

Lab Assignment 1. Installation of a Cross Compilation and Simulation Environment for the LEON 3 Processor

1. Introduction

This document describes the installation process of a cross-compiling environment for the LEON3 processor. The document also explains how to integrate, in this environment, the control of the execution of programs on a simulator of this processor called TSIM2. LEON3 belongs to the SPARC family, and it has been selected as a reference processor by the European Space Agency (ESA) for its space missions. For this reason, numerous satellites use LEON3 in the design of the embedded system, called On-Board Data Handling (OBDH), that controls the different satellite subsystems and their data.

The development environment has been installed on the Linux operating system. The specific distribution on which the installation was performed was Ubuntu 18.04 (Bionic Beaver), although the installation steps can be considered analogous in any other Linux distribution. A solution to avoid any problems associated with the distribution is to use a virtual machine built from Ubuntu 18.04 as the one that can be downloaded from this url: <https://releases.ubuntu.com/18.04/> If you are using a 64-bit Ubuntu installation (**as at the university laboratory**), the installation of the default packages should be completed as follows:

```
sudo dpkg --add-architecture i386
sudo apt-get update
sudo apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386
sudo apt-get install libnspr4-dev:i386 zlib1g:i386 libx11-6:i386
sudo apt-get install build-essential
```

In addition, for any of the installations (32 bits or 64 bits) of Ubuntu, it is necessary to have installed the Java run-time that allows you to run Java applications such as the Eclipse development environment that will be used in the laboratory. To install it, use the apt-get command as follows (If you are on the university's workstation, the user is atcsol, and this package has already been installed):

```
sudo apt-get update
sudo apt-get install openjdk-8-jre
```

The following sections will list the steps required to make the installation effective, as well as some tests to check that the installation has been completed correctly.

2. Eclipse environment installation

The C/C++ projects to be created on the LEON3 processor can be managed from an integrated development environment such as Eclipse. This environment can be easily installed within the ubuntu 18.04 linux distribution by following the next steps:

1. Download the tar.gz file containing the files from the Eclipse environment configured to work with C/C++ projects. (If you are at the university laboratory, this file has been downloaded) This file, for the kepler version of Eclipse, can be downloaded directly from the link:

32 bits:

<http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/kepler/SR2/eclipse-cpp-kepler-SR2-linux-gtk.tar.gz>

64 bits:

http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/kepler/SR2/eclipse-cpp-kepler-SR2-linux-gtk-x86_64.tar.gz

2. Move the file to the /opt directory

```
sudo mv eclipse-cpp-kepler-SR2-linux-gtk.tar.gz /opt
```

3. Unzip the file in the same directory

```
cd /opt/  
sudo tar -xzf eclipse-cpp-kepler-SR2-linux-gtk.tar.gz
```

4. Add (using, for example, gedit or vim) **at the end of the /home/user/.profile** file (or **/home/atcsol/.profile if you are in the university lab**) the following line that modifies the PATH in order to include the eclipse binaries directory:

```
PATH="/opt/eclipse:$PATH"
```

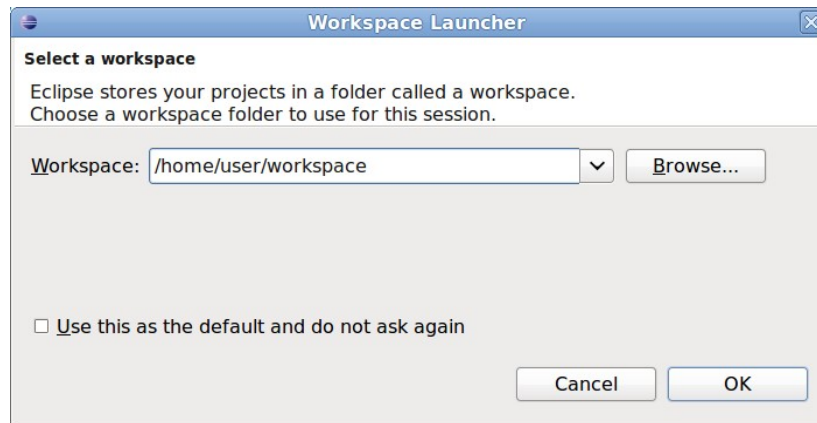
5. Execute the script ./profile with the command:

```
source .profile
```

6. run eclipse and check that the application opens:

