# ARM cross development with GNU Toolchain and Eclipse

Version 1.1



Ronetix GmbH Waidhausenstrasse 13/5 1140 Vienna Austria Tel: +43-720-500315 +43-1962-720 500315 Fax: +43-1- 8174 955 3464 Internet: www.ronetix.at E-Mail info@ronetix.at

Acknowledgments:

ARM, ARM7, ARM9, and Thumb are trademarks of ARM Ltd. Windows, Win32, Windows CE are trademarks of Microsoft Corporation. Ethernet is a trademark of XEROX. All other trademarks are trademarks of their respective companies.

© 2005-2008 RONETIX All rights reserved.

2

### Change log

May 2007	- Updated download links
November 2006	- First release

1	INTRODUCTION	.5
2	PEEDI JTAG EMULATOR INSTALLATION	.6
3	TOOLSET INSTALLATION ON LINUX	.7
4	TOOLSET INSTALLATION ON WINDOWS	11
5	WORKING WITH ECLIPSE	12
5.1	Adding a project	12
5.2	Configuring and working with the Eclipse built-in debugger	17
5.3	Using Insight debugger as an Eclipse external tool	24

# **1** Introduction

This User Manual will show you how to install the GNU Toolchain and Eclipse, how to compile and debug a simple example using the Ronetix Evaluation board EB9261 with an Atmel AT91SAM9261 and PEEDI JTAG Emulator and Flash Programmer.

The necessary software components for an ARM cross development are:

- GNU toolchain (compiler, linker, gdb)
- Eclipse IDE including Zylin CDT Plug-in for C/C++ development
- Java Runtime

To enable a quick start in the ARM cross development Ronetix provides pre-built packages for Linux and Windows hosts.

The necessary files for a Linux host are:

- The GNU toolchain: http://download.ronetix.info/toolchains/arm/ronetix-gnutools-arm-elf-4.1.1-linux.tar.bz2
- The Eclipse IDE: <u>http://download.ronetix.info/eclipse/eclipse.tar.bz2</u>
- A simple example:http://download.ronetix.info/examples/at91sam9261 example mmu.tar.gz

The necessary file for a Windows host is:

http://download.ronetix.info/toolchains/arm/ronetix-toolset-arm.exe

This is a Windows installer which installs the GNU toolchain, Eclipse IDE + Zylin CDT and Java.

# **2 PEEDI JTAG Emulator Installation**

PEEDI (Powerful Embedded Ethernet Debug Interface) is an EmbeddedICE solution that enables you to debug software running on ARM processor cores via the JTAG port.

In order to debug you need to configure PEEDI JTAG Emulator. The configuration of PEEDI is common for both toolchains for Linux and Windows.

You can find detailed information about PEEDI in the PEEDI's User Manual: <u>http://download.ronetix.info/peedi/doc/peedi\_rev.A\_manual.pdf</u>

- Connect PEEDI to a free port of your LAN switch/hub using the supplied UTP patch cable.
- Connect PEEDI to the target using a JTAG cable and if needs the one of the supplied JTAG adapters. The JTAG adapter must be on the PEEDI side of the JTAG cable:



- Connect PEEDI to a COM port of your PC using the RS232 cable. Start any kind of terminal emulation program (HyperTerminal) and set it to 115200 bauds, 8 data bits, no parity and no flow control.
- Restart PEEDI holding pressed both front panel buttons to enter RedBoot command line.
- Use fconfig command to set the network configuration and other parameters.

#### WARNING:



If PEEDI is set to get its network settings from a DHCP server and if the Ethernet cable is unplugged or there is no DHCP server on the Ethernet, it may take some time for PEEDI to boot. To avoid this, make sure PEEDI can reach a DHCP server or assign a static IP address.

- Restart PEEDI again for the changes to take effect

After PEEDI is up and running (this should take some seconds after reset), press and hold the green front panel button and PEEDI will start to display its IP address on the display.

Connect to PEEDI with telnet application using the IP address from the previous statement. If connected, you should see the PEEDI CLI prompt

### **3 Toolset installation on Linux**

To install the pre-built from Ronetix GNU cross-development tools:

Download the GNU tools form here:

http://download.ronetix.info/toolchains/arm/ronetix-gnutools-arm-elf-4.1.1-linux.tar.bz2

or get it from the CD.

