AndeSight™ User Manual

Working with AndESLive™

Version 1.0

Table of Contents

AndeSight™ User Manual

Preface

Welcome to AndeSight™, the integrated software development environment for embedded SoC.

About this manual

AndeSight™ consists of IDE (Integrated Development Environment) and Toolchains providing the complete and integrated environment for users to develop the software. Through working with AndESLive™, virtual evaluation platform, users will be able to run the simulation of the applications more easily and convenient.

This manual explains AndeSight™ in detail from the novice aspect and guides the users how to start to use the tools by example projects.

Version of AndeSight

This manual covers the latest release features and functionalities of AndeSight/AndESLive Developer Suite version 1.0.

Contact information

Please contact Andes Technology Corporation by email at andes-up-support@andestech.com or on the Internet at www.andestech.com for support.

Copyright notice

Chapter 1

Introduction and Overview

This chapter introduces the AndeSight™ and supporting facilities provided by Andes Technology Corporation.

AndeSight™ Development Tool contains the following section:

1.1 AndeSight Overview

AndeSight, a versatile development tool, consists of IDE (Integrated Development Environment) and supporting Toolchains. Through perfectly connecting with AndESLive, we provide a complete set of development tools for the process of writing code on one system, known as a host, and running on another system, known as a target.

This chapter provides an overview of whole development concept (as shown in *Figure 1-1)*, and an introduction to basic AndeSight features and functionality.

The following diagram indicates the full view of AndeSight and AndESLive, and the relationship between each part.

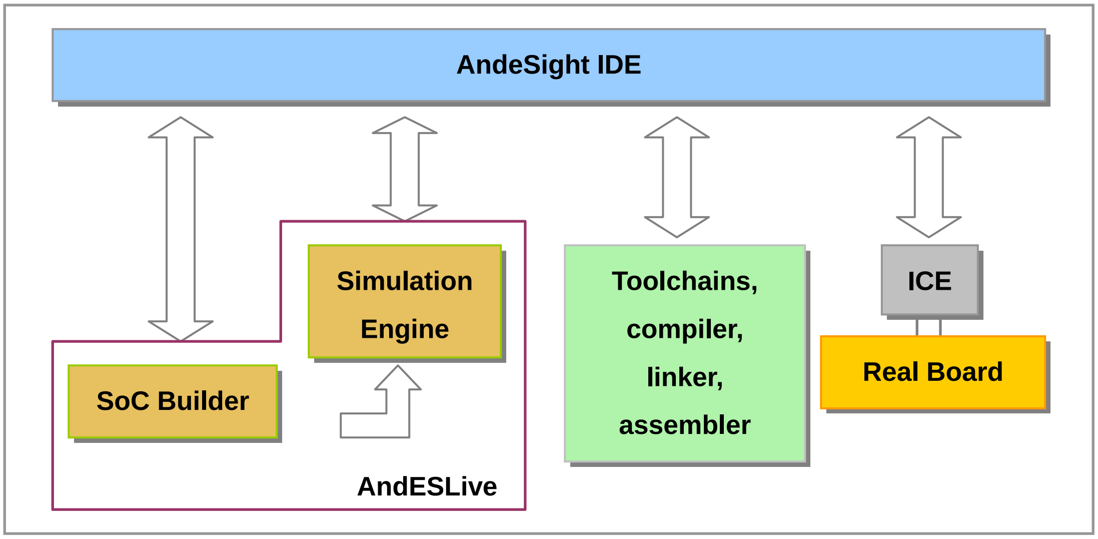


Figure 1-1: AndeSight

1.2 IDE Overview

AndeSight IDE is an Eclipse-based development suite that provides an efficient way to develop embedded applications of the target system.

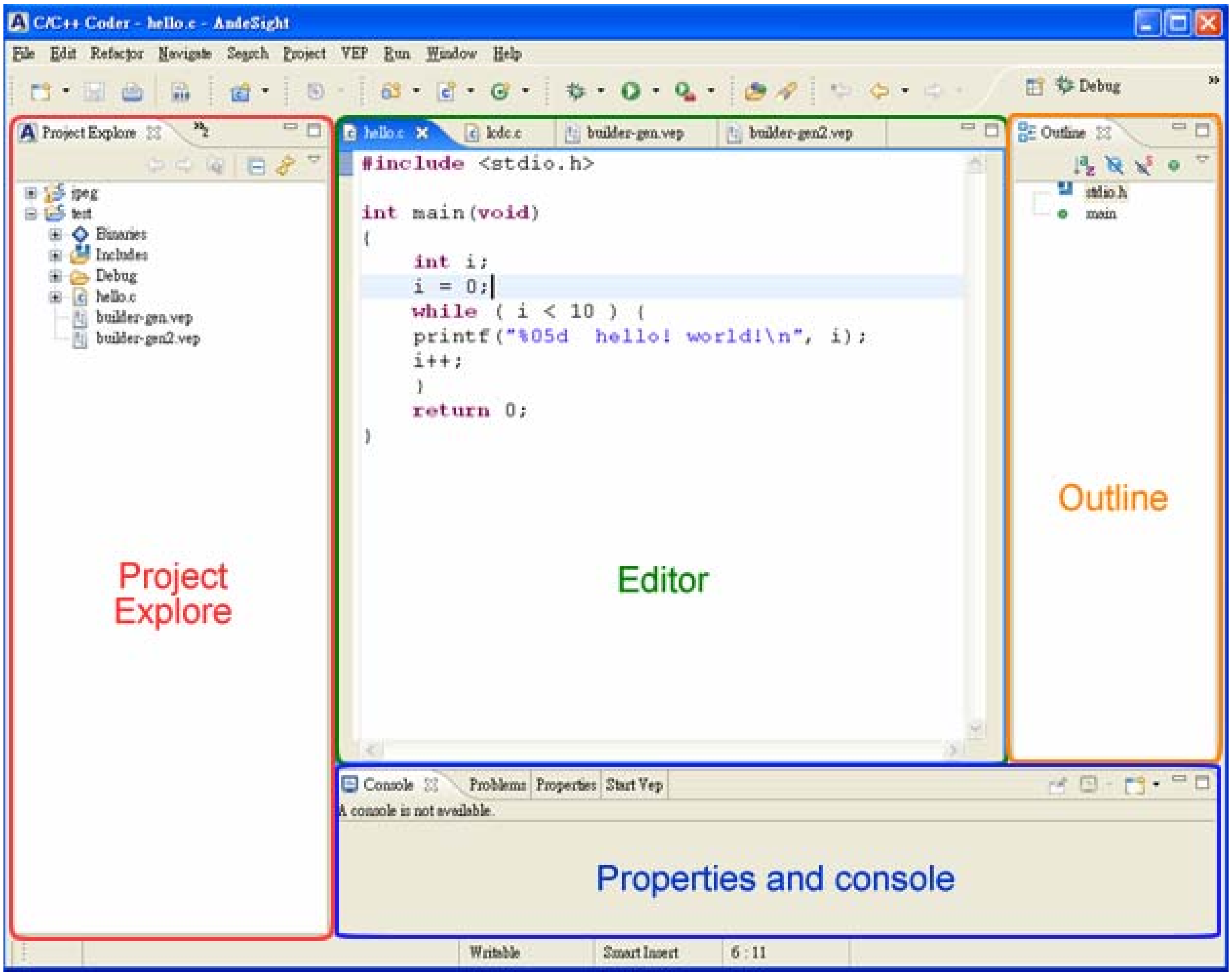
AndeSight IDE is based on Eclipse 3.1. This section provides a brief introduction to the *Workbench* features which are common to all Eclipse-based systems. For a more detailed introduction, please refer to the online **Help** (by selecting **Help > Help Contents**, expand the

Workbench User Guide entry) or go to www.eclipse.org/documentation.

AndeSight IDE is described with several terms: *Windows, Views, Editor, Perspective, and Plug-ins*. The following sections outline these terms and explain how to use these features.

1.2.1 Windows

The term *Window* is used only for the overall outer frame shown in *Figure1-2*. Use **Window > New Window** to create a new same outer frame.



```
#include <stdio.h>

int main(void)
{
    int i;
    i = 0;
    while ( i < 10 ) {
    printf("%05d  hello! world!\n", i);
    i++;
    }
    return 0;
}
```

A console is not available.

Figure1-2: Application Development Perspective

1.2.2 Views

The term *View* refers to the individual panes within a window. As *Figure1-2* shows, the *Project Explorer* view on the left side of the screen, the *Outline* view on the top-right, and the stacked view on the bottom-right with the title *Tasks*, such as console, problems and properties etc.

Many views include a menu that is accessed by clicking the down arrow to the right of the title bar. This menu typically contains items that apply to the entire contents of the view rather than a selected item within the view.

To open a view and add it to the existing perspective (see 1.2.3 Perspectives), select **Window > Show View**. Select the desired view from the list, or select **Other…** to display expandable lists with more choices. The view is added at its default location in the window, and you can move it if desired. The following section lists some frequently used views.

- **Project Explore View (as shown in *Figure1-2*)**

  The C/C++ *Project Explore* view displays the relevant elements to C and C++ project files in a tree structure. For detailed information, please refer to the help content by clicking **Help > Help Contents > Eclipse Platform Guide > C/C++ Development User Guide > Concepts > Navigation aids > Project File views.**

- **Source Code View**

  The *Source Code* view provides specialized features for editing files. Please refer to 1.2.4 Editors.

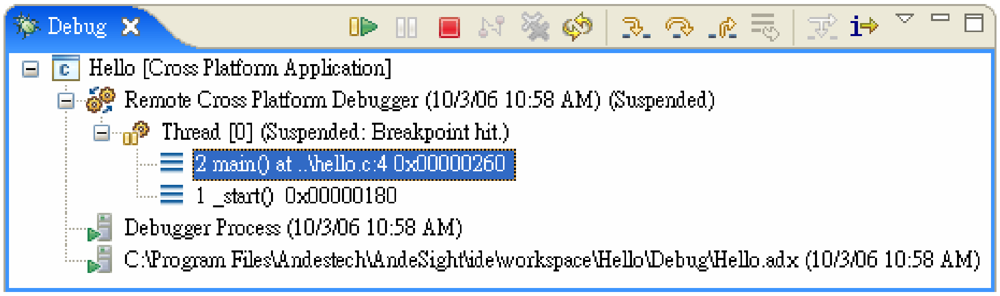
- **Properties and Console Stack (as shown in *Figure1-2*)**

  The *Properties* view displays property names and values for a selected item. For detailed information, please refer to the help content by clicking **Help > Help Contents > Eclipse Platform Guide > C/C++ Development User Guide > Reference > C/C++ Views and Editors > Properties View.**

- **Outline View (as shown in *Figure1-2*)**

The *Outline* view displays an outline of a structured C/C++ file that is currently open in the editor area, by listing the structural elements. For detailed information, please refer to the help content by clicking **Help > Help Contents > Eclipse Platform Guide > C/C++ Development User Guide > Concepts > Navigation aids > Outline View.**