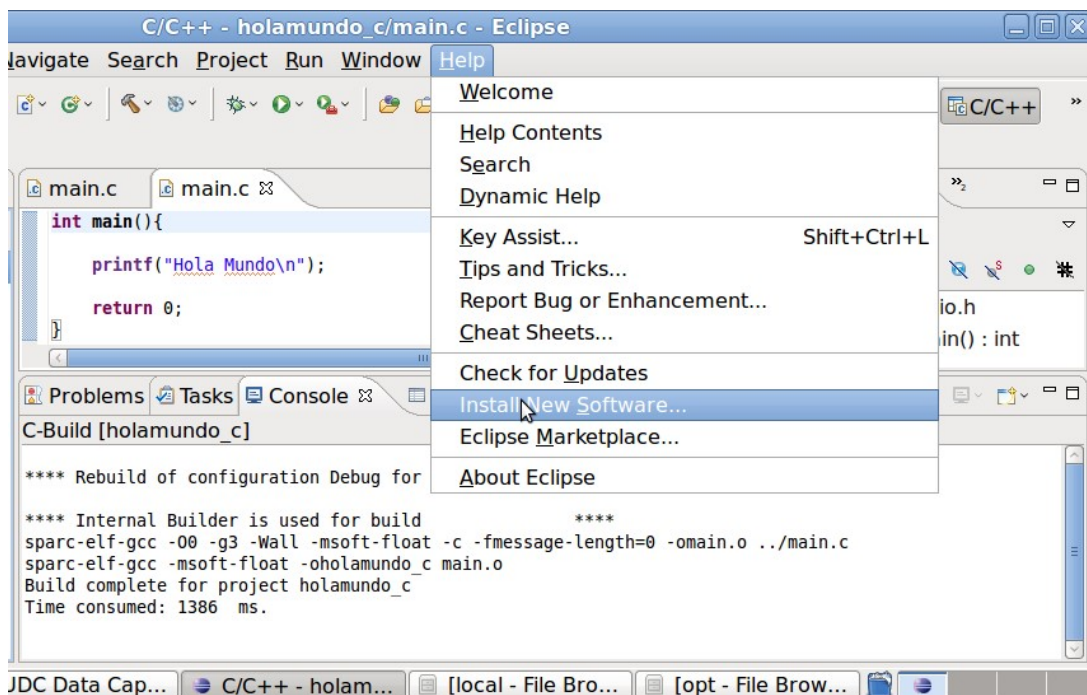
```
eclipse
```

7. Accept the name of the workspace proposed by the following initial window of eclipse:

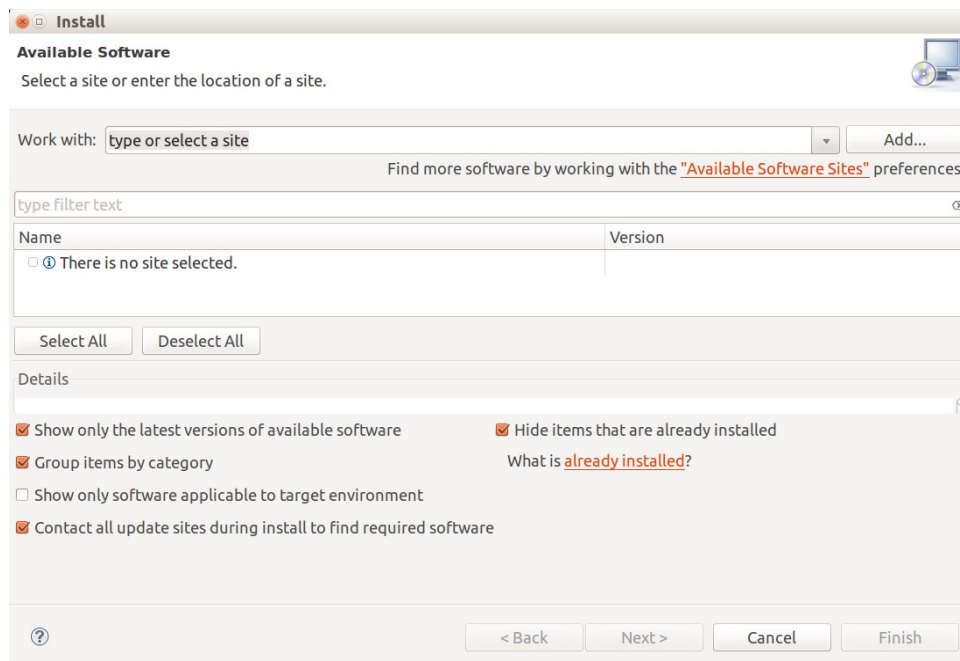


3. Eclipse Plugin for Software Project Configuration Control

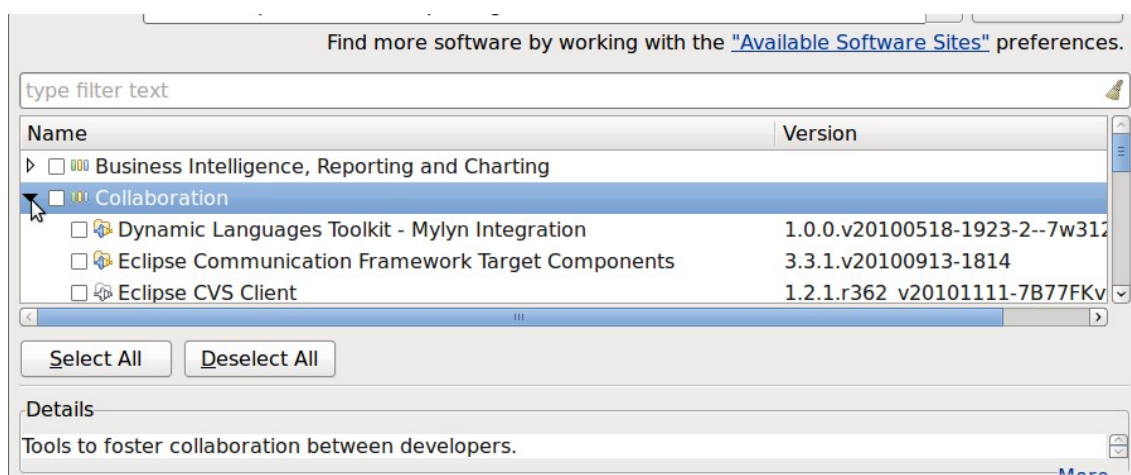
The software project configuration control allows engineers to have at all times the traceability of changes that have occurred in the source code of their projects, as well as to associate labels to the project status in specific phases of development. For the embedded systems laboratory, we are going to use as a configuration control tool a Subversion client integrated as a plugin in the Eclipse environment. For its installation, it is necessary to use *Install New Software* de Eclipse.



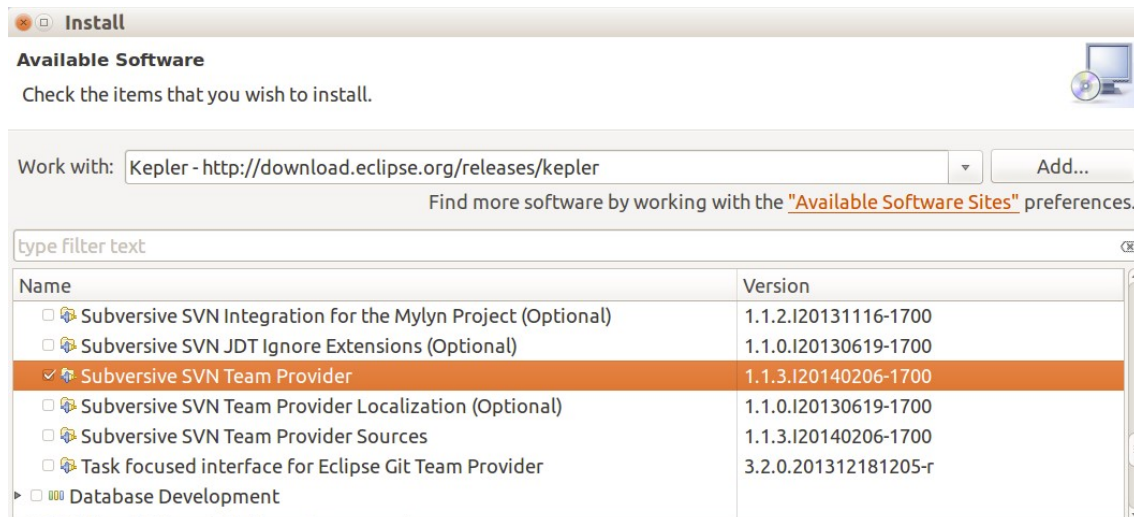
Then, select from the *Working With* drop-down list the following software site:
Kepler <http://download.eclipse.org/releases/kepler>.



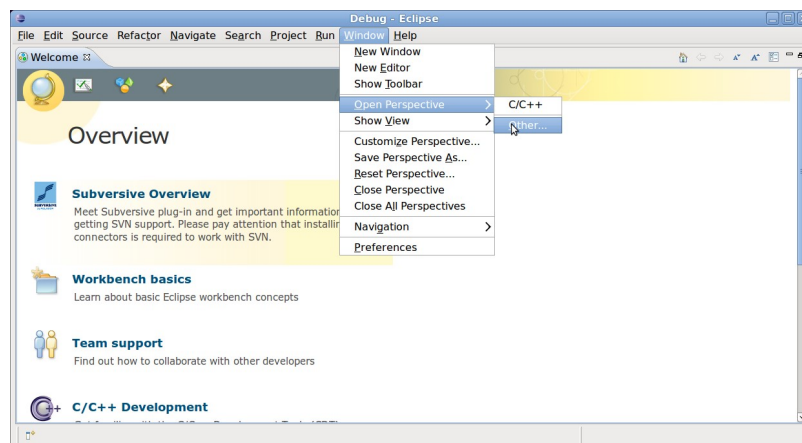
Expand the list of items that correspond to the *Collaboration* group



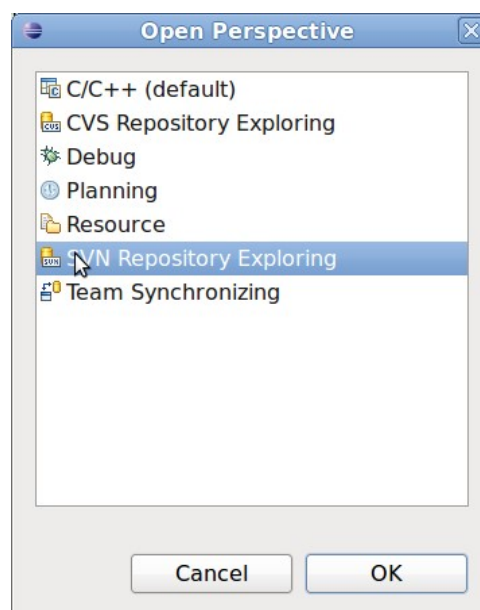
Select the last item in the list *Subversive SVN Team Provider* (Incubation), and select *Next* to start the installation. After accepting all installation steps, the restart of Eclipse will be requested and must be accepted.