Uncompress the archive

Cd / tar xvfj ronetix-gnutools-arm-elf-4.1.1-linux.tar.bz2

The toolchain will be installed in the /usr/cross/arm directory. If want to install the toolset in another directory make sure you have a symbolic link in the /usr/cross

1. Set a path to the /usr/cross/arm/bin: in the .bashrc file add the following:

export PATH=\$PATH:/usr/cross/arm/bin

2. Test the toolchain installation:

```
[linbox]$ arm-elf-gcc -v
Using built-in specs.
Target: arm-elf
Configured with: /home/src/cross/gcc-4.1.1/configure --target=arm-elf -
-build i686-pc-linux-gnu --host i686-pc-linux-gnu --disable-nls --with-
float=soft --prefix=/usr/cross/arm-elf --enable-interwork --enable-
multilib --enable-languages=c,c++ --with-newlib --enable-win32-
registry=ronetix-arm --with-gnu-as --with-gnu-ld --with-
headers=/home/src/cross/newlib-1.14.0/newlib/libc/include
Thread model: single
gcc version 4.1.1
```

3. Installing Eclipse IDE

Download the Eclipse IDE from here or get it form the CD:

http://download.ronetix.info/eclipse/eclipse.tar.bz2

The file eclipse.tar.bz2 includes:

- Eclipse SDK v3.2.1
- Embedded CDT v3.1 patched by Zylin
- Zylin plugin
- Java Virtual Machine v1.5.0\_09

```
cd /usr/local
tar xvfj eclipse.tar.bz2
```

Set a path to the /usr/local/eclipse: in the .bashrc file add the following:

export PATH=\$PATH:/usr/local/eclipse

4. Installing an example

Download the EB9261 example from here:

http://download.ronetix.info/examples/at91sam9261\_example\_mmu.tar.gz

or get it from the CD.

Cd mkdir work cd work tar xvfz at91sam9261\_example\_MMU.tar.gz

At this point you should be able to build, debug and run applications on embedded ARM targets. You can compile and debug the example manual, from the shell prompt or using Eclipse. The working with Eclipse is explained in "Section 5: Working with Eclipse" from this manual.

5. Compiling from the shell

```
Cd at91sam9261
```

```
[linbox]$ make
ārm-elf-gcc -x assembler-with-cpp -c -mcpu=arm9 -g -gdwarf-2 -wa,-
amh1s=src/cstartup.1st src/cstartup.5 -o src/cstartup.o
arm-elf-gcc -c -mcpu=arm9 -00 -gdwarf-2 -mthumb-interwork -fomit-frame-
pointer -Wall -Wstrict-prototypes -fverbose-asm -Wa, -ahlms=src/init.lst
-MD -MP -MF .dep/init.o.d -I . -I./inc src/init.c -o src/init.o
arm-elf-gcc -c -mcpu=arm9 -00 -gdwarf-2 -mthumb-interwork -fomit-frame-
pointer -Wall -Wstrict-prototypes -fverbose-asm -Wa,-ahlms=src/main.lst
-MD -MP -MF .dep/main.o.d -I . -I./inc src/main.c -o src/main.o
arm-elf-gcc _c -mcpu=arm9 -00 -gdwarf-2 -mthumb-interwork -fomit-frame-
pointer -wall -wstrict-prototypes -fverbose-asm -wa,
ah1ms=src/dhry_1.1st
                                -MD -MP -MF .dep/dhry_1.o.d -I . -I./inc
src/dhry_1.c -o src/dhry_1.o
arm-elf-gcc -c -mcpu=arm9 -00 -gdwarf-2 -mthumb-interwork -fomit-frame-
pointer -wall -wstrict-prototypes -fverbose-asm -wa,
ah1ms=src/dhry_2.1st
                                -MD -MP -MF .dep/dhry_2.o.d -I . -I./inc
src/dhry_2.c -o src/dhry_2.o
arm-elf-gcc -c -mcpu=arm9 -00 -gdwarf-2 -mthumb-interwork -fomit-frame-
pointer -wall -wstrict-prototypes -fverbose-asm -wa,-ahlms=src/mmu.lst
-MD -MP -MF .dep/mmu.o.d -I . -I./inc src/mmu.c -o src/mmu.o
arm-elf-gcc ./src/cstartup.o ./src/init.o ./src/main.o ./src/dhry_1.o
./src/dhry_2.o ./src/mmu.o -mcpu=arm9 -nostartfiles -
T./prj/AT91SAM9261.1d -W1,-Map=BasicMMU.map,--cref,--no-warn-mismatch
-o BasicMMU.elf
arm-elf-objcopy -O ihex BasicMMU.elf BasicMMU.hex
```

