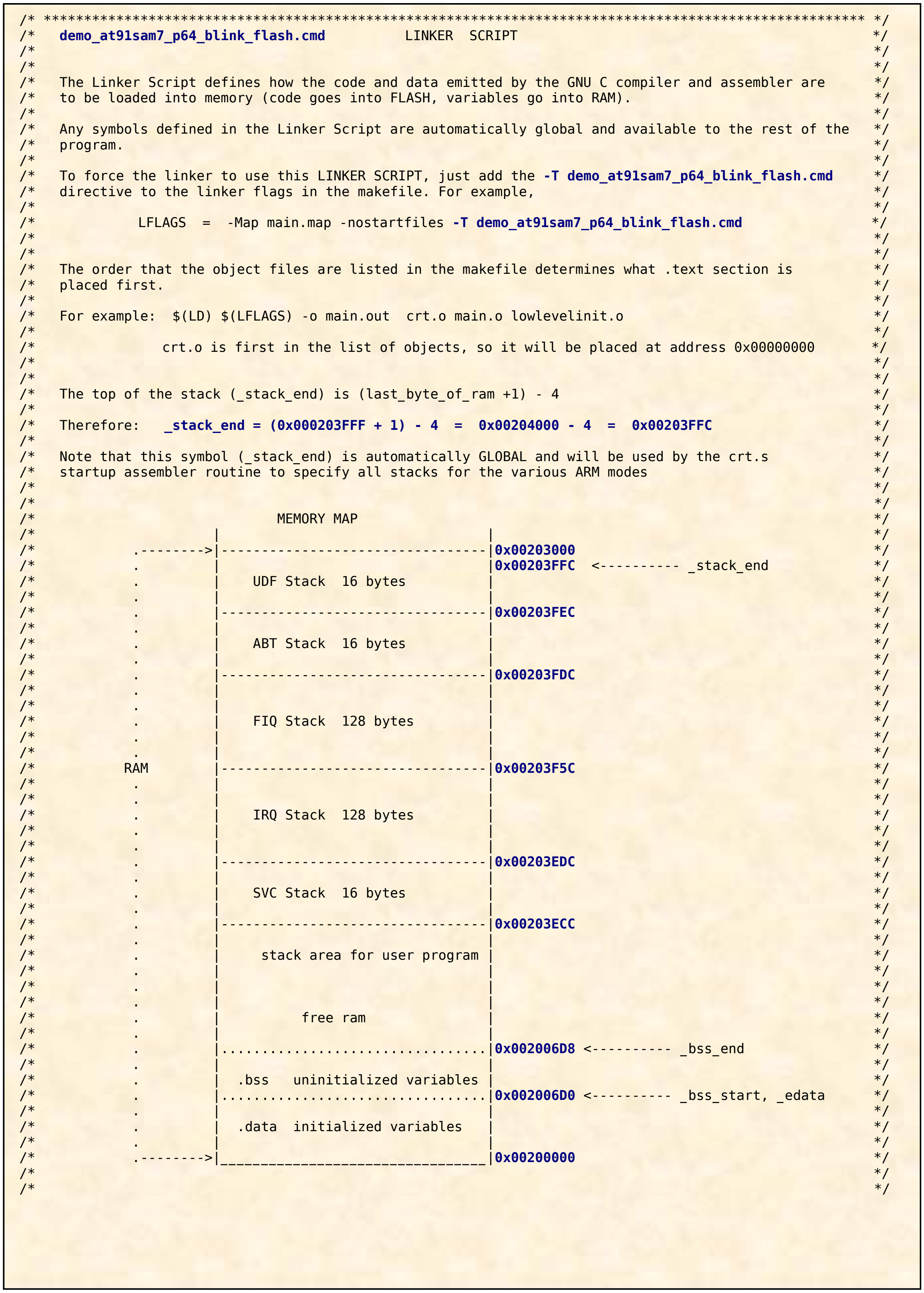
Warning: The author discovered that the Olimex ARM-USB-OCD JTAG interface's built-in power supply tends to latch up during OpenOCD FLASH programming. Possibly the Olimex board draws more current than the Atmel board; it does have a pot installed for A/D input. If you are planning to use OpenOCD FLASH programming with the Olimex board, it behooves you to use a separate "wall-wart" power supply instead.