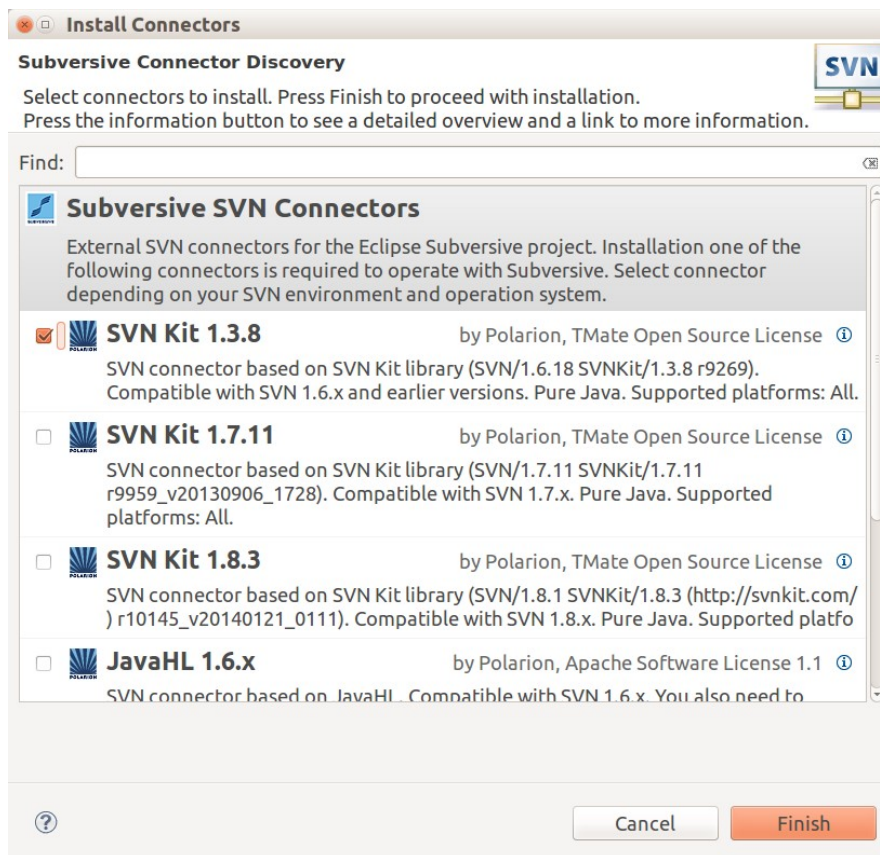
After reset, the menu *Open Perspective/Other* is used to start the utilization of the *Subversion (SVN)* client.



Select the perspective *SVN Repository Exploring*.



and select the connector *SVN Kit 1.3.8* to finish the SVN client installation.



(If this menu does not appear, install SVN Kit 1.3.8 using again *Install New Software* with <http://community.polarion.com/projects/subversive/download/eclipse/3.0/kepler-site/>)

Close eclipse to continue with the installation of the rest of the tools

4. Installing the BCC Cross Compiler

The BCC (Bare-C Cross Compiler) cross compiler is a cross compiler for the LEON2 and LEON3 processors. The compiler distribution also includes the C Newlib standalone library and low-level routines for input and output on LEON2 and LEON3. Finally the distribution also provides the Mkprom utility that allows to create, from a RAM executable software system, a new software executable from a non-volatile memory (PROM or EEPROM) and whose objective is only to deploy in RAM the original system and pass the control to it.

The installation of the BCC compiler includes the following steps:

1. Download the BCC distribution `sparc-elf-4.4.2-1.0.50.tar.bz2` using the following link (If you are at the university laboratory, this file has been downloaded):

<http://www.gaisler.com/anonftp/bcc/bin/linux/sparc-elf-4.4.2-1.0.50.tar.bz2>

2. Unzip it in the /opt directory

```
sudo tar -C /opt -xjf sparc-elf-4.4.2-1.0.50.tar.bz2
```

3. Create a soft link with the name sparc-elf

```
cd /opt
sudo ln -s sparc-elf-4.4.2 sparc-elf
```

4. Add (using, for example, gedit or vim) **at the end of the /home/user/.profile file (or /home/atcsol/.profile if you are in the university lab)** the following line that modifies the PATH in order to include the BCC binaries directory:

```
PATH="/opt/sparc-elf/bin:$PATH"
```

5. *Installing the TSIM2 simulator*

TSIM2 simulates the LEON3 processor. Its installation includes the following steps:

1. Due to a change in the evaluation version of the LEON3 processor simulator (it has changed to dual core), in the lab we are going to use an old version of the simulator, TSIM2, which is a 2020 evaluation version, so for this course to work we will need to provide the application with a 2020 date. To do this, we will need to install faketime using the following command

```
sudo apt-get install faketime
```

2. After that, it is necessary to download the evaluation version of the LEON3 TSIM2 simulator tsim-eval-2.0.65.tar.gz using the following link and move it to the /opt directory:

<http://www.gaisler.com/anonftp/tsim/tsim-eval-2.0.65.tar.gz>

```
sudo mv tsim-eval-2.0.65.tar.gz /opt
```

3. Unzip it in the /opt directory

```
cd /opt
sudo tar -xvzf tsim-eval-2.0.65.tar.gz
```

4. Download from the blackboard the TSIM2 launcher that use faketime tsim-launch.tar.gz , move it to /opt and unzip it there

```
sudo mv tsim-launch.tar.gz /opt
cd /opt
sudo tar xvfz tsim-launch.tar.gz
```

5. Create a soft link with the name tsim

```
sudo ln -s tsim-launch/tsim/linux tsim
```

6. Add (using, for example, gedit or vim) **at the end of the /home/user/.profile** file (or **/home/atcsol/.profile if you are in the university lab**) the following line that modifies the PATH in order to include the TSIM2 binaries:

```
PATH="/opt/tsim:$PATH"
```

7. Force the execution of the `./profile` script with the following command:

```
source .profile
```

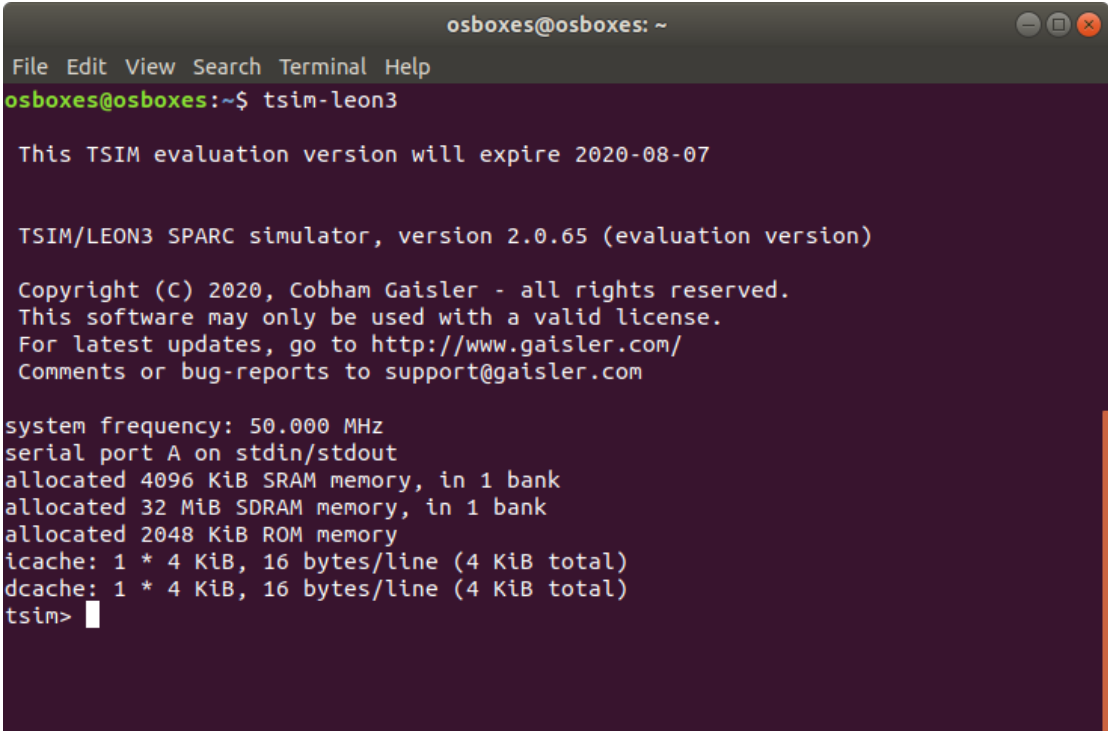
8. Show the current PATH and check that it includes the directories `/opt/tsim` , `/opt/sparc-elf/bin` and `/opt/eclipse`.

```
echo $PATH
```