6. Debugging

To simplify the debugging there is a file '.gdbinit' in the project directory which is read when you run gdb/insight. Adjust the '~/work/at91sam9261/.gdbinit' to match the IP address of PEEDI. From a X-console start the Insight debugger:

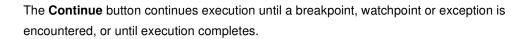
[linbox] arm-elf-insight BasicMMU.elf

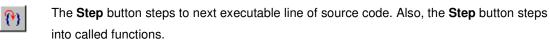
× main	n.c - Source W	Vindow								• 🗆 🗙
<u>F</u> ile	<u>R</u> un <u>V</u> iew	<u>C</u> ontrol	<u>P</u> referer	nces	Help					
35	P P (P	<b>*</b> 0 (¶)	1	A. (	<b>=</b> 69	<b>#</b> 4		Find:		
main	. C	▼	main			•			SOURCE	•
$\begin{array}{c} 129\\ - 129\\ - 130\\ - 133\\ - 133\\ - 134\\ - 133\\ - 133\\ - 133\\ - 133\\ - 133\\ - 133\\ - 133\end{array}$	0 void ( 1 AT91F) 2 3 AT91F) 4 AT91F) 5 AT91F) 6 AT91F) 7	oid) *pDhrystor DBGU_Print DisableICa DisableDCa Dhrystone( DBGU_Print	k("-I- En k("-I- n che(); che(); );	nter i o MMU,	n main I+D C	() secti	on\n\r sabled	"); (\n\r");		
Progr	ram stopped	at line	130						20000d30	130

The following descriptions discuss the use of the default debugger toolbar buttons.

The **Run** button, do not use this button. Instead of this use the "**Continue**" button

During the debugging process, the **Run** button turns into the **Stop** button to interrupt the debugging.





- The **Next** button steps to the next executable line of source code in the current file. Unlike the **Step** button, the **Next** button steps over called functions.
- The **Finish** button finishes execution of a current frame. If clicked while in a function, it finishes the function and returns to the line that called the function.

#### **Download Performance**

\*0

0

₽

By default, GDB versions previous than v6.5.50 (from 06.08.2006) download program code and data in small packets that are not necessarily a multiple of four bytes in length. This causes program download times to be slower than necessary, especially with ARM targets. There are two GDB internal variables that affect this. To improve GDB download performance, you should set the download write size to a binary value 16KB and the memory write packet size to a larger value to allow for packet overhead (+100 bytes is plenty). For example, to download 16KB at a time:

```
(gdb) set remote memory-write-packet-size fixed
(gdb) set remote memory-write-packet-size 16384
(gdb) set download-write-size 16128
```

#### Note:

No error is returned to GDB in case of an invalid 4 byte memory read access during



a period defined by **GDB\_READ\_INGNORE\_TIME** in the target configuration file. This is because GDB refuses to connect to the target if it gets an error during this connection sequence. GDB tries to read the stack frame during the connection sequence and this may lead to invalid memory access.