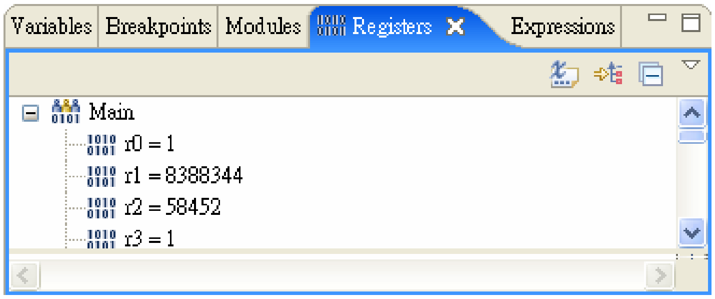
- **Debug View**



```
Hello [Cross Platform Application]
    Remote Cross Platform Debugger (10/3/06 10:58 AM) (Suspended)
        Thread [0] (Suspended: Breakpoint hit.)
            2 main() at .\hello.c:4 0x00000260
            1 _start() 0x00000180
    Debugger Process (10/3/06 10:58 AM)
    C:\Program Files\Andestech\AndeSight\ide\workspace\Hello\Debug\Hello.adx (10/3/06 10:58 AM)
```

The *Debug* view let you control program execution, such as *Run, Pause, Terminate, Step into,* and *Step over* etc. For detailed information, please refer to the help content by clicking **Help > Help Contents > Eclipse Platform Guide > C/C++ Development User Guide > Reference > C/C++ Views and Editors > Debug View > Debug View.**

- **Registers View**

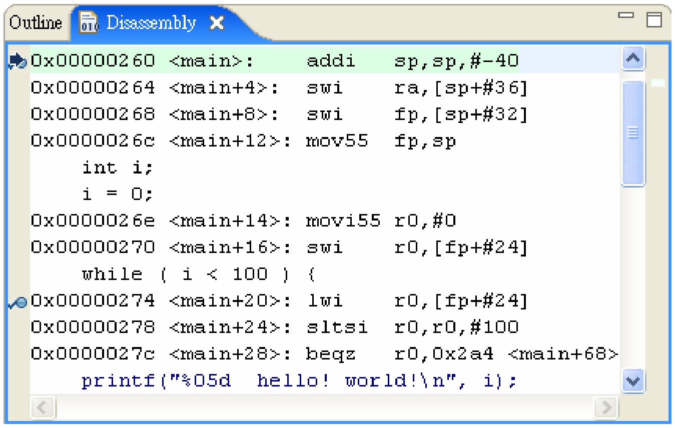


```
Main
    r0 = 1
    r1 = 8388344
    r2 = 58452
    r3 = 1
```

The *Registers* view of the *Debug* perspective lists information about the registers in a selected stack frame. Values that have changed are highlighted in the *Registers* view when your program stops. For detailed information, please refer to the help content by clicking **Help > Help Contents > Eclipse Platform Guide > C/C++ Development User Guide >**

Reference > C/C++ Views and Editors > Debug View > Registers View.

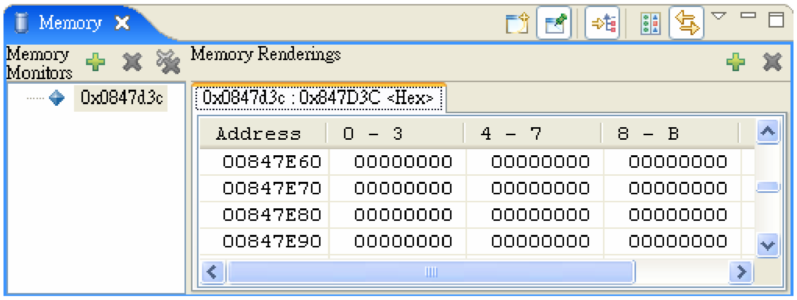
- **Disassembly View**



```
0x00000260 <main>:     addi   sp,sp,#-40
0x00000264 <main+4>:   swi    ra,[sp+#36]
0x00000268 <main+8>:   swi    fp,[sp+#32]
0x0000026c <main+12>:  mov55  fp,sp
    int i;
    i = 0;
0x0000026e <main+14>:  movi55 r0,#0
0x00000270 <main+16>:  swi    r0,[fp+#24]
    while ( i < 100 ) {
0x00000274 <main+20>:  lwi    r0,[fp+#24]
0x00000278 <main+24>:  sltsi  r0,r0,#100
0x0000027c <main+28>:  beqz   r0,0x2a4 <main+68>
    printf("%05d  hello! world!\n", i);
```

Disassembly view lets you examine your program as it steps into disassembled code. This is useful when the instruction pointer enters a function for which it does not have the source. For detailed information, please refer to the help content by clicking **Help > Help Contents > Eclipse Platform Guide > C/C++ Development User Guide > Tasks > Running and debugging projects > Debugging > Stepping into assembler functions.**

- **Memory View**

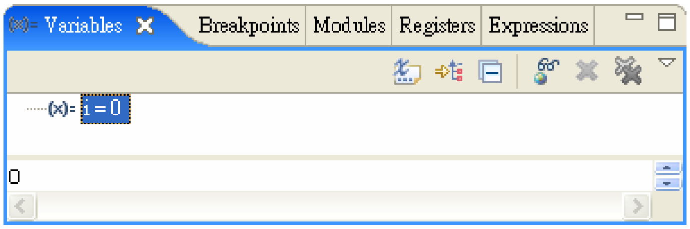


```
Address    0 - 3     4 - 7     8 - B
00847E60   00000000  00000000  00000000
00847E70   00000000  00000000  00000000
00847E80   00000000  00000000  00000000
00847E90   00000000  00000000  00000000
```

Memory view lets you examine the content of your program's memory space. *Memory* view contains two panes - *Memory Monitors* and *Memory Renderings* pane. *Memory Monitors* pane displays the list of memory monitors added currently. *Memory Renderings* pane is controlled by the selection in the *Memory Monitors* pane and consists of the tabs that display renderings. For detailed information, please refer to the help content by clicking **Help > Help**

Contents > Eclipse Platform Guide > C/C++ Development User Guide > Reference > C/C++ Views and Editors > Debug View > Memory View.

- **Variables View**

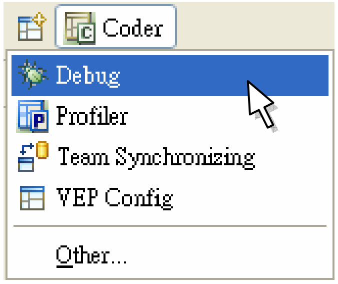


Variables view displays variable types and let user change or disable variable values. For detailed information, please refer to the help content by clicking **Help > Help Contents > Eclipse Platform Guide > C/C++ Development User Guide > Tasks > Running and debugging projects > Debugging > Working with variables.**

1.2.3 Perspectives

A *Perspective* is the initial set and layout of views in the *Window*. Each perspective provides a set of functionality aimed at accomplishing a specific type of task or works, for example creating projects, browsing files, and editing and building source code. A single window can maintain several perspectives, but show only one perspective at a time.

To open new perspectives and switch between them, use the icons in the shortcut bar along the top right edge of the window. For example, if you want to switch the *Coder* perspective to *Debug* perspective, click **Window > Open Perspective > Debug** or Select **Debug** on the perspective shortcut bar.



Then the *Debug* perspective appears, containing the *Debug*, *Breakpoints*, *Local Variables*, and C*onsole* views and so forth, which are useful when running and debugging a program. These views replace the *Outline* view of the *C/C++ coder* perspective.

The Andes IDE contributes the following perspectives to the workbench:

- **Andes Coder perspective**

This perspective is tuned for working with C/C++ projects. By default it consists of an editor area and the following views:

- Project explore (the file navigator for C/C++ resources)
- Navigator (the file navigator for all Eclipse resources)
- Console
- Properties
- Problems
- Outline

- **Debug perspective**

This perspective is tuned for debugging your C/C++ program. By default it includes a *Debug* area and the following views:

- Variables
- Breakpoints
- Expressions
- Registers
- Memory
- Outline
- Console
- Tasks

- **VEP Config (AndESLive) perspective**

This perspective is tuned for working with AndESLive projects. (See section 1.4 for description of AndESLive). By default it consists of a VEP builder area and the following views:

- VEP Project explore
- Properties
- Outline
- Memory Mapping
- IRQ Map

- VEP Manager

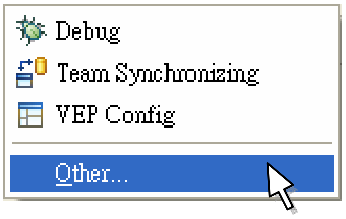
Profiler perspective

This perspective is tuned for profiling your C/C++ program. By default it includes the following views:

- Debug
- Project explore
- Console
- Tasks
- Variables
- Breakpoints
- C/C++ editor

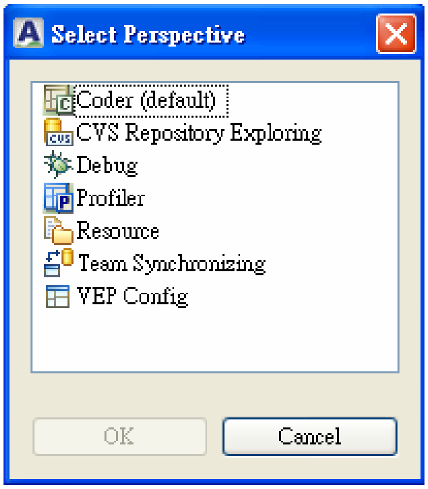
Other Perspectives

In addition to the perspectives named above, Andes also has perspectives that are tuned to other types of development. To open other perspectives, click the icon in the shortcut bar, and then select **Other…**.



The followings are other perspectives we provide:

- CVS Repository Exploring
- Resource
- Team Synchronizing



Select Perspective

Coder (default)
CVS Repository Exploring
Debug
Profiler
Resource
Team Synchronizing
VEP Config

You can also create your own perspectives to suit your development needs. Follow the steps below to create a customized perspective:

1. Arrange the views in the window as desired by opening any required views and moving them to an appropriate location.
2. Select **Window > Save Perspective As**, enter a name for your custom perspective.
3. Click **OK**.

Note: 1. The customized perspective is saved in the workspace.

2. Switch back to the C/C++ coder perspective by clicking its icon on the top right side and resetting it with **Window > Reset Perspective**.

1.2.4 Editors

Editor is a special type of view used to edit files. You can associate different Editors with different types of files such as C, C++, XML, Assembler, and Make files. The associated Editor is invoked in the perspective's *Editor* area while you open a file.

Any number of Editors can be open at once, but only one can be active at a time. By default, *Editors* are stacked in the Editor area, but you can tile them in order to view source files simultaneously. Tabs in the Editor area indicate the names of files that are currently open for editing. An asterisk (*) indicates that an Editor contains unsaved changes.