9. Launch the `tsim-leon3` simulator using the following command

```
tsim-leon3
```

The screen will show the following initialization log:



```
osboxes@osboxes: ~  
File Edit View Search Terminal Help  
osboxes@osboxes:~$ tsim-leon3  
  
This TSIM evaluation version will expire 2020-08-07  
  
TSIM/LEON3 SPARC simulator, version 2.0.65 (evaluation version)  
  
Copyright (C) 2020, Cobham Gaisler - all rights reserved.  
This software may only be used with a valid license.  
For latest updates, go to http://www.gaisler.com/  
Comments or bug-reports to support@gaisler.com  
  
system frequency: 50.000 MHz  
serial port A on stdin/stdout  
allocated 4096 KiB SRAM memory, in 1 bank  
allocated 32 MiB SDRAM memory, in 1 bank  
allocated 2048 KiB ROM memory  
icache: 1 * 4 KiB, 16 bytes/line (4 KiB total)  
dcache: 1 * 4 KiB, 16 bytes/line (4 KiB total)  
tsim> 
```

10. Launch the `paranoia` test inside `tsim-leon3`, and check if the message “Program exited normally” appears

```
load /opt/tsim-eval/tests/paranoia  
run
```

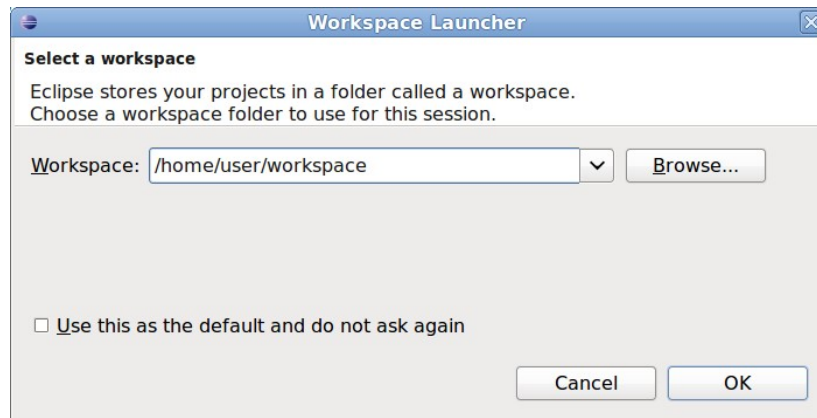
11. Use `quit` to exit from `tsim-leon3`.

6. *Installation of the Eclipse plugin for the integration of the BCC cross compiler and the TSIM2 simulator*

1. **Open a terminal**, and execute eclipse after loading the PATH of the .profile

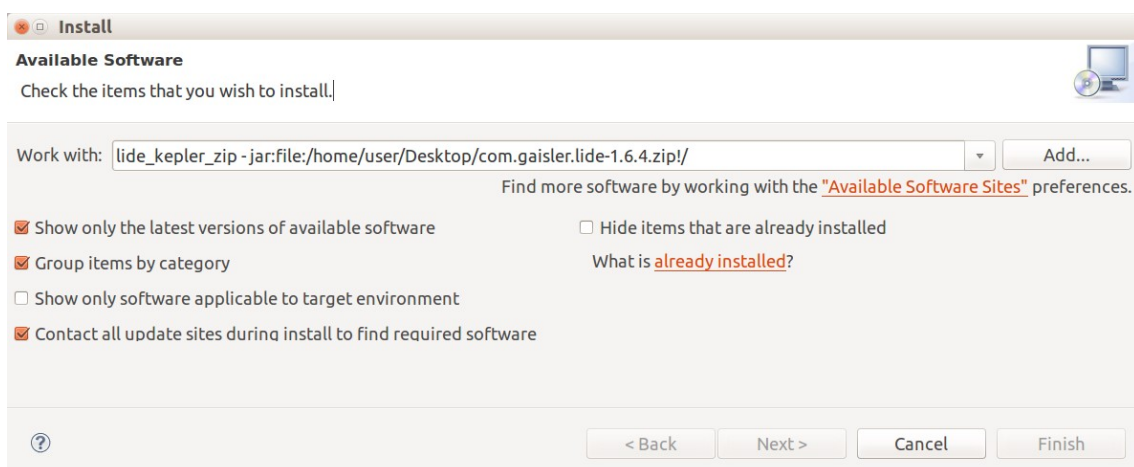
```
source .profile  
eclipse
```

2. Accept the name of the workspace that eclipse proposes:

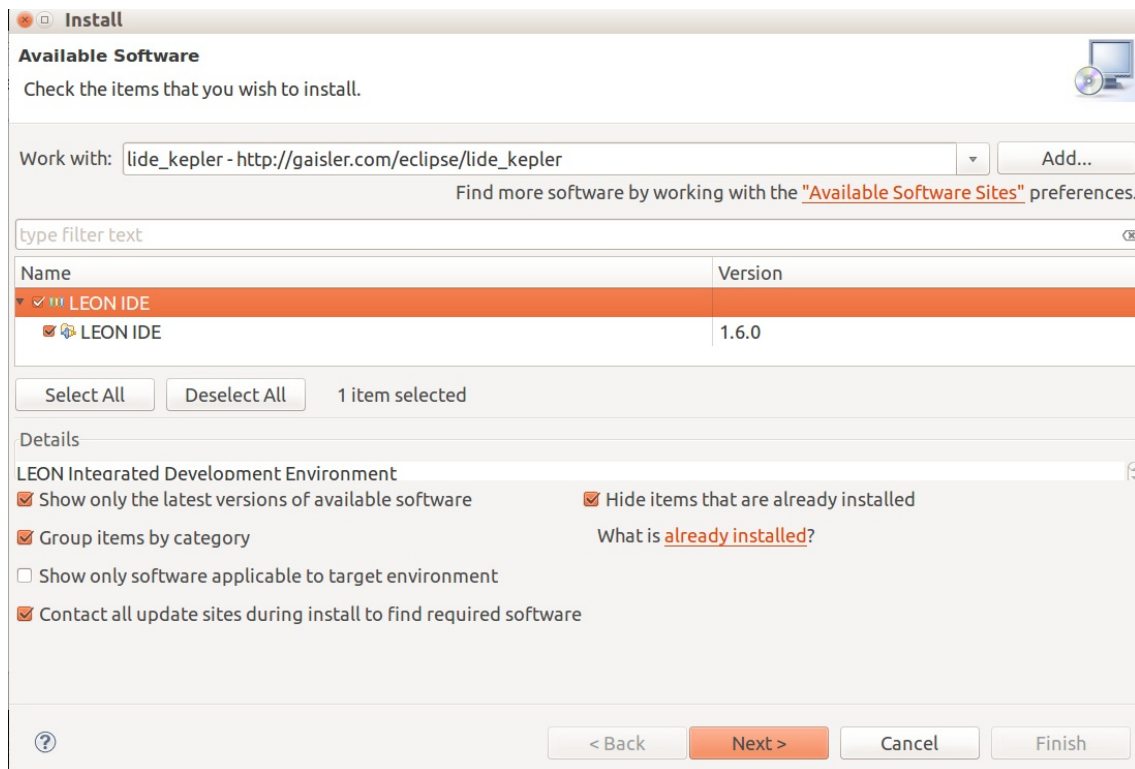


In order to integrate the compiler and simulator with the Eclipse development environment in a simple way we will use the Gaisler plugin called CDT Plugin 1.6.0. The installation of this plugin includes the following steps:

3. Select the Eclipse menu **Help->Install New Software**
4. Add a new repository using the **Add...** button, and include the **lide_kepler plugin** using the .zip file of the following URL <https://www.gaisler.com/eclipse/com.gaisler.lide-1.6.4.zip> (Use the **Archive** option) If you are at the university laboratory, this file has been downloaded.