## **4 Toolset installation on Windows**

Please download the GNU toolchain installer from our web site, launch it and follow the instructions. <u>http://download.ronetix.info/toolchains/arm/ronetix-toolset-arm.exe</u>

The setup will install:

By default, the installer provided by Ronetix installs everything necessary for developing. This includes:

- arm-elf GNU tools (compiler, linker, and debugger)
- Eclipse IDE v3.2.1 and Zylin CDT plug-in
- Java run-time environment for the Eclipse
- ready-to-build examples
- a free TFTP server

# **5 Working with Eclipse**

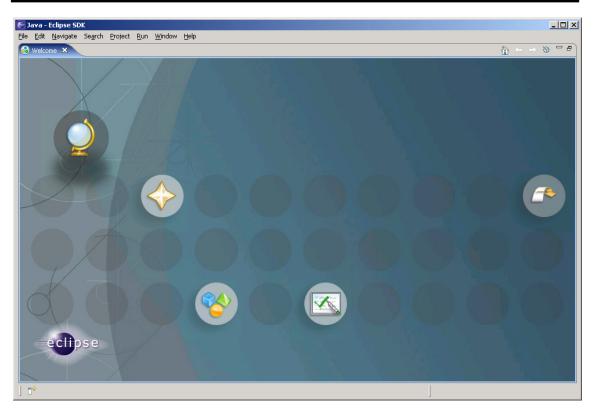
### 5.1 Adding a project

Eclipse (<u>http://www.eclipse.org</u>) is an open-source, Java based, powerful Integrated Development Environment (IDE). Adding the CDT plug-in (C/C++ Development Toolkit), you can edit and build C programs using the GNU compiler toolkit. A detailed C/C++ Development User Guide for Eclipse can be downloaded from <u>http://www.eclipse.org/cdt</u> or from the Ronetix site.

The Java run-time environment, Eclipse and the CDT plug-in are installed by the installer from Ronetix. Now start Eclipse IDE using the desktop icon, the "Workspace Launcher" dialog should appear, where you need to point the workspace folder. In our case point "C:\Program Files\Ronetix\examples\at91sam9261\pm9261", this way eclipse will automatically include the project files when we create a new project. Next click OK.

🗲 Workspace Launcher	×
Select a workspace	
Eclipse SDK stores your projects in a folder called a workspace. Choose a workspace folder to use for this session.	
Workspace: C:\Program Files\Ronetix\examples\at91sam9261\pm9261  Browse	
Use this as the default and do not ask again	
OK Cancel	

Now click at the "Workbench" icon at the right edge of the window.



The next window will look like:

🗲 Java - Eclipse SDK					- O ×
Eile Edit Source Refactor Navigate Search	<u>Project Run Window H</u> elp				
] 📬 • 🔡 📥 ] 🏇 • 💽 • 🤷 • ] 🍰	瞪 🞯 • ] 🥭 🖋 ] 🔁 • ] 🧏 •	₩ + +			
🚦 Package Explorer 🗙 🛛 Hierarchy 🖓 🗖				Outline 🛛	
← → ᡚ 🔲 😫 🏹			An	outline is not available.	
				->1	
	Problems X Javadoc Declaration				~
	0 errors, 0 warnings, 0 infos Description	Resource	Path	Loca	tion
			8		
			0		
			e la companya de la compa		
] □◆				]   🐴   奠 🗵	3 🥸 🔶 🛛

Now you need to add as projects the pm9261 example. Click File->New->Project. In the New Project dialog select Standard Make C Project and click Next:

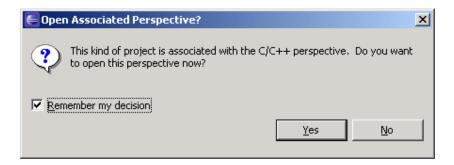
13

E New Project	×
Select a wizard Create a new C Project which uses a simple makefile.	
<u>W</u> izards:	
type filter text	
Java Project Java Project from Existing Ant Buildfile Plug-in Project General C Managed Make C Project Standard Make C Project C C++ C++ CVS	
⑦ < Back Next > Einish	Cancel

In the "New Project" dialog put "BasicMMU" for project name and click Finish. The BasicMMU already exists there so all of its files will be automatically imported in the new created Eclipse project.