C/C++ editor is provided with complete functionalities helping you edit your source code, such as *Content Assist*, which is a set of tools built to help reduce the number of keystrokes you must type. *Content Assist* consists of several components that forecast what a developer will type, based on the current context, scope, and prefix. It provides code completion and code templates features which triggered by typing **Ctrl** + **Space**. For more detailed information, please refer to the help content by clicking **Help > Help Contents > Eclipse Platform Guide > C/C++ Development User Guide > Concepts > Code aids > Content Assistant.**

1.2.5 Plug-ins

The *Eclipse* platform is structured as a core runtime engine and a set of additional features that are installed as platform *Plug-ins*. When you start up the workbench, you are not starting up a single program. You are activating a platform runtime which can dynamically discover registered plug-ins and start them as needed. Through adding the plug-ins, AndeSight contributes complete functionality to the platform letting users develop their program more convenient and easily.

1.3 Toolchains

GNU based toolchains:

- Compiler
- Assembler
- Linker
- GDB Debugger
- GDB agent

Toolchains utilities will be invoked via IDE interface to build your target software from the project.

1.4 AndESLive Overview

AndESLive consisting of SoC simulation engine and SoC Builder provides the models of Andes CPU and peripheral devices.

With the AndESLive, Virtual Evaluation Platform, you can go from simulation to prototyping for SoC with minimal or no time spent characterizing physical evaluation boards. It provides the flexibility for you to evaluate SoC configuration and performance to meet the requirements of your whole system.

Here is the pre-configured SoC AG101 that we provide as an example in AndESLive.

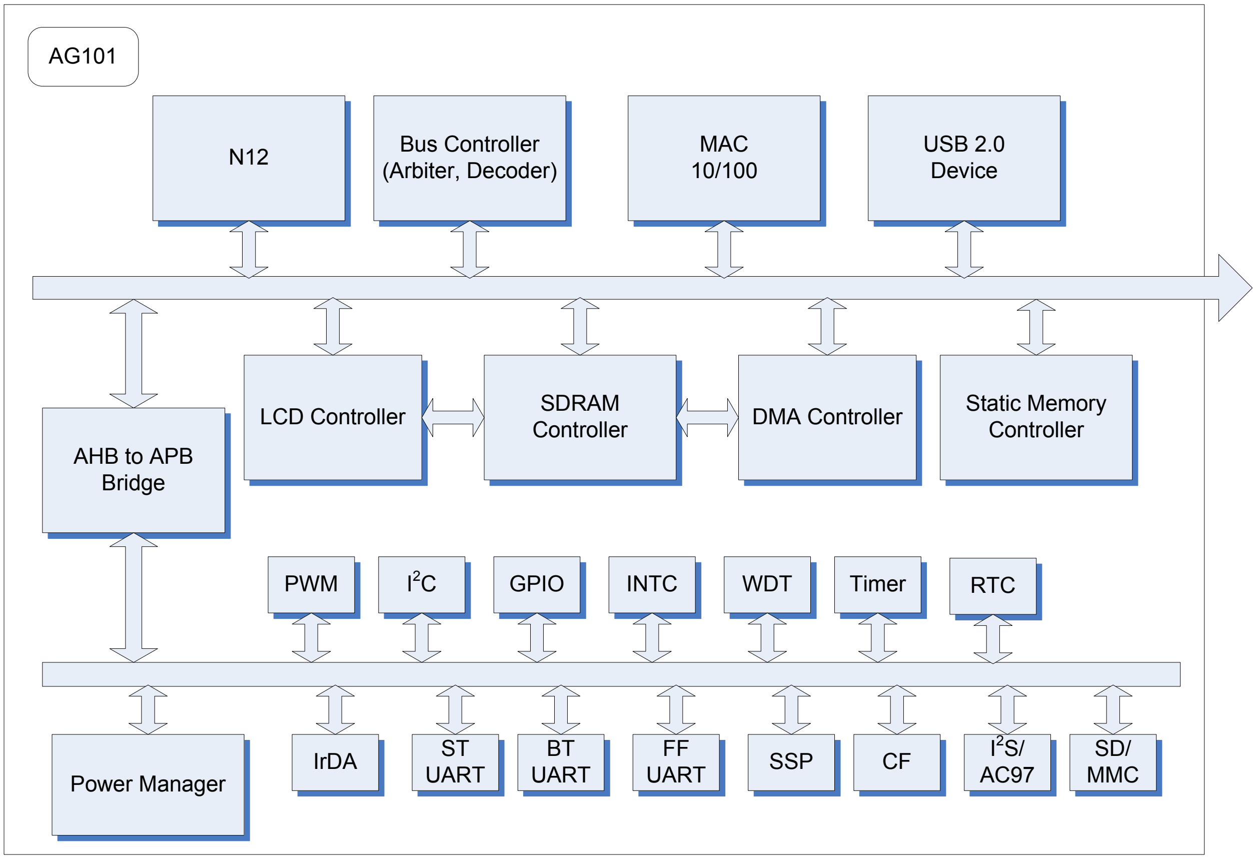


Figure 1-3: Block Diagram of AG101 VEP

The following lists all components of AG101:

- CPU / MMU / CACHE
- Bus controller
- MAC 10/100
- USB 2.0
- LCD controller
- SDRAM controller
- DMA controller
- Static memory controller
- AHB to APB bridge
- PWM
- I^2C
- GPIO
- INTC
- WDT
- Timer
- RTC
- Power manager
- IrDA
- ST UART
- BT UART
- FF UART
- CF
- SSP / I2S / AC97
- SD / MMC

Note: For detailed information, please refer to AndESLive reference.

Chapter 2

Installation Guide

This chapter provides a quick overview of how to set up AndeSight. It contains the following sections:

2.1 Contents of Installation package

The whole installation package contains:

- AndeSight
  - IDE
  - Toolchains
- AndESLive
  - Virtual Evaluation Platform
  - Components models
- Documentations
- Cygwin
- IBM JVM (Java Virtual Machine)

2.2 Optimum system requirements

To install and run AndeSight, you should have the following:

- Windows XP operating system
- 1 Gigabytes of RAM for minimum requirement
- 1 Gigabytes of free disk space for installation

2.3 Installation procedure

The InstallShield wizard will guide you into the set-up process. The following steps will be brought out.

Step 1 Double click AndeSight_Install.exe file

Welcome to the installation of AndeSight

Step 2 Click **Next**

Step 3 Enter the User and Company names, and then click **Next**

Step 4 Choose **Complete** to install all the features of AndeSight or **Custom** to choose the features, and then click **Next**

Step 5 Click **Change** to change the destination directory or use the default directory, and then click **Next**

Step 6 Choose what features you want to install (For **Complete** installation, skip this step)

Step 7 Click **Install** to begin the installation

Step 8 Installation complete and click **Finish** to exit the wizard
Installation completes.

Note: Default option is to install the whole package.

1. **Complete** installation will install the whole package including *Cygwin, IBM JVM*, and *AndeSight*.
2. **Custom** installation will let user choose what component they want to install.
3. *Cygwin* is a Linux-like environment for Windows. Installation of *Cygwin* is the prerequisite of running AndeSight.
4. Linux Platform version will be released in Q1, 2007

Chapter 3

Getting Started

The following provides a tutorial that walks you through all the major features of AndeSight IDE: creating projects, building and debugging code, connecting to VEP, and running your program.

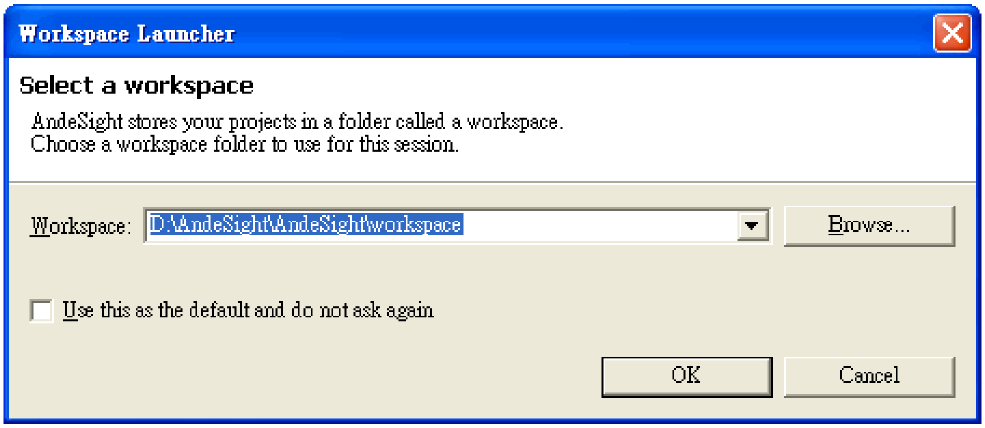
3.1 Create a new project

This section describes in great detail the process of creating, compiling, and running a simple "Hello, World" program. It is best if you actually run AndeSight while reading this section and perform the steps as they are described.

With the installation procedure in *Chapter 2*, you can easily start AndeSight by the following steps:

1 Invoke AndeSight from Desktop's shortcut "AndeSight IDE" or from Windows Start Menu **Programs > Andestech > AndeSight > AndeSight**. *Workspace launcher* dialog will ask you to define a workspace as shown in *Figure 3-1*. You can click the **Browse …** button or type directly in the text field to define the preferred workspace directory. Press **OK** button after done.

Workspace is where a project is saved. Users can create multiple workspaces required by different projects or teams. Furthermore, users can keep the whole workspace and use in next release of AndeSight.



Workspace Launcher

Select a workspace

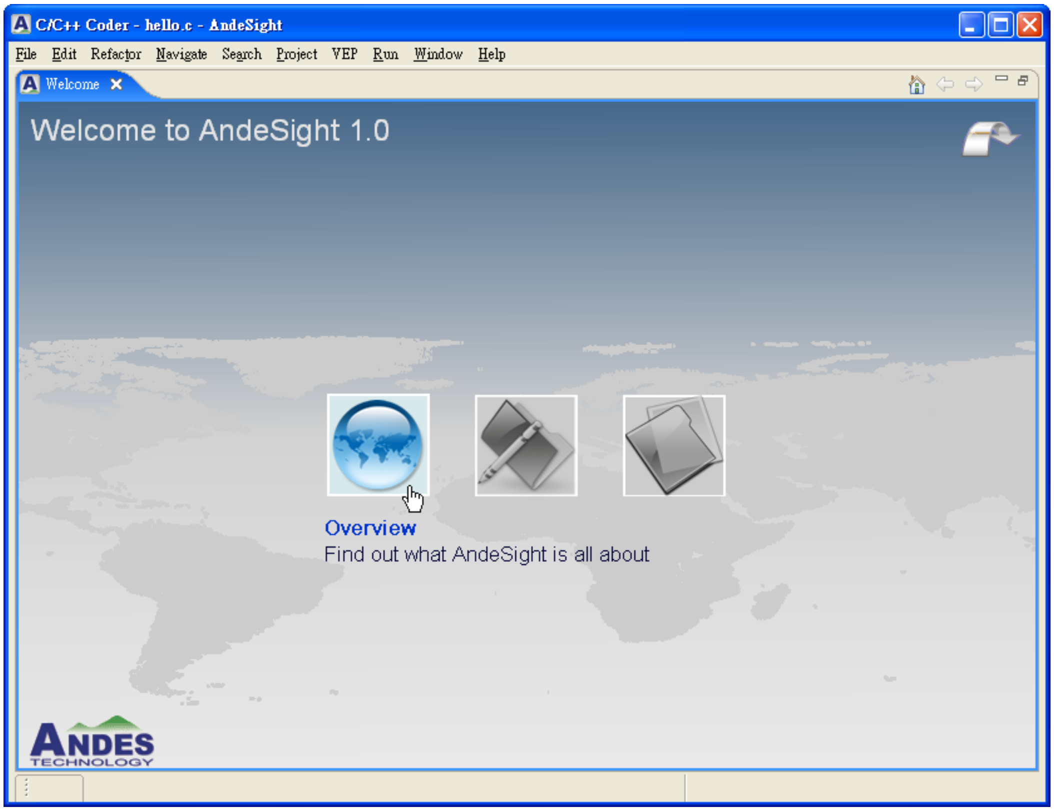
AndeSight stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: D:\AndeSight\AndeSight\workspace