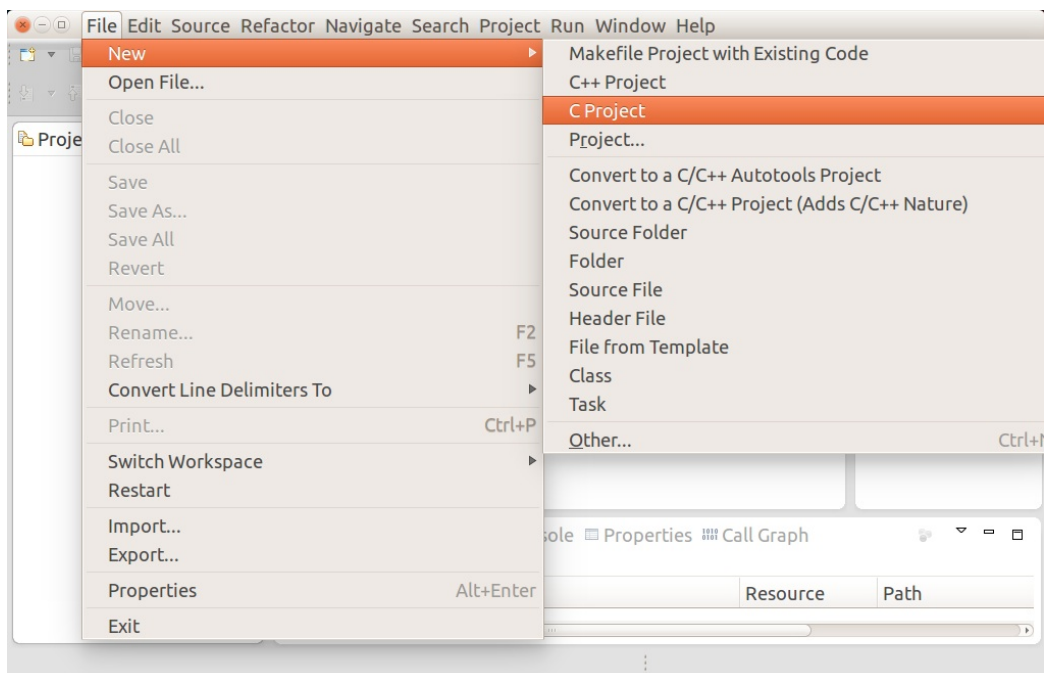


5. Select LEON IDE, as shown in the following figure, and accept the rest of the menus, including the one that forces the eclipse reset.

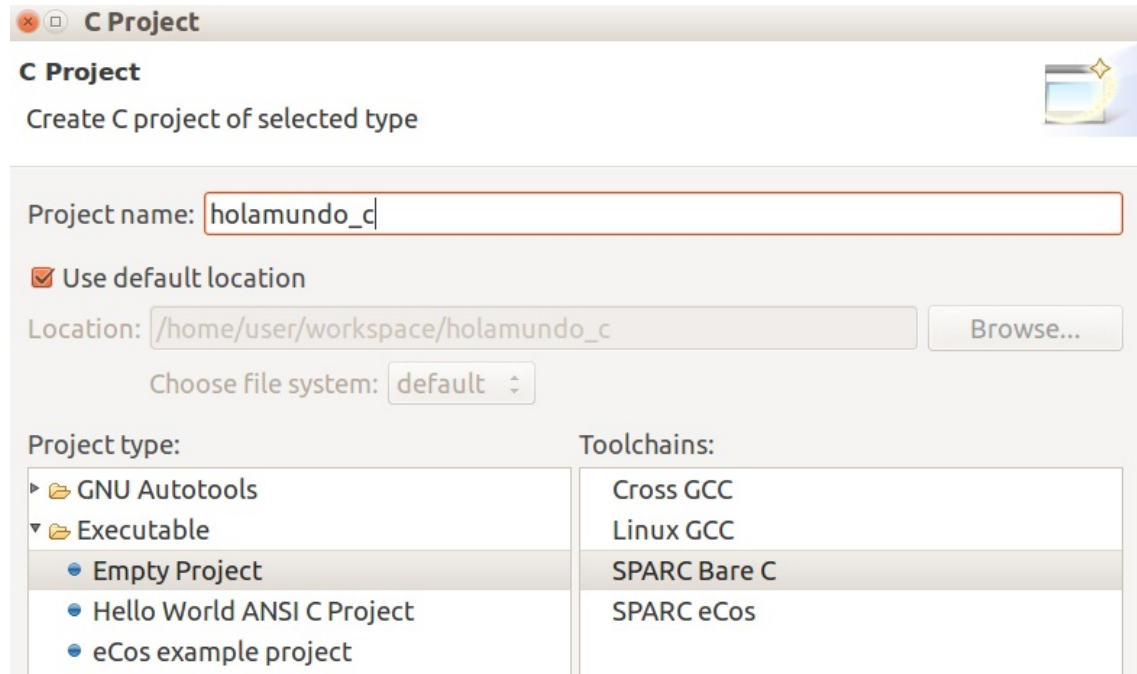


6. BCC project creation

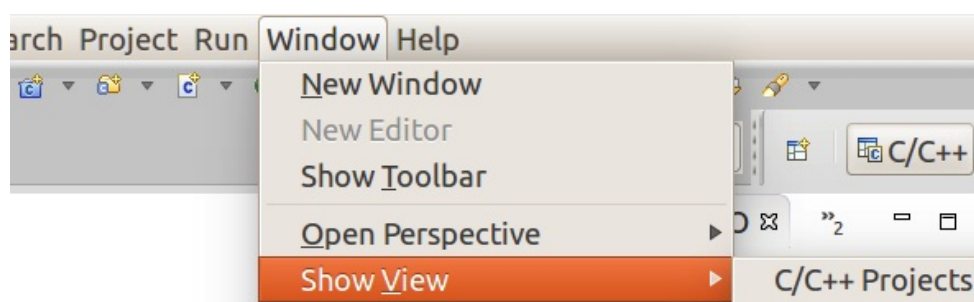
In order to create a C project for the BCC cross compiler in which the Makefile is generated automatically select the menu **File->New-> C Project**:



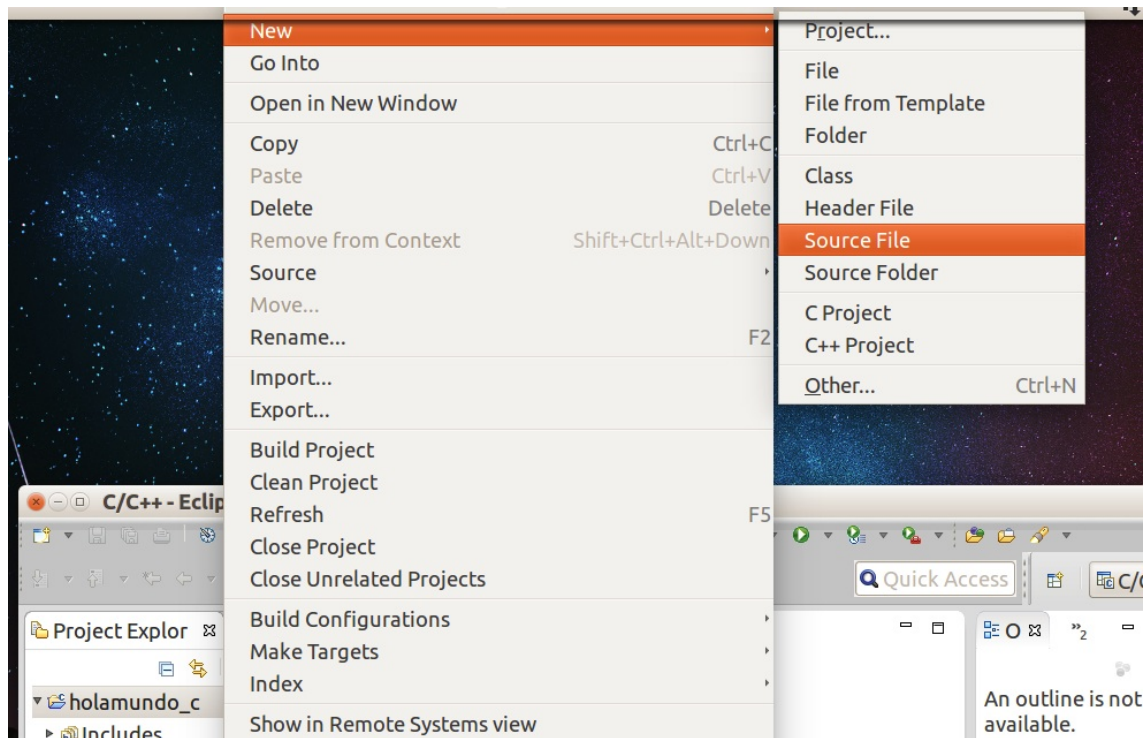
Then, you have to assign a name to the project (`holamundo_c` in this case) and define the type of project as **Executable Empty Project**, and **Sparc Bare C** as *Toolchain*, in order to use the cross compiler *sparc-elf*, that generates code for an SPARC processor, as LEON3.



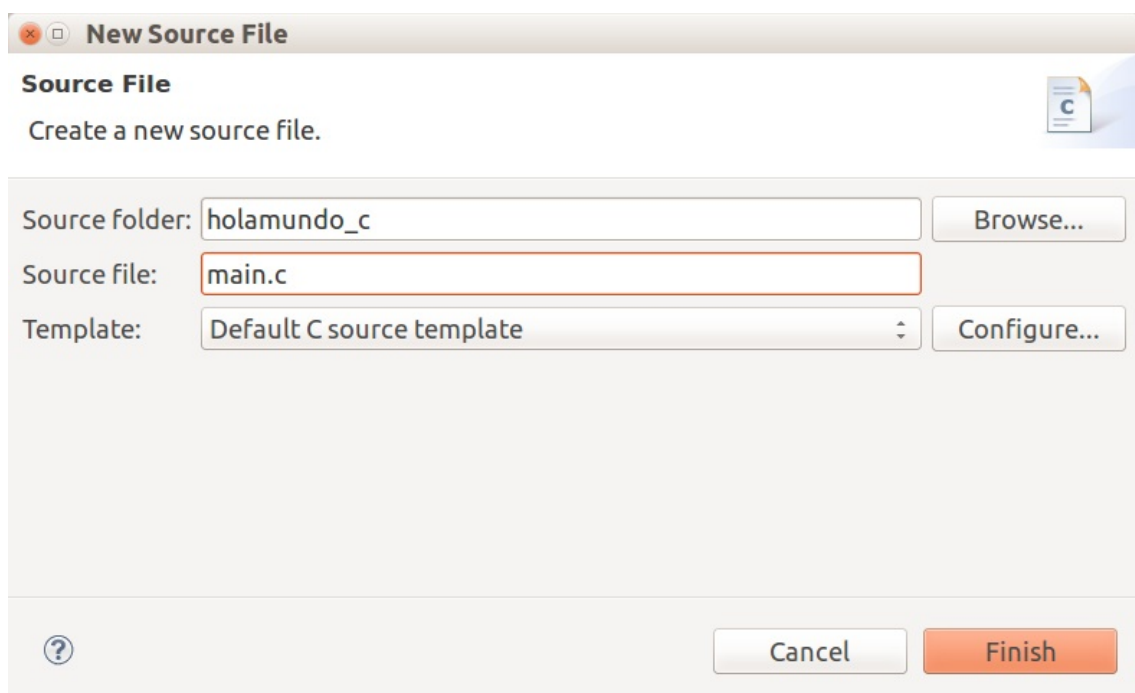
Once the project is created, select the C/C++ view using the menu **Window->ShowView> C/C++ Projects**



and add a source file using the right button on the `holamundo_c` project



, and set the file name **main.c**



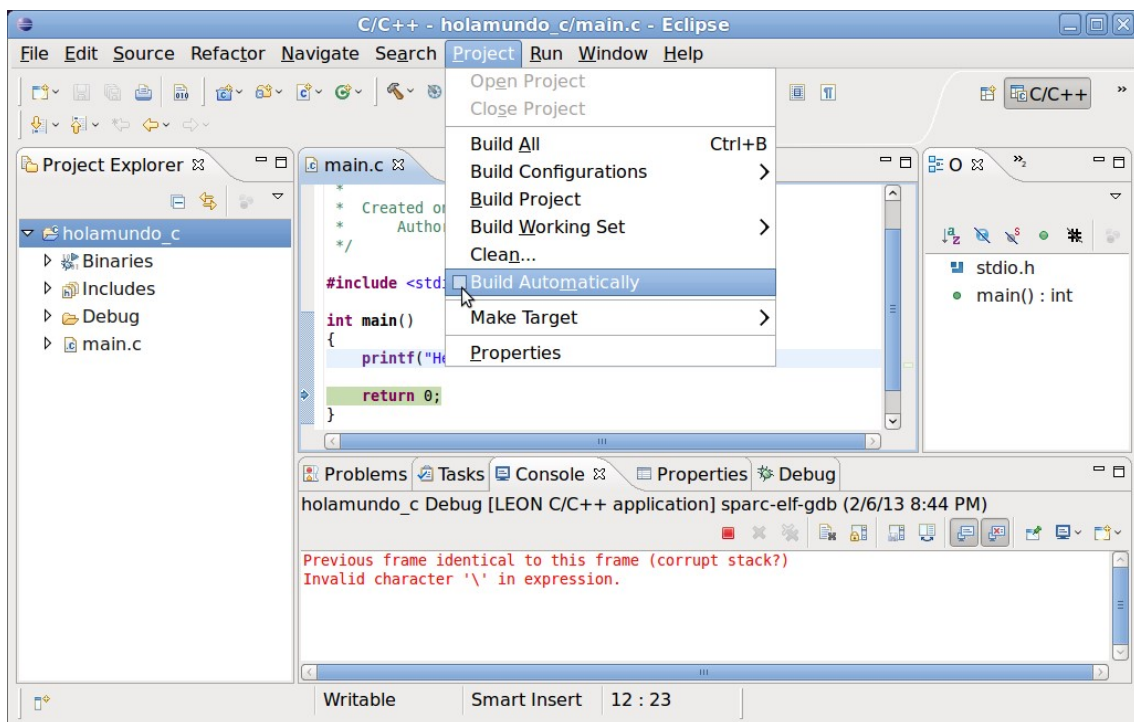
Use double-click in order to edit the file. Add the following "hello" code:

```
#include <stdio.h>

int main()
{
    printf("Hello\n");

    return 0;
}
```

According to the initial configuration of Eclipse, the executable of this project is created automatically after saving it. This configuration can be changed using **Project** menu and unchecking the **Build Automatically** option.