🗲 New Project	x
C/Make Project Create a New C Project using 'make' to build it	G
Project name: BasicMMU Use default location Location: C:/Program Files/Ronetix/examples/at91sam9261/pm9261/BasicMMU/Ba	Browse,
⑦ < <u>Back</u> <u>Next</u> > <u>Finish</u>	Cancel

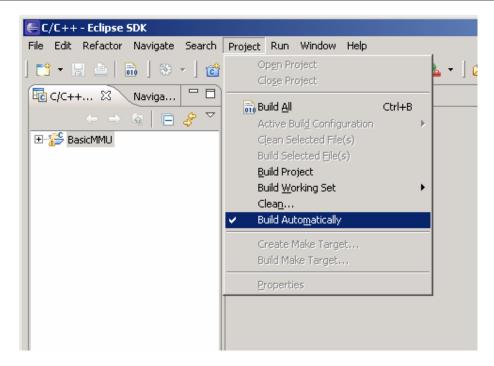
Now Eclipse will ask you to change the perspective to C/C++, check remember my decision and click YES:



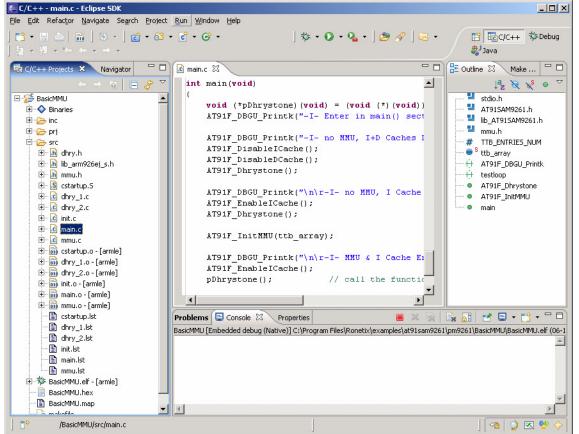
Now the Eclipse window should look like:

E/C++ - Eclipse SDK			
<u>File Edit Refactor Navigate Search Project Run Window H</u> elp			
] 📸 • 🗒 ≜   🖬   ⊗ + ] @ • @ • @ • @ • ] 巻 • Ø • 🧏 • ] 🥭 🖋   🗟 •	」と、行		
1 1 C/C++ X Naviga □ □		- 0	╊ O 🕅 M 🗖 🗖
$\leftarrow \rightarrow \alpha \models \mathscr{D} \checkmark$			An outline is not available.
BasicMMU			
R Problems X Console Properties			⇒ ~ □
0 errors, 1 warning, 0 infos	-	1	
Description A	Resource	Path	Location
B Warnings (1 icem) B Error launching external scanner info generator (gcc -E -P - v -	di BasicMMU		Unknown
∫ <b>□</b> ◆			🗠 💖 🙍   🔊

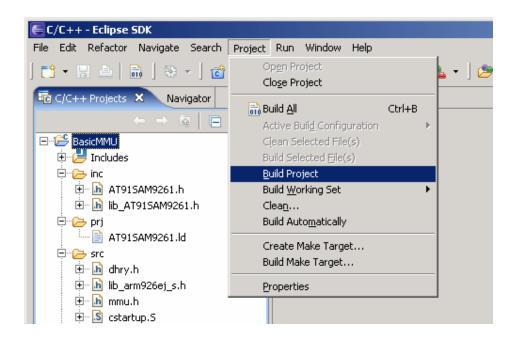
Use the "Project" menu and remove the checkmark at "Build Automatically":



Open main.c by double clicking on it in the BasicMMU/src folder:

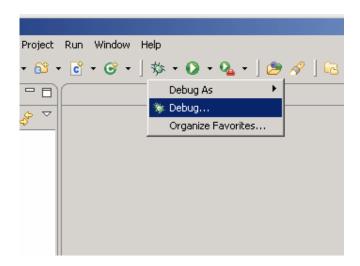


Now build the project:



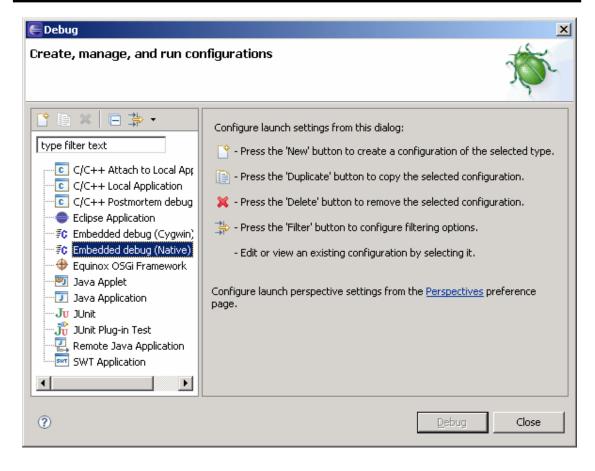
At this point your project should be compiled and liked to a single executable "BasicMMU.elf" file. You can check its presence.