□ Use this as the default and do not ask again

Figure 3-1: Define the preferred workspace directory

2 The present screen should be "Welcome to AndeSight 1.0" page as shown in *Figure 3-2.* This welcome page provides three links for your quick guide into AndeSight IDE, which are *Overview*, *Getting Started*, and *Examples*. (This page could also be invoked after you enter the workbench by clicking **Help > Welcome** in IDE). For now, click the curved arrow on the top of the right hand side to enter the IDE Workbench as shown in *Figure 3-3.*

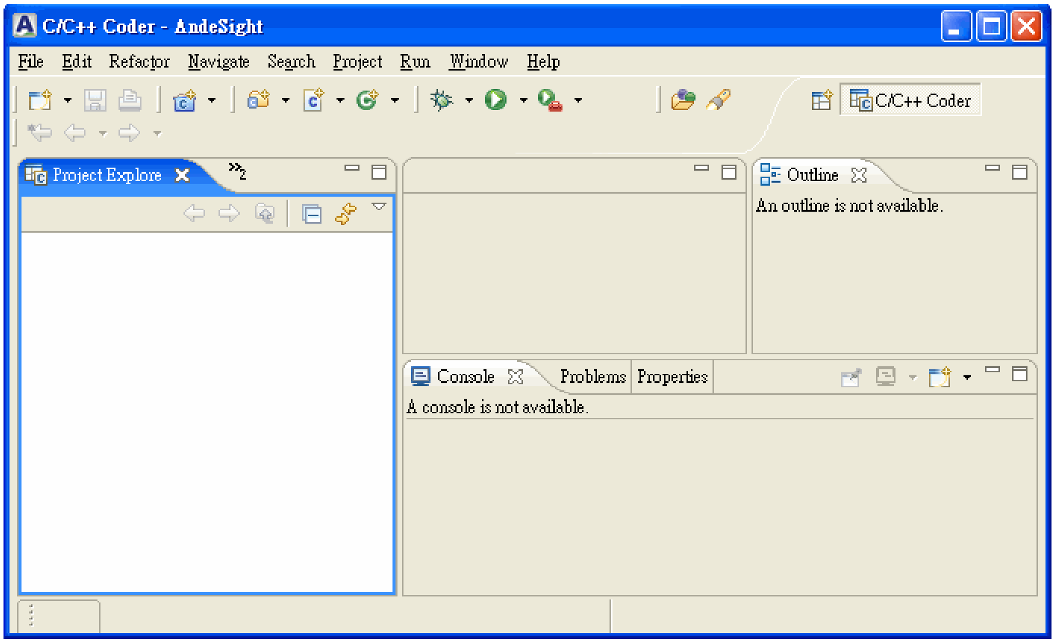

C/C++ Coder - hello.c - AndeSight
File Edit Refactor Navigate Search Project VEP Run Window Help
Welcome
ANDES TECHNOLOGY

Welcome to AndeSight 1.0

Overview

Find out what AndeSight is all about

Figure 3-2: Welcome page



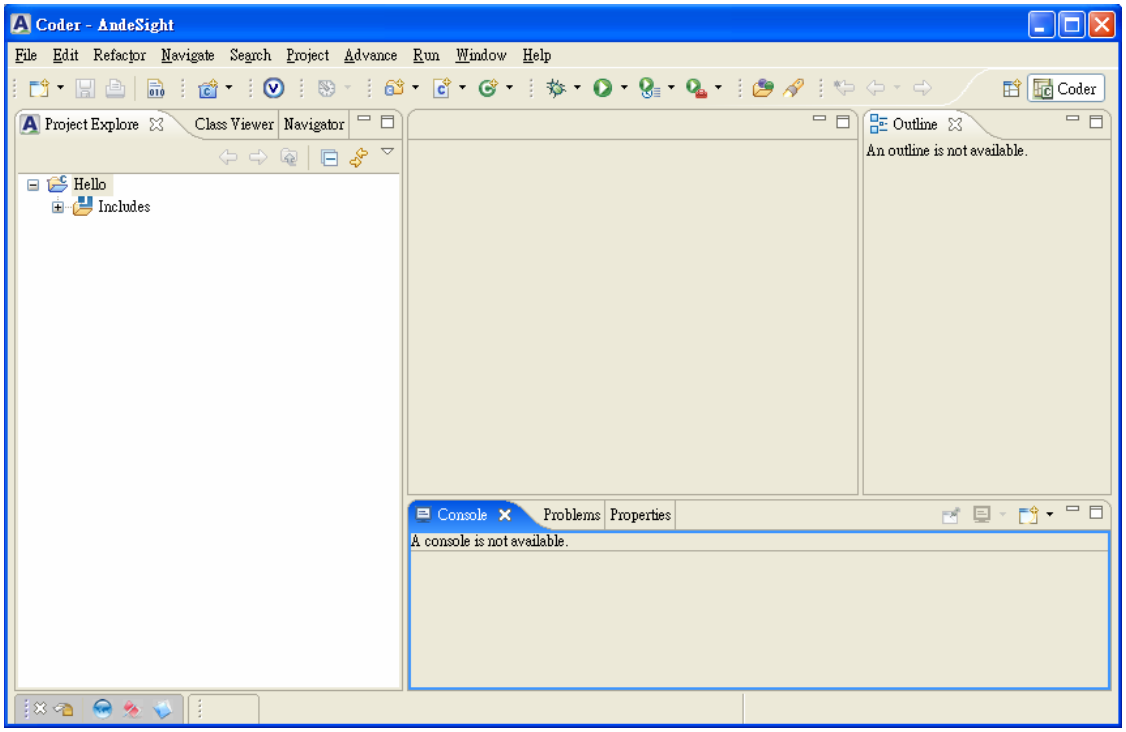
An outline is not available.

A console is not available.

Figure 3-3: AndeSight Workbench

3 The Welcome page only presents at the first time starting AndeSight. Afterward, you will always get the views that you previous work on. To create a new project, you can easily click **File > New > Project…** and the Project wizard will guide you through the whole process.

- Select **File > New > Project …**, a new project wizard will popup to guide you to create a project.
- Expand **Andes Project** by clicking the "**+**" icon, and then select either C or C++ project based on your application program. For this tutorial, please select **C project** and press **Next**.
- Please input **Hello** as the project name in the next dialog and then click **Finish**.
- The *Hello* project will be created and shown in the Project Explore as the *Figure 3-4*.



An outline is not available.

A console is not available.

Figure 3-4: Create a demo Project

3.2 Editing

The next step for the project is to create a C source code file and edit it, but in this tutorial, AndeSight provides a pre-prepared example for you to add in the new project.

- Select the *Hello* project and click the **File > Import…**, then the *Import* dialog will be popup.
- Select the **File System** and click **Next** (or double click the **File System**).
- Click **Browse** and navigate the directory structure to <AndeSight-installed-path>\hello, and then click **OK.**
- In the left window of the *Import* dialog, click the *hello* folder. The files inside the *hello* folder will be shown in the right window.
- Check the checkbox of the **hello.c**, and then click **Finish.**

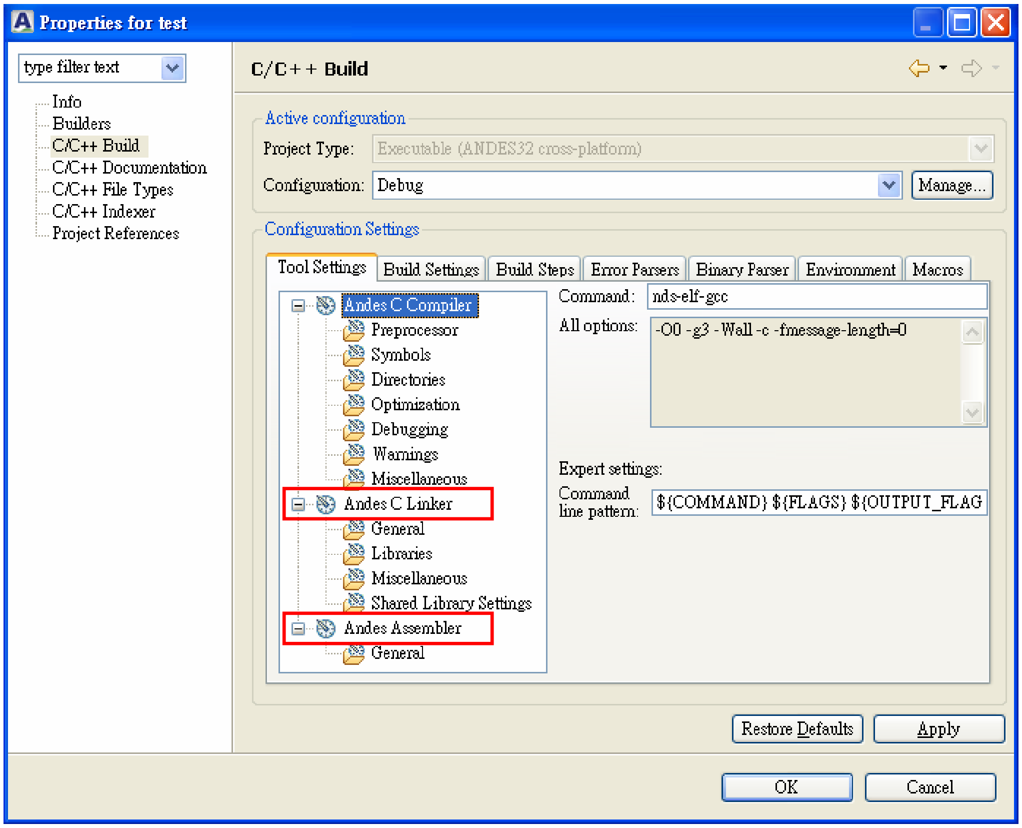
The *Project Explore* now shows that hello.c has been added to the project. Source file in the *Project Explore* can be edited by double clicking on its icon.

Note: 1. There are many convenient editing tools provided, such as Search, Refactoring, Syntax coloring, and Auto-completion.