BOARD.H

```
//----------------------------------------------------------------------------------------------------
//          ATMEL Microcontroller Software Support  -  ROUSSET  -
//----------------------------------------------------------------------------------------------------
// The software is delivered "AS IS" without warranty or condition of any
// kind, either express, implied or statutory. This includes without
// limitation any warranty or condition with respect to merchantability or
// fitness for any particular purpose, or against the infringements of
// intellectual property rights of others.
//----------------------------------------------------------------------------------------------------
// File Name:  Board.h
// Object:     AT91SAM7S Evaluation Board Features Definition File.
//
// Creation:   JPP   16/June/2004
//----------------------------------------------------------------------------------------------------
#ifndef Board_h
#define Board_h

#include "AT91SAM7S256.h"
#define __inline inline

#define true   -1
#define false  0

//-------------------------------------------------
// SAM7Board Memories Definition
//-------------------------------------------------
// The AT91SAM7S64 embeds a 16-Kbyte SRAM bank, and 64 K-Byte Flash

#define  INT_SRAM         0x00200000
#define  INT_SRAM_REMAP   0x00000000

#define  INT_FLASH        0x00000000
#define  INT_FLASH_REMAP  0x00100000

#define  FLASH_PAGE_NB    512
#define  FLASH_PAGE_SIZE  128

//-------------------------
// Leds Definition
//-------------------------
#define LED1          (1<<18)                  // PA18
#define LED2          (1<<17)                  // PA17
#define LED3          (1<<2)                   // PA2 (LED added to board by author)
#define NB_LEB        3
#define LED_MASK      (LED1|LED2|LED3)


//------------------------------------
// Push Buttons Definition
//-------------------------------------
#define SW1_MASK      (1<<19)                  // PA19
#define SW2_MASK      (1<<20)                  // PA20
#define SW_MASK       (SW1_MASK|SW2_MASK)

#define SW1           (1<<19)                  // PA19
#define SW2           (1<<20)                  // PA20

//--------------------------
// USART Definition
//--------------------------
// SUB-D 9 points J3 DBGU
#define DBGU_RXD      AT91C_PA9_DRXD           // JP11 must be close
#define DBGU_TXD      AT91C_PA10_DTXD          // JP12 must be close
#define AT91C_DBGU_BAUD   115200               // Baud rate
#define US_RXD_PIN    AT91C_PA5_RXD0           // JP9 must be close
#define US_TXD_PIN    AT91C_PA6_TXD0           // JP7 must be close
#define US_RTS_PIN    AT91C_PA7_RTS0           // JP8 must be close
#define US_CTS_PIN    AT91C_PA8_CTS0           // JP6 must be close

//---------------
// Master Clock
//---------------
#define EXT_OC        18432000                 // Exetrnal ocilator MAINCK
#define MCK           47923200                 // MCK (PLLRC div by 2)
#define MCKKHz        (MCK/1000)               //

#endif   // Board_h
```