### 5.2 Configuring and working with the Eclipse built-in debugger



First click the arrow right to the "bug" button and next "Debug..."

This opens the Debug configuration dialog. Select Embedded debug (Native) and click the New button:



This will change the look of the window to:

🗲 Debug		×
Create, manage, and run confi	gurations	Ť.
Image: Second system         Image: Second system <th>Name:       BasicMMU         Main       Image: Debugger       Image: Commands       Image: Source         Project:       Image: Source       Image: Source       Image: Source         BasicMMU       Image: Source       Image: Source       Image: Source       Image: Source         C/C++ Application:       Image: Search Project       Image: Source       Image: Source       Image: Source         Apply_       Image: Source       Image: Source</th> <th>Erowse Browse Revert</th>	Name:       BasicMMU         Main       Image: Debugger       Image: Commands       Image: Source         Project:       Image: Source       Image: Source       Image: Source         BasicMMU       Image: Source       Image: Source       Image: Source       Image: Source         C/C++ Application:       Image: Search Project       Image: Source       Image: Source       Image: Source         Apply_       Image: Source       Image: Source	Erowse Browse Revert
0	Debug	Close

If your window does not look like the previous, follow the next two steps:

1. Enter BasicMMU for name and click Project Browse to select the BasicMMU and click OK:

in the search for a pro	gram
ОК	Cancel
	in the search for a pro

2 .Next enter BasicMMU.elf for C/C++ Application.

🗲 Debug	×
Create, manage, and run cor	nfigurations
Image: Second system         Image: Second system <td>Name:       BasicMMU         Main       Source       Common         Project:       BasicMMU       Browse         C/C++ Application:       Browse       Browse         BasicMMU.elf       Search Project       Browse         Apply       Reyert</td>	Name:       BasicMMU         Main       Source       Common         Project:       BasicMMU       Browse         C/C++ Application:       Browse       Browse         BasicMMU.elf       Search Project       Browse         Apply       Reyert
0	Debug Close

Now select the Debugger tab and Browse to locate "...\Ronetix\arm-elf\bin\arm-elf-gdb.exe":

🗲 Debug	×
Create, manage, and run	configurations
Image: Second system         Image: Second system <th>Name:       BasicMMU         Main       Commands       Source       Common         Debugger:       Embedded GDB       Stop on startup at:       Advanced         Debugger Options       Imain       Stop on startup at:       Advanced         GDB debugger:       Imain       Imain       Browse         GDB command file:       Browse       Browse         (Warning: Some commands in this file may interfere with the startup operation of the debugger, for example "run".)       GDB command set:       Standard         Protocol:       Imi       Imi       Imi       Imi</th>	Name:       BasicMMU         Main       Commands       Source       Common         Debugger:       Embedded GDB       Stop on startup at:       Advanced         Debugger Options       Imain       Stop on startup at:       Advanced         GDB debugger:       Imain       Imain       Browse         GDB command file:       Browse       Browse         (Warning: Some commands in this file may interfere with the startup operation of the debugger, for example "run".)       GDB command set:       Standard         Protocol:       Imi       Imi       Imi       Imi
۲	Appl <u>y</u> Re <u>v</u> ert
0	Debug Close

Now change to the Commands tab and enter these commands considering your PEEDI IP address:

듣 Debug	×
Create, manage, and run confi	gurations 🔅
<ul> <li>▶</li> <li>▶</li> <li>▶</li> <li>▶</li> <li>▶</li> <li>▶</li> <li>▶</li> <li>♥</li> <li>♥</li></ul>	Name:       BasicMMU         Main       Debugger       Commands         'Initialize' commands       Source       Common         'Initialize' commands       Image: Source       Common         'Run' commands       Image: Source       Image: Source       Image: Source         'Run' commands       Image: Source       Image: Source       Image: Source
	Appl <u>y</u> Re <u>v</u> ert
0	Debug Close