2. There is no executable image existing for the project yet.

3.3 Build

Select the project *Hello* you just created, right click to invoke the **Build Project** command. Builder will call Andes' backend toolchains to create the make file and compile, link and at last generate the executable file. The compiler and linker have several options which could be configured by right-clicking the project selecting **Properties**.



Project Type: Executable (ANDES32 cross-platform)

Configuration: Debug

Command: nds-elf-gcc

All options: -O0 -g3 -Wall -c -fmessage-length=0

Command line pattern: ${COMMAND} ${FLAGS} ${OUTPUT_FLAG

Figure 3-5: Build Properties

3.4 Start GDBAgent and VEP

Before running your program/software, you need to set up the backend simulation environment. For your convenience, Andes provides you GDBAgent to complete this task.

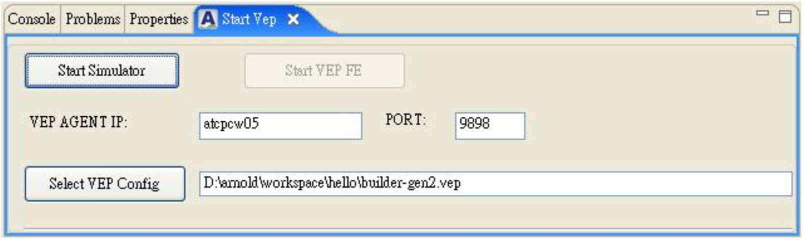
1 from Windows Start Menu **Programs > Andestech > AndeSight > Cygwin,** GDBAgent is started in Cygwin prompt by

```
%> GDBAgent –v none
```

GDBAgent is to coordinate and handle all the detail communications between/among front-end IDE and back-end VEP and ICE. The GDBAgent utilizes the socket IO to communicate between AndeSight components, so that the front-end IDE could be run on such as local desktop PC, and VEP could be run on Linux server; in this way users would get benefit in both convenience and efficiency.

Note: You can also start the GDBAgent in a shell on Linux server by this same command as above.

2 Start *Virtual Evaluation Platform* (*VEP*) by selecting **Advance > Start VEP** from menu bar or clicking **Start VEP** icon A from toolbar. The following *VEP Manager* shows the control GUI of *VEP*.



VEP AGENT IP: atcpcw05

PORT: 9898

D:\arnold\workspace\hello\builder-gen2.vep

Figure 3-6: VEP Manager

3 Drag the general.vep file from Project Explore and drop to the space next to **Select VEP Config**.

4 Press **Start Simulator** to start simulator.

5 AndeSight IDE will pass the vep file to VEP through GDBAgent. The SoC configured in the vep file will be run as the simulation target.

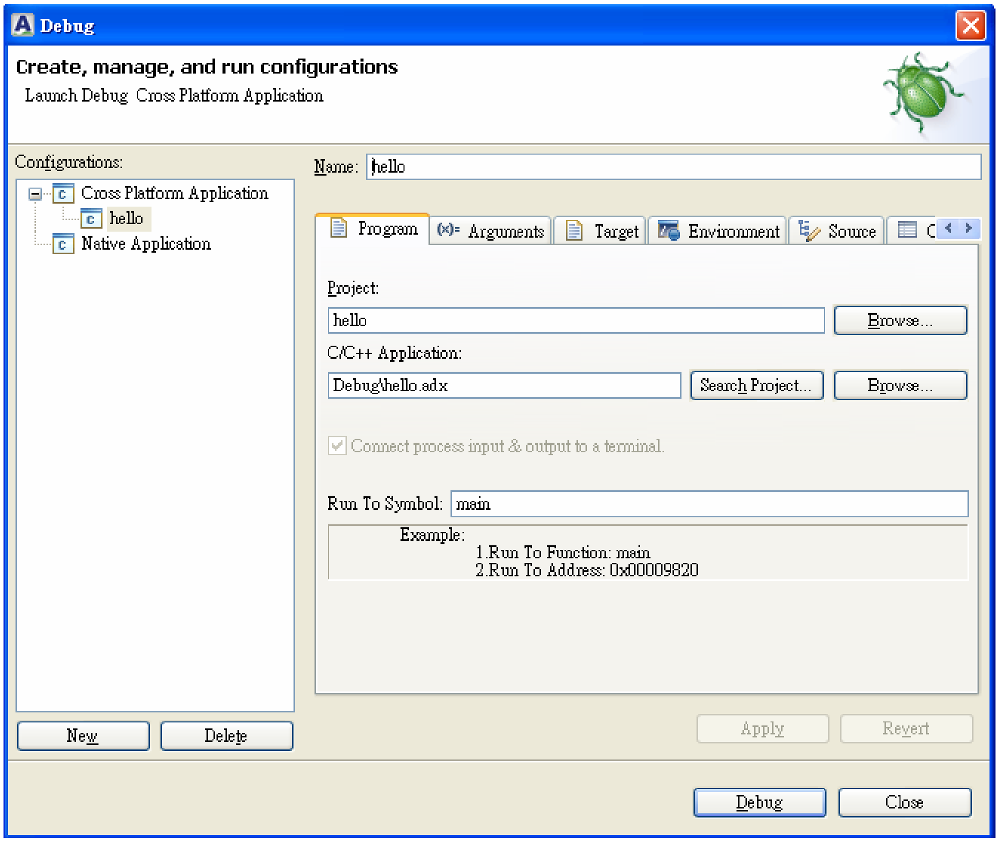
Note: 1. Cygwin is a Linux-like environment for Windows

2. The total duration of VEP startup and connection will be around 15 seconds.

3.5 Target management

To run and debug the program, you need to specify the target before executing the program by the following steps:

1 Select **Run > Debug…** from menu bar, a *Debug* dialog is popup to let you set the VEP configuration, including the simulator of AndeScore and models of peripheral IPs. Select **Cross Platform Application** and press **New**. A *demo* configuration will be created under **Cross Platform Application.**



Debug

Create, manage, and run configurations

Launch Debug Cross Platform Application

Configurations:

- Cross Platform Application
  - hello
- Native Application

Name: hello

Project: hello

C/C++ Application: Debug\hello.adx

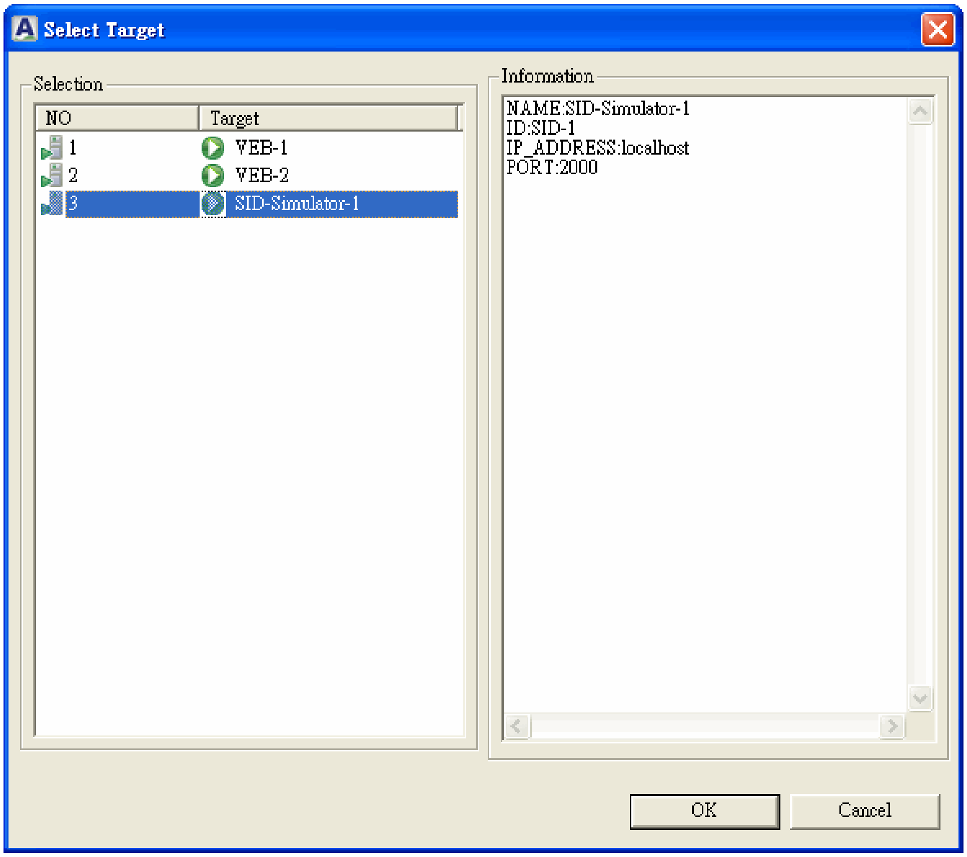
☑ Connect process input & output to a terminal.

Run To Symbol: main

Example:
1.Run To Function: main
2.Run To Address: 0x00009820

Figure 3-7: Target Configuration to run the executable

2 Click the third tab **Target** on the right pane and click **Auto Select** button for selecting a VEP. You can select evaluation board VEP-1, VEP-2, or VEP SID-Simulator-1 from the **Select Target** dialog. Select **SID-Simulator-1** for VEP and click **OK** to finish.



NO	Target
1	VEB-1
2	VEB-2
3	SID-Simulator-1

```
NAME:SID-Simulator-1
ID:SID-1
IP_ADDRESS:localhost
PORT:2000
```