The Olimex AT91SAM7-P64 board used the AT91SAM7S64 chip, which has 64K of FLASH and 16K of RAM. The linker command script, **demo_at91sam7_p64_blink_flash.cmd**, is modified to support these memory limits.



```
/* ******************************************************************************************************* */
/*   demo_at91sam7_p64_blink_flash.cmd          LINKER  SCRIPT                                             */
/*                                                                                                         */
/*                                                                                                         */
/*   The Linker Script defines how the code and data emitted by the GNU C compiler and assembler are       */
/*   to be loaded into memory (code goes into FLASH, variables go into RAM).                               */
/*                                                                                                         */
/*   Any symbols defined in the Linker Script are automatically global and available to the rest of the    */
/*   program.                                                                                              */
/*                                                                                                         */
/*   To force the linker to use this LINKER SCRIPT, just add the -T demo_at91sam7_p64_blink_flash.cmd      */
/*   directive to the linker flags in the makefile. For example,                                           */
/*                                                                                                         */
/*             LFLAGS  =  -Map main.map -nostartfiles -T demo_at91sam7_p64_blink_flash.cmd                 */
/*                                                                                                         */
/*                                                                                                         */
/*   The order that the object files are listed in the makefile determines what .text section is           */
/*   placed first.                                                                                         */
/*                                                                                                         */
/*   For example:  $(LD) $(LFLAGS) -o main.out  crt.o main.o lowlevelinit.o                                */
/*                                                                                                         */
/*                crt.o is first in the list of objects, so it will be placed at address 0x00000000        */
/*                                                                                                         */
/*                                                                                                         */
/*   The top of the stack (_stack_end) is (last_byte_of_ram +1) - 4                                        */
/*                                                                                                         */
/*   Therefore:   _stack_end = (0x000203FFF + 1) - 4  =  0x00204000 - 4  =  0x00203FFC                     */
/*                                                                                                         */
/*   Note that this symbol (_stack_end) is automatically GLOBAL and will be used by the crt.s              */
/*   startup assembler routine to specify all stacks for the various ARM modes                             */
/*                                                                                                         */
/*                                                                                                         */
/*                              MEMORY MAP                                                                 */
/*                      |                                 |                                                */
/*            .-------->|---------------------------------|0x00203000                                      */
/*            .         |                                 |0x00203FFC  <---------- _stack_end              */
/*            .         |    UDF Stack  16 bytes          |                                                */
/*            .         |                                 |                                                */
/*            .         |---------------------------------|0x00203FEC                                      */
/*            .         |                                 |                                                */
/*            .         |    ABT Stack  16 bytes          |                                                */
/*            .         |                                 |                                                */
/*            .         |---------------------------------|0x00203FDC                                      */
/*            .         |                                 |                                                */
/*            .         |                                 |                                                */
/*            .         |    FIQ Stack  128 bytes         |                                                */
/*            .         |                                 |                                                */
/*            .         |                                 |                                                */
/*           RAM        |---------------------------------|0x00203F5C                                      */
/*            .         |                                 |                                                */
/*            .         |                                 |                                                */
/*            .         |    IRQ Stack  128 bytes         |                                                */
/*            .         |                                 |                                                */
/*            .         |                                 |                                                */
/*            .         |---------------------------------|0x00203EDC                                      */
/*            .         |                                 |                                                */
/*            .         |    SVC Stack  16 bytes          |                                                */
/*            .         |                                 |                                                */
/*            .         |---------------------------------|0x00203ECC                                      */
/*            .         |                                 |                                                */
/*            .         |     stack area for user program |                                                */
/*            .         |                                 |                                                */
/*            .         |                                 |                                                */
/*            .         |                                 |                                                */
/*            .         |          free ram               |                                                */
/*            .         |                                 |                                                */
/*            .         |.................................|0x002006D8 <---------- _bss_end                 */
/*            .         |                                 |                                                */
/*            .         |  .bss   uninitialized variables |                                                */
/*            .         |.................................|0x002006D0 <---------- _bss_start, _edata       */
/*            .         |                                 |                                                */
/*            .         |  .data  initialized variables   |                                                */
/*            .         |                                 |                                                */
/*            .-------->|_________________________________|0x00200000                                      */
/*                                                                                                         */
/*                                                                                                         */
/*            .-------->|----------------------------------|0x00010000                                */
/*            .         |                                  |                                          */
/*            .         |                                  |                                          */
/*            .         |         free flash               |                                          */
/*            .         |                                  |                                          */
/*            .         |                                  |                                          */
/*            .         |..................................|0x000006D0 <---------- _bss_start, _edata      */
/*            .         |                                  |                                          */
/*            .         |  .data  initialized variables    |                                          */
/*            .         |                                  |                                          */
/*            .         |----------------------------------|0x000006C4 <----------- _etext            */
/*            .         |                                  |                                          */
/*            .         |            C code                |                                          */
/*            .         |                                  |                                          */
/*            .         |                                  |                                          */
/*            .         |----------------------------------|0x00000118  main()                        */
/*            .         |                                  |                                          */
/*            .         |    Startup Code  (crt.s)         |                                          */
/*            .         |         (assembler)              |                                          */
/*            .         |                                  |                                          */
/*            .         |----------------------------------|0x00000020                                */
/*            .         |                                  |                                          */
/*            .         | Interrupt Vector Table           |                                          */
/*            .         |          32 bytes                |                                          */
/*            .-------->|----------------------------------|0x00000000 _vec_reset                     */
/*                                                                                                    */
/*                                                                                                    */
/*  Author:  James P. Lynch      May 12, 2007                                                         */
/*                                                                                                    */
/* ******************************************************************************************************* */


/* identify the Entry Point  (_vec_reset is defined in file crt.s)  */
ENTRY(_vec_reset)

/* specify the AT91SAM7S64 memory areas  */
MEMORY
{
    flash  : ORIGIN = 0,          LENGTH = 64K      /* FLASH EPROM    */
    ram    : ORIGIN = 0x00200000, LENGTH = 16K      /* static RAM area*/
}


/* define a global symbol _stack_end  (see analysis in annotation above) */
_stack_end = 0x203FFC;


/* now define the output sections  */
SECTIONS
{
    . = 0;                           /* set location counter to address zero  */

    .text :                          /* collect all sections that should go into FLASH after startup  */
    {
        *(.text)                     /* all .text sections (code)  */
        *(.rodata)                   /* all .rodata sections (constants, strings, etc.)  */
        *(.rodata*)                  /* all .rodata* sections (constants, strings, etc.)  */
        *(.glue_7)                   /* all .glue_7 sections  (no idea what these are) */
        *(.glue_7t)                  /* all .glue_7t sections (no idea what these are) */
        _etext = .;                  /* define a global symbol _etext just after the last code byte */
    } >flash                         /* put all the above into FLASH */

    .data :                          /* collect all initialized .data sections that go into RAM  */
    {
        _data = .;                   /* create a global symbol marking the start of the .data section  */
        *(.data)                     /* all .data sections  */
        _edata = .;                  /* define a global symbol marking the end of the .data section  */
    } >ram AT >flash                     /* put all the above into RAM (but load the LMA initializer copy into
FLASH)  */

    .bss :                           /* collect all uninitialized .bss sections that go into RAM  */
    {
        _bss_start = .;                  /* define a global symbol marking the start of the .bss section */
        *(.bss)                      /* all .bss sections  */
    } >ram                           /* put all the above in RAM (it will be cleared in the startup code */

    . = ALIGN(4);                    /* advance location counter to the next 32-bit boundary */
    _bss_end = . ;                   /* define a global symbol marking the end of the .bss section */
}
    _end = .;                        /* define a global symbol marking the end of application RAM */
```