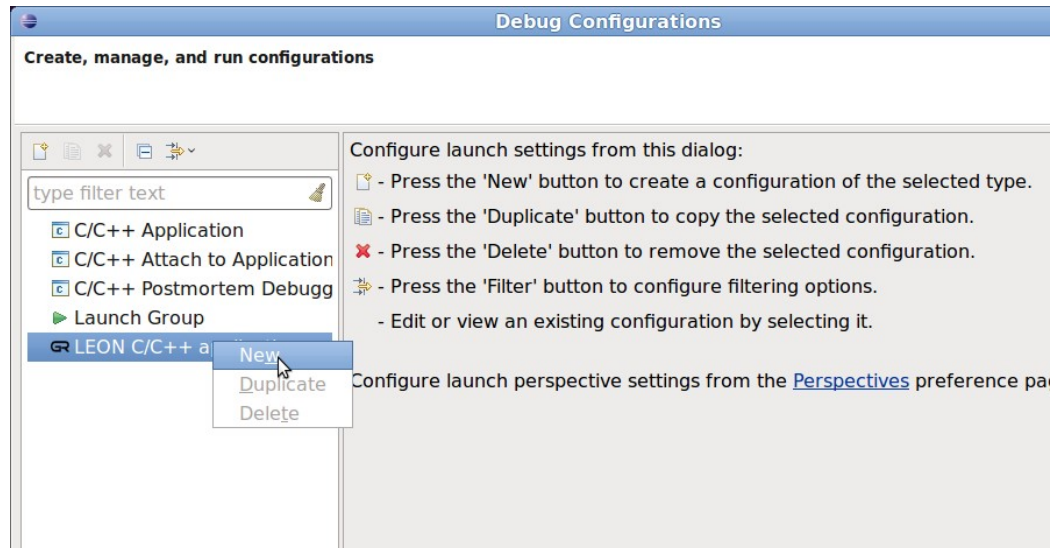


When the automatic configuration is disabled, the executable can be built using the **Project->Build Project** menu

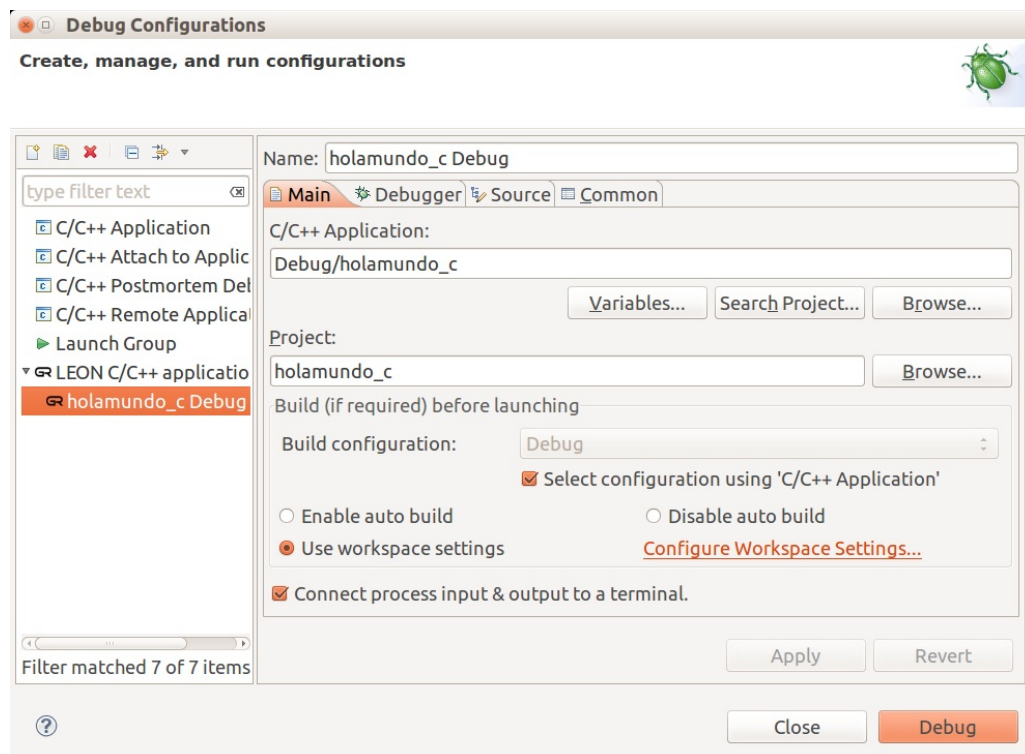
7. Debugging the project on the TSIM2 simulator

In order to execute the program, it is necessary to create a launcher that is configured to work with the TSIM2 simulator. The following steps are required:

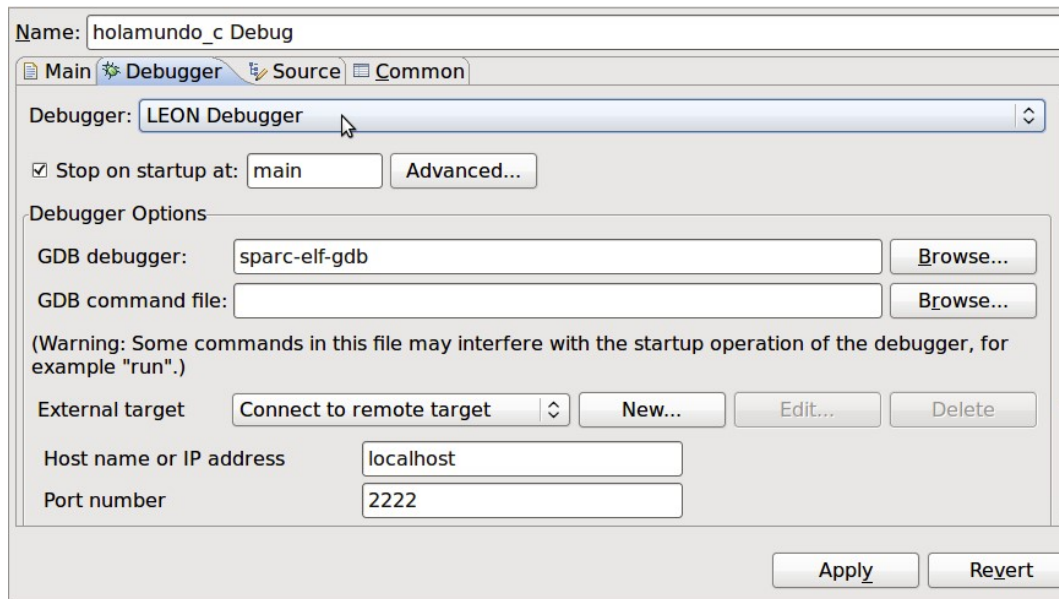
1. Access to the debug configuration menu via the **Run->Debug Configurations...** menu
2. Create a new launcher for an LEON C/C++ application



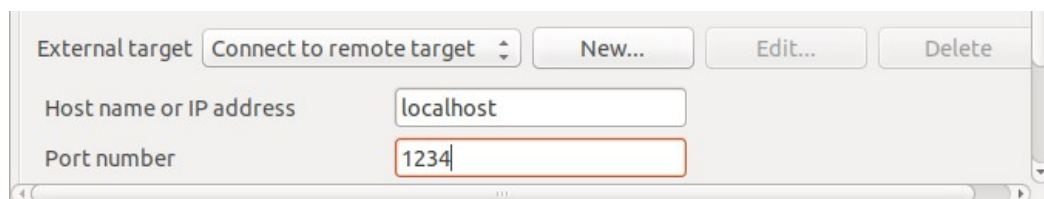
3. Check that the **Main** configuration is as follows:



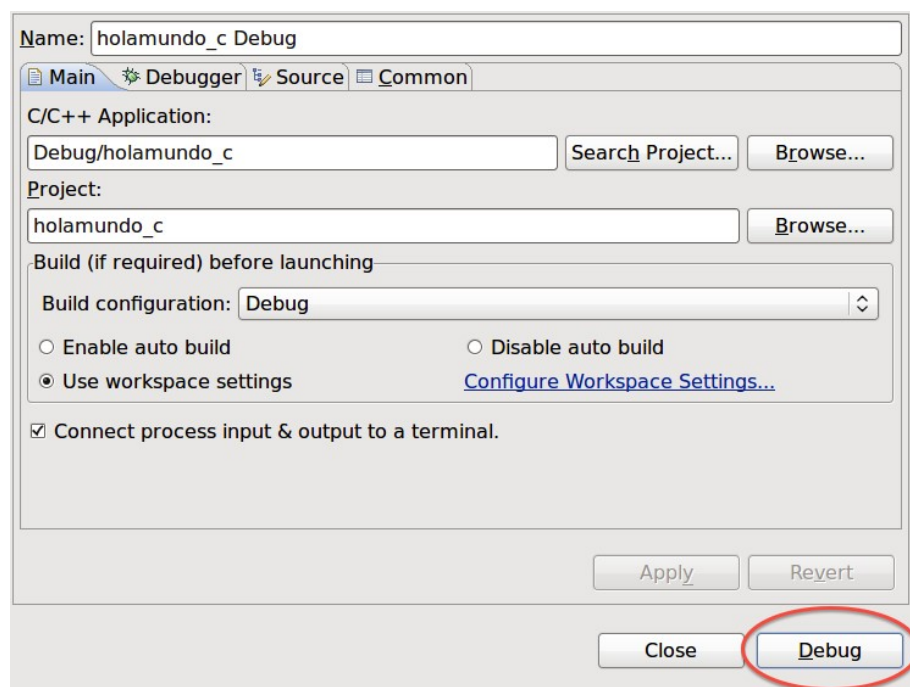
4. Select **LEON Debugger** as **Debugger** and **sparc-elf-gdb** as **GDB debugger**:



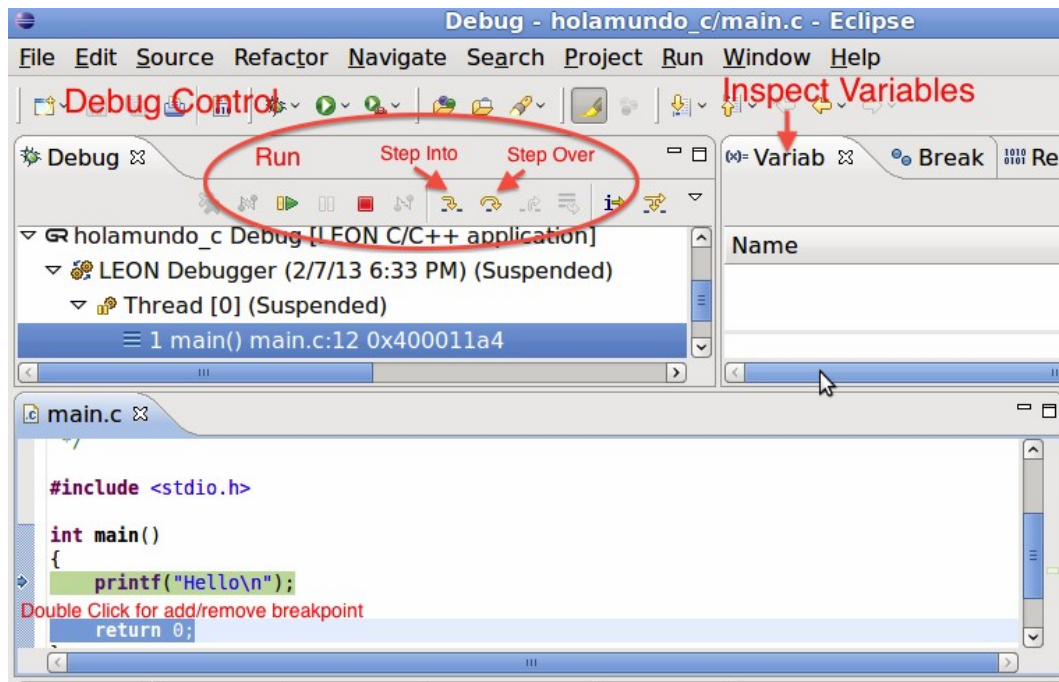
5. Change the *Port number* to 1234 to connect gdb with TSIM2



6. Finally, start debugging using the **Debug** button:



- From **Debug** view, it is possible to order the continuous execution (**Run**), the step by step execution (**Step Into** y **Step Over**), the breakpoints setting (**Double Click for add/remove breakpoint**) and the variables inspection (**Inspect Variables**).



- Execute the program by selecting **Run** and check on the TSIM2 console that the "Hello" message appears.

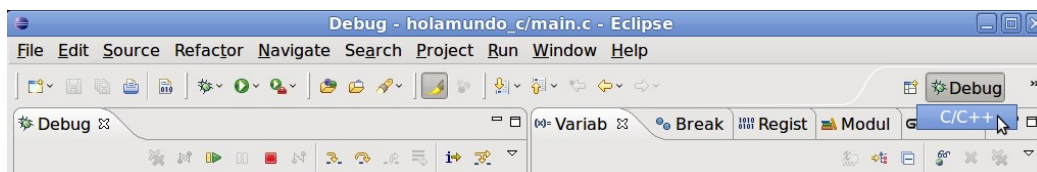
```
system frequency: 50.000 MHz
serial port A on stdin/stdout
allocated 4096 KiB SRAM memory, in 1 bank
allocated 32 MiB SDRAM memory, in 1 bank
allocated 2048 KiB ROM memory
icache: 1 * 4 KiB, 16 bytes/line (4 KiB total)
dcache: 1 * 4 KiB, 16 bytes/line (4 KiB total)
tsim> gdb
gdb interface: using port 1234
Starting GDB server. Use Ctrl-C to stop waiting for connection.
connected
Hello
gdb: disconnected
gdb interface: using port 1234
Starting GDB server. Use Ctrl-C to stop waiting for connection.
```

- Since the program has finished, we are going to reset the processor from TSIM2 in order to be able to load a new program in the next debugging. To do this we must first use **Ctrl+C** to return to the TSIM2 command line, and then use the `reset` command which resets the processor. Finally, we again use the `gdb` command so that TSIM2 is ready to load a new program from eclipse.

```
Starting GDB server. Use Ctrl-C to stop waiting for connection.  
^C  
tsim> reset  
tsim> gdb  
gdb interface: using port 1234  
Starting GDB server. Use Ctrl-C to stop waiting for connection.
```

Before each debug, we must make sure that TSIM2 is ready waiting for gdb after a reset, so repeat the sequence Ctrl+C, reset, gdb to prevent the load or execution from failing.

10. Go back to the C/C++ view

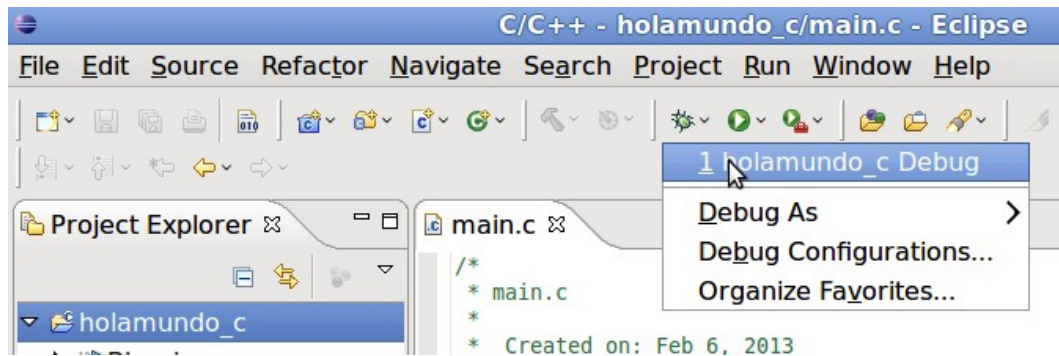


11. Modify **main.c** in order to create a loop that shows *Hello* 10 times.

```
#include <stdio.h>  
  
int main()  
{  
    int i;  
    for(i=0; i< 10; i++)  
        printf("Hello\n");  
  
    return 0;  
}
```

12. Save the file and use **Build->Project** to build the executable.

13. Launch the debugging of the new executable using the following drop-down menu.



14. Add a **breakpoint** on the sentence tha call to *printf* and start the continuous execution (**Run**) checking in the inspection panel the value of the variable *i* in each iteration.