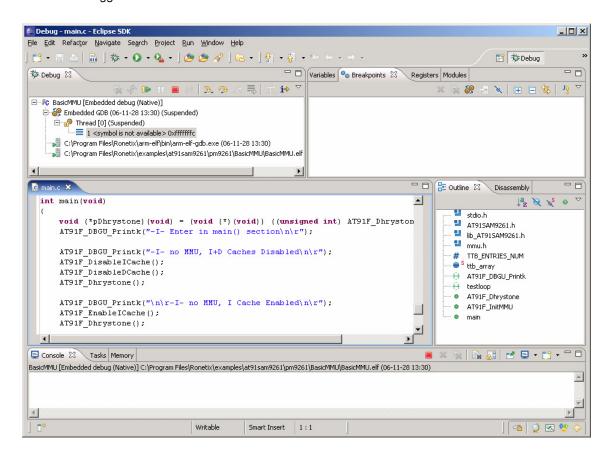
These command will tell gdb:

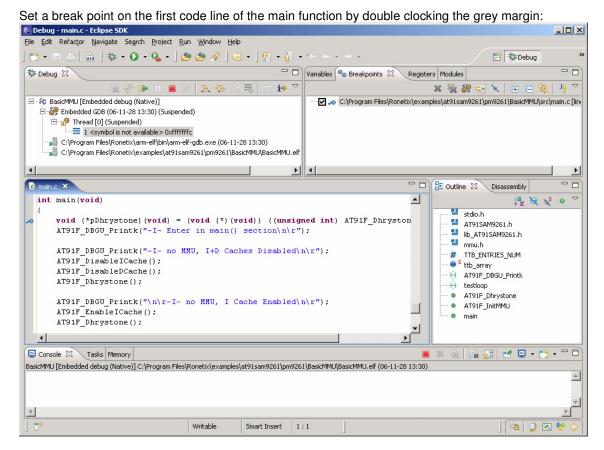
- to connect to your PEEDI
- to use big blocks for memory load, this speeds up the transfer
- to set the CPU CPSR register so all interrupts are disabled
- to load the project executable file to the target

Next clock Apply and Debug, this will start the debugger and Eclipse will ask you:

E Conf	irm Perspective Switch		
?	This kind of launch is configured to open the Debug perspective when it suspends.		
	This Debug perspective is designed to support application debugging. It incorporates views for displaying the debug stack, variables and breakpoint management.		
	Do you want to open this perspective now?		
Ren	nember my decision		
	Yes No		

Check remember my decision and click YES. Now the debugger is started:





Next click the Resume botton. The application is started and shortly it should break were the break point is:

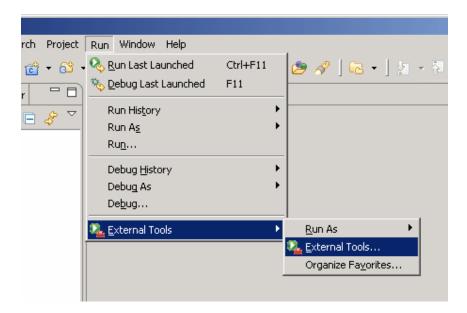
Debug - main.c - Eclipse SDK			
e <u>E</u> dit Refac <u>t</u> or <u>N</u> avigate Se <u>a</u> rch <u>P</u> roject <u>R</u> un <u>W</u> indow <u>H</u> elp			
월 • 📰 🍐 🔚 🛛 🎋 • 🔕 • 隆 🖉 🥔 🖉 🕹 • 🛛 🖅			
	Variables 💁 Breakpoints 🖾 Registers Modules 🗖 🗖		
🍇 🚸 🕪 🗉 🔳 😣 🔁 🖂 📆 📝 🏟 🎽	🗙 💥 🗞 🕾 🖉 🗡 🖽 🖻 🔁 👬 .		
B=sictMMU [Embedded debug (Native)] D=sictMMU [Embedded GDB (06-11-28 13:39) (Suspended) D=uP Thread [0] (Suspended) D=1 main() at C:\Program Files\Ronetix\examples\at91sam9261\pm9261\Basic C:\Program Files\Ronetix\examples\at91sam9261\pm9261\BasicMMU\BasicMMU.elf C:\Program Files\Ronetix\examples\at91sam9261\pm9261\BasicMMU\BasicMMU.elf			
	<b>▼</b>		
main.c 🕅	🗖 🗖 Outline 励 Disassembly 🕱 👘		
<pre> }  f  int main(void) { void (*pDhrystone)(void) = (void (*)(void)) ((unsigned int) AT91F_Dhryst AT91F_DBGU_Printk("-I- Enter in main() section\n\r"); AT91F_DBGU_Printk("-I- Enter in main() section\n\r"); AT91F_DisableICache(); AT91F_DisableICache(); AT91F_DisableDCache(); AT91F_DisableDCache(); AT91F_Dbrystone(); AT91F_DBGU_Printk("\n\r-I- no MMU, I Cache Enabled\n\r"); AT91F_DBGU_Printk("\n\r-I- no MMU, I Cache Enabled\n\r"); AT91F_DBGU_Printk("\n\r-I- no MMU, I Cache Enabled\n\r"); </pre>			
Console X Tasks Memory sichMU [Embedded debug (Native)] C:\Program Files\Ronetix\examples\at91sam9261\pm9261\			
1			
 ₽¢	🗿 🔾 🔜 🧐 🖉		