The makefile has three small changes; all concern the reference to the linker command script file. The changes are indicated in bold-face type.

MAKEFILE

```
# ***************************************************************
# *     Makefile for Atmel AT91SAM7S64 - flash execution        *
# *                                                             *
# *                                                             *
# *   James P Lynch  May 12, 2007                               *
# ***************************************************************

NAME = demo_at91sam7_p64_blink_flash

# variables
CC      = arm-elf-gcc
LD      = arm-elf-ld -v
AR      = arm-elf-ar
AS      = arm-elf-as
CP      = arm-elf-objcopy
OD     = arm-elf-objdump

CFLAGS  = -I./ -c -fno-common -O0 -g
AFLAGS  = -ahls -mapcs-32 -o crt.o
LFLAGS  =  -Map main.map -Tdemo_at91sam7_p64_blink_flash.cmd
CPFLAGS = --output-target=binary
ODFLAGS= -x --syms

OBJECTS = crt.o   main.o timerisr.o timersetup.o isrsupport.o lowlevelinit.o blinker.o


# make target called by Eclipse (Project -> Clean ...)
clean:
    -rm $(OBJECTS) crt.lst main.lst main.out main.bin main.hex main.map main.dmp


#make target called by Eclipse  (Project -> Build Project)
all:  main.out
    @ echo "...copying"
    $(CP) $(CPFLAGS) main.out main.bin
    $(OD) $(ODFLAGS) main.out > main.dmp

main.out: $(OBJECTS) demo_at91sam7_p64_blink_flash.cmd
    @ echo "..linking"
    $(LD) $(LFLAGS) -o main.out $(OBJECTS) libc.a libm.a libgcc.a

crt.o: crt.s
    @ echo ".assembling"
    $(AS) $(AFLAGS) crt.s > crt.lst

main.o: main.c
    @ echo ".compiling"
    $(CC) $(CFLAGS) main.c

timerisr.o: timerisr.c
    @ echo ".compiling"
    $(CC) $(CFLAGS) timerisr.c

lowlevelinit.o: lowlevelinit.c
    @ echo ".compiling"
    $(CC) $(CFLAGS) lowlevelinit.c

timersetup.o: timersetup.c
    @ echo ".compiling"
    $(CC) $(CFLAGS) timersetup.c

isrsupport.o: isrsupport.c
    @ echo ".compiling"
    $(CC) $(CFLAGS) isrsupport.c

blinker.o: blinker.c
    @ echo ".compiling"
    $(CC) $(CFLAGS) blinker.c
```

```
# **********************************************************************************************
#                             FLASH PROGRAMMING
#
# Alternate make target for flash programming only
#
# You must create a special Eclipse make target (program) to run this part of the makefile
# (Project -> Create Make Target...  then set the Target Name and Make Target to "program")
#
# OpenOCD is run in "batch" mode with a special configuration file and a script file containing
# the flash commands. When flash programming completes, OpenOCD terminates.
#
# Note that the script file of flash commands (script.ocd) is part of the project
#
# Programmers: Martin Thomas, Joseph M Dupre, James P Lynch
# **********************************************************************************************

# specify output filename here (must be *.bin file)
TARGET = main.bin

# specify the directory where openocd executable and configuration files reside (note: use forward slashes /)
OPENOCD_DIR = 'c:/Program Files/openocd-2007re141/bin/'

# specify OpenOCD executable (pp is for the wiggler, ftd2xx is for the USB debuggers)
#OPENOCD = $(OPENOCD_DIR)openocd-pp.exe
OPENOCD = $(OPENOCD_DIR)openocd-ftd2xx.exe

# specify OpenOCD configuration file (pick the one for your device)
#OPENOCD_CFG = $(OPENOCD_DIR)at91sam7s256-wiggler-flash-program.cfg
#OPENOCD_CFG = $(OPENOCD_DIR)at91sam7s256-jtagkey-flash-program.cfg
OPENOCD_CFG = $(OPENOCD_DIR)at91sam7s256-armusbocd-flash-program.cfg

# program the AT91SAM7S256 internal flash memory
program: $(TARGET)
    @echo "Flash Programming with OpenOCD..."          # display a message on the console
    $(OPENOCD) -f $(OPENOCD_CFG)                       # program the onchip FLASH here
    @echo "Flash Programming Finished."                # display a message on the console
```