Figure 3-8: Select target

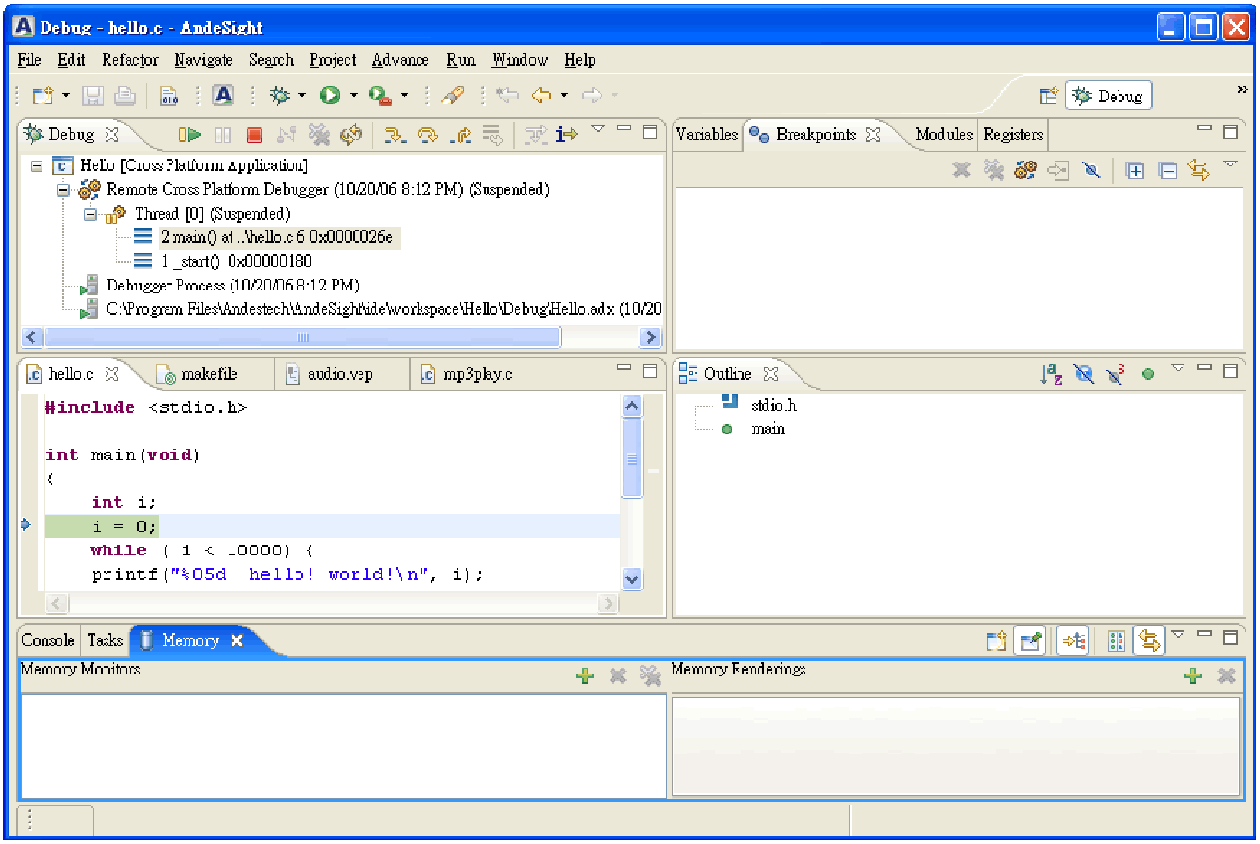
3 The Debug configuration dialog will be appearing and then click **Debug** button to run the selected project on the target.

4 AndeSight will automatically switch to *Debug* Perspective which is shown as *Figure 3-9*. *Debug* will be discussed in the *section 3.6*.

3.6 Debugging

AndeSight provides integrated and advantageous views for debugging in the *Debug* perspective.

A software debugging could be a series of complicated tasks such as error detections, source of bugs locating, and validation of the program etc. *Figure 3-9* shows the views of *Debug* perspective provided by AndeSight.



```
Hello [Cross Platform Application]
    Remote Cross Platform Debugger (10/20/06 8:12 PM) (Suspended)
        Thread [0] (Suspended)
            2 main() at ..\hello.c 6 0x0000026e
            1 _start() 0x00000180
    Debugger Process (10/20/06 8:12 PM)
    C:\Progrem Files\Andestech\AndeSight\ide\workspace\Hello\Debug\Hello.adx (10/20
```

```c
#include <stdio.h>

int main(void)
{
    int i;
    i = 0;
    while ( i < 10000) {
    printf("%05d hello! world!\n", i);
```

Figure 3-9: Debug perspective

1 The program will stop at the *main()* entry and a pointer locates at the line after *main()*. There are two views presenting different information of the source code. The execution could step in different formats, either in higher level C source code or lower level assembler, depending on your debugging needs. In the *Debug* view, icons at the tool bar are **Run**, **Pause**, **Step into**, **Step over**, and **Terminate** etc. The root of the content is the application, and the **Cross Platform Debugger** below the root indicates the target you specify in section *3.5 Target management*.

2 Select *main() at hello.c* in the *Debug* view as shown in *Figure 3-9*, and press **Step over**. The source code view will show a green indicator for current position of execution. In the *Disassembly* view, there will be a corresponding cursor pointing to the highlighted line of assembly code.

3 Memory view provides a very convenient way to inspect the data in memory after loading or storing. The memory cells changing between the steps such as stepping or suspending will be shown with arrow,

4 The result will be shown in the Console view.

Note: 1. All the debug-related views do not refresh while you run your executable. The refreshment occurs when the execution stops.

2. For more information, please refer to the **online Help > Eclipse Platform Guide > C/C++ Development User Guide > Tasks > Running and debugging projects > Debugging**

3.7 Profiling

After programs have been debugged, the next task of programmers will be to tune the software and hardware in order to get the better performance. Andes provides *Profiling* function letting users observe and examine their programs and hardware more easily.
The following describes the profiling steps.

1 To start the profiling, please switch the perspective from **Debugger** to **Profiler** first.

2 As you can see, project explore will be on your left hand side. Right click the project that was being debugged, and then select **Profile As > Profile…**.

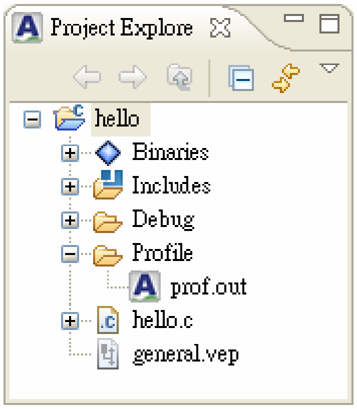
3 Profiling dialog will show and inherit the settings you just set in the debugging dialog.

4 Click **Profile.**

5 Trigger the **Enable Source Profiling** icon Ⓟ to start the profiling.

6 Click **Resume** to run the program till finished, and then click Ⓟ icon again to terminate profiling.

7 Now refresh the project explore, you will find *prof.out* generated in the **profile** folder.

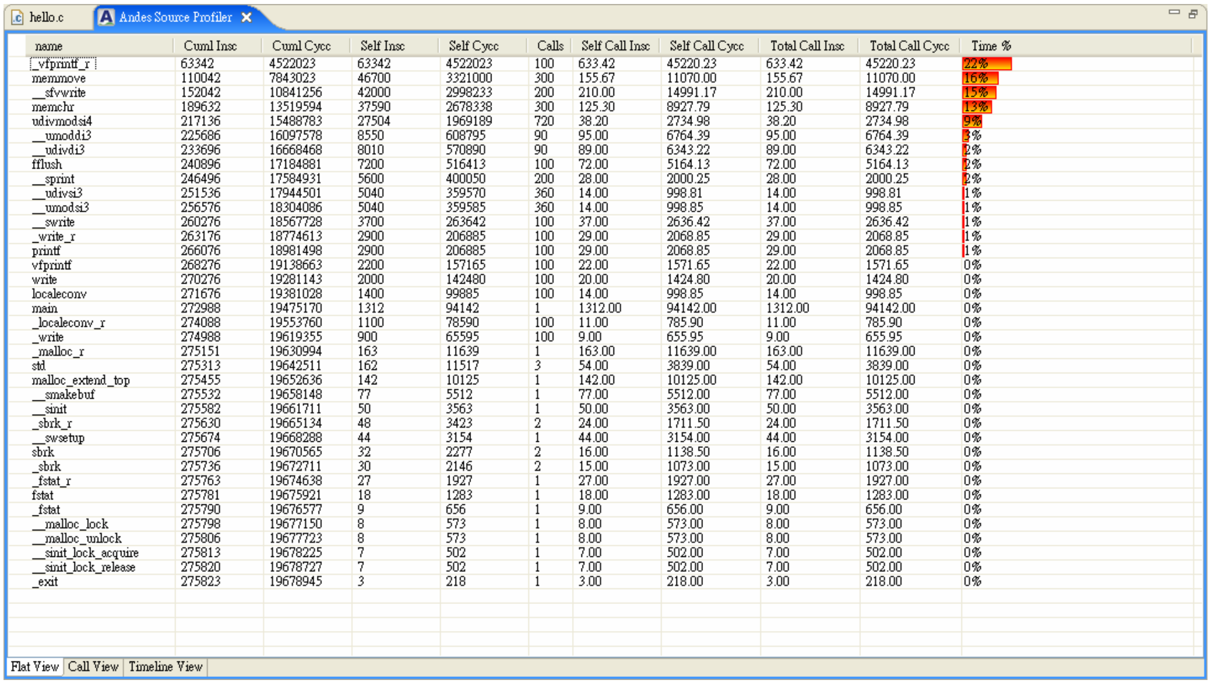


8 Double click *prof.out* or click to show profiling results. In order to present detailed and clear profiling, we provide three views, which are timeline, call, and flat view. The followings will demonstrate and explain these views and their features.

Note: You can also profile between two breakpoints through triggering the profiling at the first breakpoint and terminating at another breakpoint.

Flat View

The *flat view* shows the total amount of time your program spent executing each function. You can sort the data by different column.



name	Cuml Insc	Cuml Cycc	Self Insc	Self Cycc	Calls	Self Call Insc	Self Call Cycc	Total Call Insc	Total Call Cycc	Time %
_vfprintf_r	63342	4522023	63342	4522023	100	633.42	45220.23	633.42	45220.23	22%
memmove	110042	7843023	46700	3321000	300	155.67	11070.00	155.67	11070.00	16%
__sfvwrite	152042	10841256	42000	2998233	200	210.00	14991.17	210.00	14991.17	15%
memchr	189632	13519594	37590	2678338	300	125.30	8927.79	125.30	8927.79	13%
udivmodsi4	217136	15488783	27504	1969189	720	38.20	2734.98	38.20	2734.98	9%
__umoddi3	225686	16097578	8550	608795	90	95.00	6764.39	95.00	6764.39	3%
__udivdi3	233696	16668468	8010	570890	90	89.00	6343.22	89.00	6343.22	2%
fflush	240896	17184881	7200	516413	100	72.00	5164.13	72.00	5164.13	2%
__sprint	246496	17584931	5600	400050	200	28.00	2000.25	28.00	2000.25	2%
__udivsi3	251536	17944501	5040	359570	360	14.00	998.81	14.00	998.81	1%
__umodsi3	256576	18304086	5040	359585	360	14.00	998.85	14.00	998.85	1%
__swrite	260276	18567728	3700	263642	100	37.00	2636.42	37.00	2636.42	1%
_write_r	263176	18774613	2900	206885	100	29.00	2068.85	29.00	2068.85	1%
printf	266076	18981498	2900	206885	100	29.00	2068.85	29.00	2068.85	1%
vfprintf	268276	19138663	2200	157165	100	22.00	1571.65	22.00	1571.65	0%
write	270276	19281143	2000	142480	100	20.00	1424.80	20.00	1424.80	0%
localeconv	271676	19381028	1400	99885	100	14.00	998.85	14.00	998.85	0%
main	272988	19475170	1312	94142	1	1312.00	94142.00	1312.00	94142.00	0%
_localeconv_r	274088	19553760	1100	78590	100	11.00	785.90	11.00	785.90	0%
_write	274988	19619355	900	65595	100	9.00	655.95	9.00	655.95	0%
_malloc_r	275151	19630994	163	11639	1	163.00	11639.00	163.00	11639.00	0%
std	275313	19642511	162	11517	3	54.00	3839.00	54.00	3839.00	0%
malloc_extend_top	275455	19652636	142	10125	1	142.00	10125.00	142.00	10125.00	0%
__smakebuf	275532	19658148	77	5512	1	77.00	5512.00	77.00	5512.00	0%
__sinit	275582	19661711	50	3563	1	50.00	3563.00	50.00	3563.00	0%
_sbrk_r	275630	19665134	48	3423	2	24.00	1711.50	24.00	1711.50	0%
__swsetup	275674	19668288	44	3154	1	44.00	3154.00	44.00	3154.00	0%
sbrk	275706	19670565	32	2277	2	16.00	1138.50	16.00	1138.50	0%
_sbrk	275736	19672711	30	2146	2	15.00	1073.00	15.00	1073.00	0%
_fstat_r	275763	19674638	27	1927	1	27.00	1927.00	27.00	1927.00	0%
fstat	275781	19675921	18	1283	1	18.00	1283.00	18.00	1283.00	0%
_fstat	275790	19676577	9	656	1	9.00	656.00	9.00	656.00	0%
__malloc_lock	275798	19677150	8	573	1	8.00	573.00	8.00	573.00	0%
__malloc_unlock	275806	19677723	8	573	1	8.00	573.00	8.00	573.00	0%
__sinit_lock_acquire	275813	19678225	7	502	1	7.00	502.00	7.00	502.00	0%
__sinit_lock_release	275820	19678727	7	502	1	7.00	502.00	7.00	502.00	0%
_exit	275823	19678945	3	218	1	3.00	218.00	3.00	218.00	0%

Here is what the fields in each line mean:

- Time %

  This is the percentage of the total execution time your program spent in this function. These should all add up to 100%.