ARM cross development

At this point you can continue debugging, i.e. step, put breaks, resume, etc.

### 5.3 Using Insight debugger as an Eclipse external tool

I you prefer you can use Insight for a debugger when developing with Eclipse IDE. For this reason you need to add it as an external tool by clicking Run->External Tools->External Tool...



In the "External Tools" dialog select Program and click the New button:

External Tools				
Create, manage, and run configurations Run a program				
type filter text	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. Press the 'Filter' button to configure filtering options. Edit or view an existing configuration by selecting it. Configure launch perspective settings from the <u>Perspectives</u> preference page.			
?	Run Close			

In the new dialog for name type "Insight", for location click Browse and navigate to "Ronetix\armelf\bin\arm-elf-insight.exe", for Working Directory click Browse Workspace and select BasicMMU and for Arguments type BasicMMU.elf.

External Tools				
Create, manage, and run configurations Run a program				
Line filter text	Name:       Insight         Image:       Main       Image: Refresh       Image: Environment       Image: Common         Location:       Image: C:\Program Files\Ronetix\arm-elf\bin\arm-elf-insight.exe       Image: Browse Workspace       Browse File System       Variables         Working Directory:       \${workspace_loc:/BasicMMU}       Image: Browse File System       Variables         More Workspace_loc:/BasicMMU       Browse File System       Variables         Arguments:       Image: Directory:       Image: Directory:         Variables:       Image: Directory:       Image: Directory:         Arguments:       Image: Directory:       Image: Directory:         Note:       Enclose an argument containing spaces using double-guotes (").			
	Apply Reyert			
0	Run Close			

Now click on Common, and in the "Display in favorites menu" select Insight to add it to the Run menu and click Apply and Close:

External Tools	×	
Create, manage, and run configurations		
Run a program		
type filter text	Name:       Insight         Main       Refresh       Environment         Save as       Local file         Shared file:       Browse         Display in favorites menu       Console Encoding         One fault (Cp1251)       Other ISO-8859-1         Standard Input and Output       Iso-8859-1	
	✓ Allocate Console (necessary for input)         ✓ File:         Browse Workspace         Browse File System         Variables         ✓ Append         ✓ Launch in background	
	Apply	
0	<u>R</u> un Close	

Now you have Insight in your Run menu.

ect	Run Window Help		
2 -	. 🗞 <u>R</u> un Last Launched	Ctrl+F11	😕 🔗 🛛 🔂 🔸 🛛 🖢 🔸
Ð	🍇 Debug Last Launched	F11	
$\exists$	Run His <u>t</u> ory	•	-
	Run A <u>s</u>	•	
	Ru <u>n</u>		
	Debug <u>H</u> istory	•	
	Debug As	•	
	De <u>b</u> ug		
	🍋 External Tools	Þ	🍇 <u>1</u> Insight
			<u>R</u> un As ►
			💁 External Tools
			Organize Fa <u>v</u> orites

You have just added the Insight.

You can follow the previous procedure to add any external tool you wish to use while developing. Now you can start Insight and debug the project.