The download package containing the sample programs also includes two sample projects for the Olimex board, one for FLASH execution and one for RAM execution. You will have to define a new Debug Launch Configuration for the Olimex projects; just employ the methods shown earlier in this tutorial.

If you have a 256K version of the Olimex board (AT91SAM7-P256), you will need to adjust the memory limits and top-of-stack specification in the linker command file shown above.

For the 64K board, these specifications are:

```
/* specify the AT91SAM7S64  */
MEMORY
{
   flash  :   ORIGIN = 0,              LENGTH = 64K     /* FLASH EPROM     */
   ram    :   ORIGIN = 0x00200000,     LENGTH = 16K     /* static RAM area */
}

/* define a global symbol _stack_end  (see analysis in annotation above) */
_stack_end = 0x203FFC;
```

For the 256K board, these specifications are:

```
/* specify the AT91SAM7S256  */
MEMORY
{
   flash  :   ORIGIN = 0,              LENGTH = 256K    /* FLASH EPROM     */
   ram    :   ORIGIN = 0x00200000,     LENGTH = 64K     /* static RAM area */
}


/* define a global symbol _stack_end  (see analysis in annotation above) */
_stack_end = 0x20FFFC;
```

Appendix 2. SOFTWARE COMPONENTS

One common problem in setting up a software development system composed of disparate modules downloaded from multiple sources on the web is ensuring that the various components will work harmoniously with each other.

To build this ARM cross development tool chain, we need the following components:

- **YAGARTO - Eclipse IDE version 3.2**
- **YAGARTO - Eclipse CDT 3.1 Plug-in for C++/C Development (Zylin custom version)**
- **YAGARTO - Native GNU C++/C Compiler suite for ARM Targets**
- **YAGARTO - OpenOCD version 141 or later for JTAGKey or ARM-USB-OCD JTAG debugging**
- **Segger J-Link GDB Server version 3.70b for SAM-ICE JTAG debugging**
- **Atmel SAM-BA version 2.5 flash programming utility**

Yagarto may be downloaded from here: http://www.yagarto.de/

The Segger J-Link GDB Server can be downloaded from the Segger web site: http://www.segger.de/

The Atmel SAM-BA flash programming utility can be downloaded from the Atmel web site:

http://www.atmel.com/dyn/products/product_card.asp?part_id=3524

A short zip file containing the tutorial and the sample Eclipse projects are hosted by Atmel at the following web address:

www.AT91.com

On the Atmel support web site above, go to the "Documents" section and search for this document "**Using Open Source Tools for AT91SAM7 Cross Development**". Follow the instructions given on page 56 of this document (Download the Tutorial Sample Projects) to retrieve the sample projects and configuration files.

A safe approach is to build the ARM software development system using the **YAGARTO** package above, get it to work and become familiar with it. Then you can monitor the Eclipse, Zylin, Yagarto and OpenOCD web sites for new versions and choose at a later time if you want to upgrade. So far, Michael has been very diligent in having the "latest and greatest" as part of YAGARTO.