- Cumulative Instruction and Cycle count

  This is the cumulative total number of instructions and cycles the simulator spent executing the functions, plus the time spent in all the functions above this one in this table.

- Self Instruction and Cycle count

  This is the number of instruction and cycle accounted for by this function alone. The flat profile listing is sorted first by this number.

- Calls

  This is the total number of times the function was called. If the function was never called, or the number of times it was called cannot be determined (probably because the function was not compiled with profiling enabled), the calls field is blank.

- Self Instruction/Cycle per call

  This represents the average number of instructions and cycle spent in this function per call, if this function is profiled. Otherwise, this field is blank for this function.

- Total Call Instruction/Cycle count

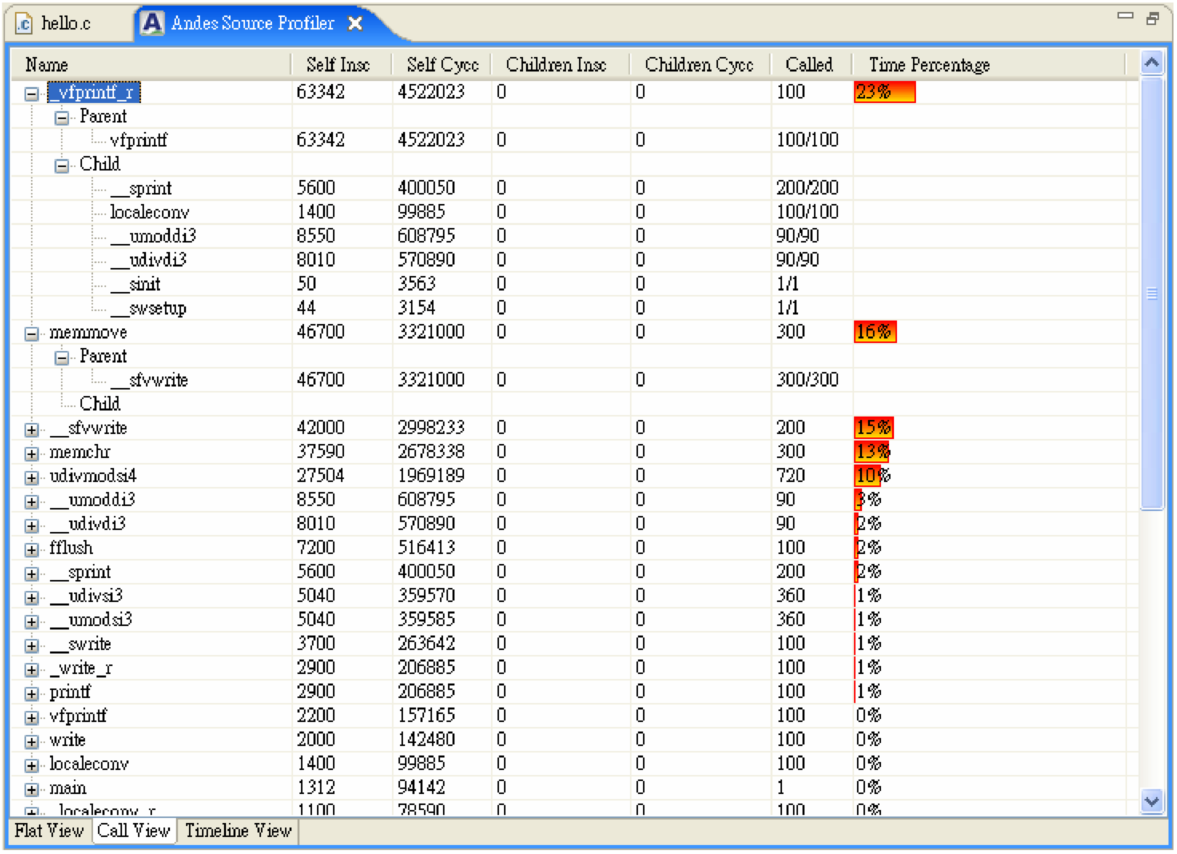
  This represents the average number of instructions and cycles spent in this function and its descendants per call, if this function is profiled. Otherwise, this field is blank for this function. This is the only field in the flat profile that uses call graph analysis.

- Name

  This is the name of the function. The flat profile is sorted by this field alphabetically after the *self seconds and calls* fields are sorted.

Call view

The *call view* shows how much time was spent in each function and its children. From this information, you can find functions that, while they themselves may not have used much time, called other functions that did use unusual amounts of time.



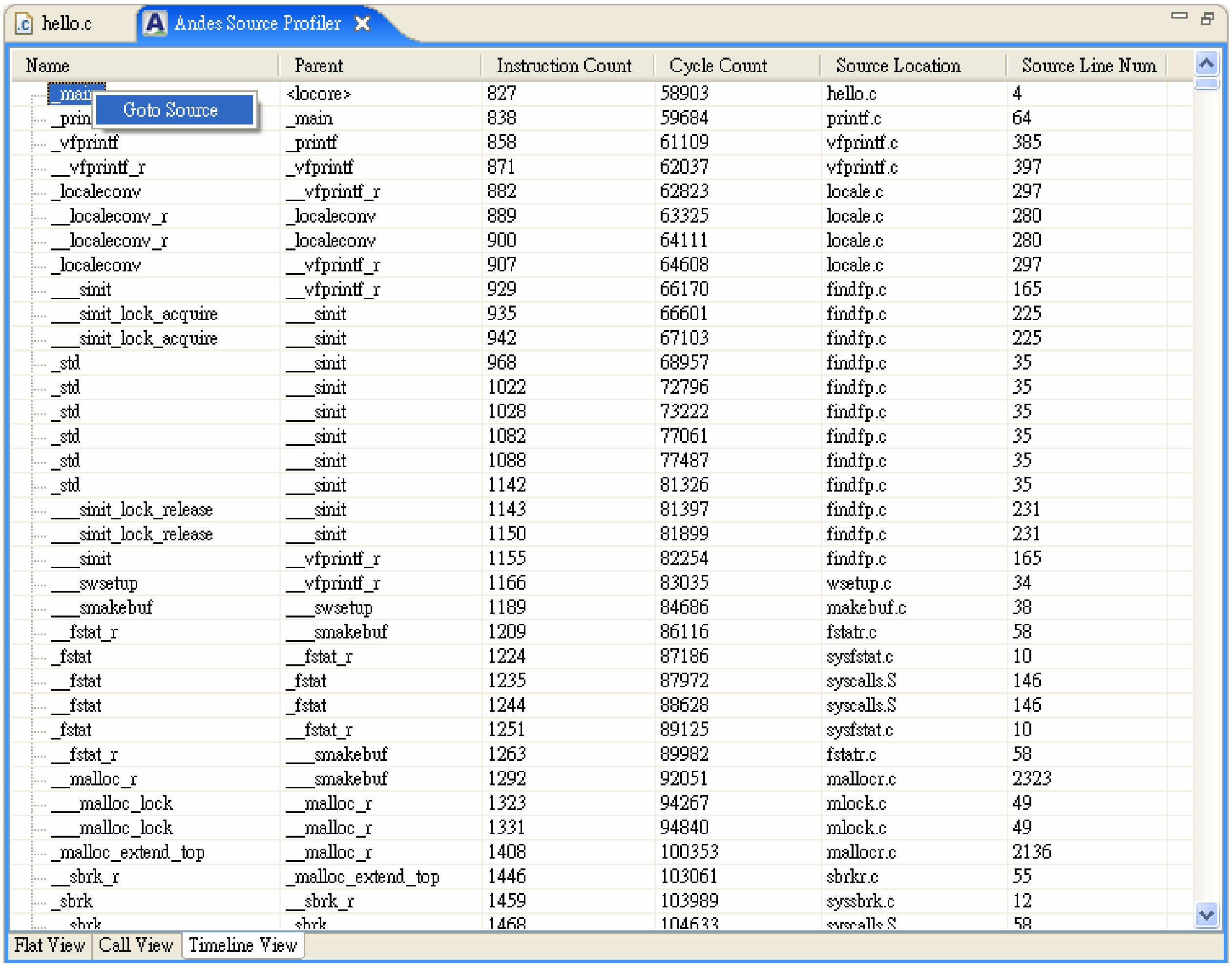
Name	Self Insc	Self Cycc	Children Insc	Children Cycc	Called	Time Percentage
⊟ _vfprintf_r	63342	4522023	0	0	100	23%
⊟ Parent						
vfprintf	63342	4522023	0	0	100/100	
⊟ Child						
__sprint	5600	400050	0	0	200/200	
localeconv	1400	99885	0	0	100/100	
__umoddi3	8550	608795	0	0	90/90	
__udivdi3	8010	570890	0	0	90/90	
__sinit	50	3563	0	0	1/1	
__swsetup	44	3154	0	0	1/1	
⊟ memmove	46700	3321000	0	0	300	16%
⊟ Parent						
__sfvwrite	46700	3321000	0	0	300/300	
Child						
⊞ __sfvwrite	42000	2998233	0	0	200	15%
⊞ memchr	37590	2678338	0	0	300	13%
⊞ udivmodsi4	27504	1969189	0	0	720	10%
⊞ __umoddi3	8550	608795	0	0	90	3%
⊞ __udivdi3	8010	570890	0	0	90	2%
⊞ fflush	7200	516413	0	0	100	2%
⊞ __sprint	5600	400050	0	0	200	2%
⊞ __udivsi3	5040	359570	0	0	360	1%
⊞ __umodsi3	5040	359585	0	0	360	1%
⊞ __swrite	3700	263642	0	0	100	1%
⊞ _write_r	2900	206885	0	0	100	1%
⊞ printf	2900	206885	0	0	100	1%
⊞ vfprintf	2200	157165	0	0	100	0%
⊞ write	2000	142480	0	0	100	0%
⊞ localeconv	1400	99885	0	0	100	0%
⊞ main	1312	94142	0	0	1	0%
⊞ _localeconv_r	1100	78590	0	0	100	0%

Parent lists the callers and **Child** lists the callee of each entry.

Timeline view

The timeline view allows you to track function calls as they occur over a period of time. This is useful for viewing function call sequence. Even though the program is recursive or has repeated loops, the function calls will be listed separately.

Once finding any query and needing modifications of the source, you can right click the function, and then **Go to Source** will popup. Through this, you can easily find and switch to the source position for review.



Name	Parent	Instruction Count	Cycle Count	Source Location	Source Line Num
_mai	<locore>	827	58903	hello.c	4
_prin	_main	838	59684	printf.c	64
_vfprintf	_printf	858	61109	vfprintf.c	385
__vfprintf_r	_vfprintf	871	62037	vfprintf.c	397
_localeconv	__vfprintf_r	882	62823	locale.c	297
__localeconv_r	_localeconv	889	63325	locale.c	280
__localeconv_r	_localeconv	900	64111	locale.c	280
_localeconv	__vfprintf_r	907	64608	locale.c	297
___sinit	__vfprintf_r	929	66170	findfp.c	165
___sinit_lock_acquire	___sinit	935	66601	findfp.c	225
___sinit_lock_acquire	___sinit	942	67103	findfp.c	225
_std	___sinit	968	68957	findfp.c	35
_std	___sinit	1022	72796	findfp.c	35
_std	___sinit	1028	73222	findfp.c	35
_std	___sinit	1082	77061	findfp.c	35
_std	___sinit	1088	77487	findfp.c	35
_std	___sinit	1142	81326	findfp.c	35
___sinit_lock_release	___sinit	1143	81397	findfp.c	231
___sinit_lock_release	___sinit	1150	81899	findfp.c	231
___sinit	__vfprintf_r	1155	82254	findfp.c	165
___swsetup	__vfprintf_r	1166	83035	wsetup.c	34
___smakebuf	___swsetup	1189	84686	makebuf.c	38
__fstat_r	___smakebuf	1209	86116	fstatr.c	58
_fstat	__fstat_r	1224	87186	sysfstat.c	10
__fstat	_fstat	1235	87972	syscalls.S	146
__fstat	_fstat	1244	88628	syscalls.S	146
_fstat	__fstat_r	1251	89125	sysfstat.c	10
__fstat_r	___smakebuf	1263	89982	fstatr.c	58
__malloc_r	___smakebuf	1292	92051	mallocr.c	2323
___malloc_lock	__malloc_r	1323	94267	mlock.c	49
___malloc_lock	__malloc_r	1331	94840	mlock.c	49
_malloc_extend_top	__malloc_r	1408	100353	mallocr.c	2136
__sbrk_r	_malloc_extend_top	1446	103061	sbrkr.c	55
_sbrk	__sbrk_r	1459	103989	syssbrk.c	12
__sbrk	_sbrk	1468	104633	syscalls.S	58

Chapter 4

Advanced Usages with project tutorial

This chapter discusses the advanced usages of Andes by introducing the two examples. Users will acquire further understanding and advanced usages of AndeSight through the tutorial.

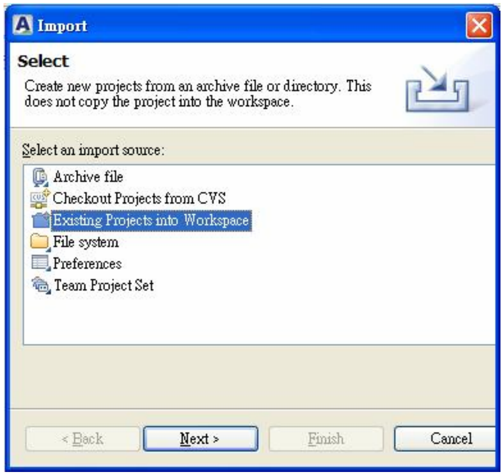
4.1 Audio - MP3 Decoder

As an embedded product, MP3 decoder is a primitive request for Andes SoC solution. This section will guide you into the advance usage of AndeSight using MP3 Decoder project provided by Andes. The VEP configure including processor, memory module, bus, and an audio peripheral will be demonstrated in this project.

You will be experiencing the debugging and finally get the audio output. Through these procedures, you will get further understanding of the integration of Andes IDE, VEP, and Toolchains.

4.1.1 Project Import

- Start *AndeSight* from Windows Start menu
- Import MP3 project by clicking the menu bar **File > import…**
- Select **Existing Projects** into Workspace from the next appearance shown as below, and click **Next**.



Import

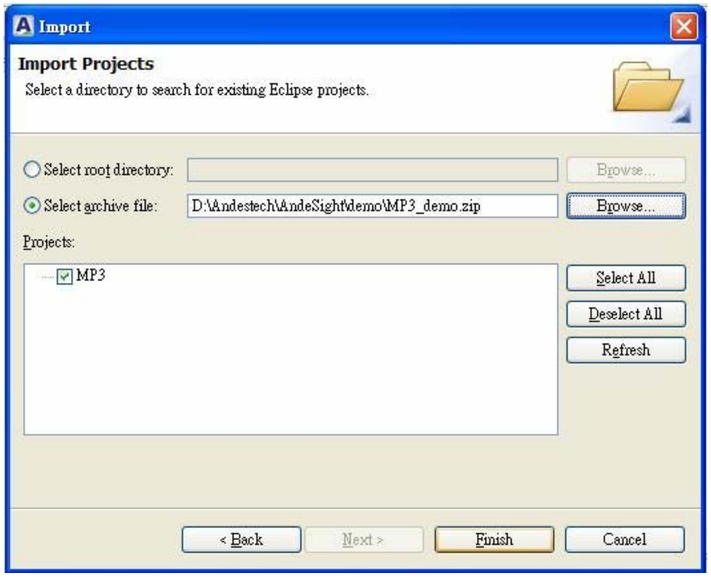
Select

Create new projects from an archive file or directory. This does not copy the project into the workspace.

Select an import source:

Figure 4-1: Import Dialog

- Check the **Select archive file**: and click the **Browse…** button to select the <AndeSight-installed-path>/demo/MP3_demo.zip



Import Projects

Select a directory to search for existing Eclipse projects.

○ Select root directory:

● Select archive file: D:\Andestech\AndeSight\demo\MP3_demo.zip

Projects:

☑ MP3

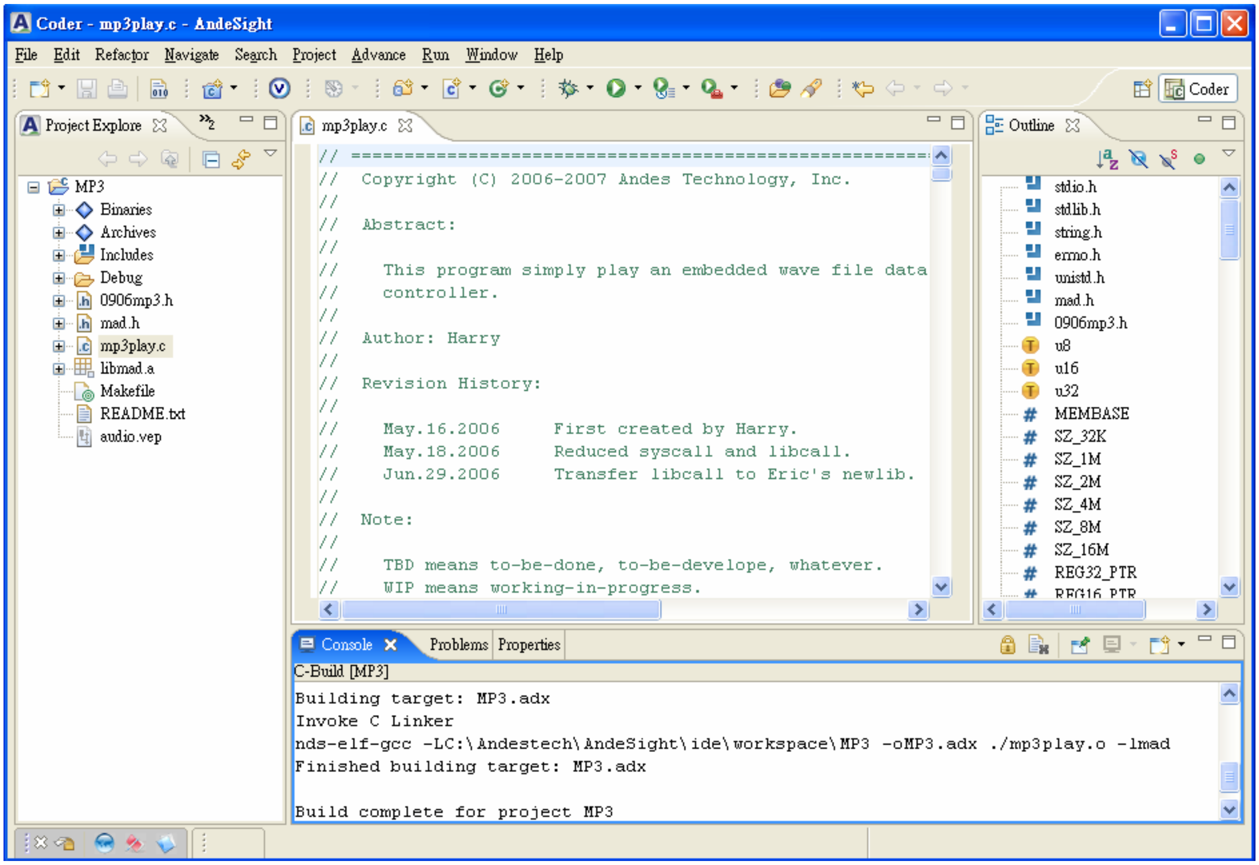
Figure 4-2: Import Projects Dialog

- Press **Finish** to complete the project import.

Note: To import the project, you can either right click to select **Import…** or choose **File > Import…** from the menu bar.

4.1.2 Source Code Editing

- The MP3 project will be shown in the *Project Explore*. You can select the MP3 node and right-click the mouse choosing **Build Project**. Then clicking on the plus sign “**+**” to expand the project, you will see **Binaries**, **Includes**, and **Debug** folders and **mp3play.c** and **builder-gen2.vep** files.
- Double click the **mp3play.c**, the *C/C++ editor* and the *Outline* view of the source code will be invoked.



```
// ======================================================
//  Copyright (C) 2006-2007 Andes Technology, Inc.
//
//  Abstract:
//
//    This program simply play an embedded wave file data
//    controller.
//
//  Author: Harry
//
//  Revision History:
//
//    May.16.2006     First created by Harry.
//    May.18.2006     Reduced syscall and libcall.
//    Jun.29.2006     Transfer libcall to Eric's newlib.
//
//  Note:
//
//    TBD means to-be-done, to-be-develope, whatever.
//    WIP means working-in-progress.
```

```
C-Build [MP3]
Building target: MP3.adx
Invoke C Linker
nds-elf-gcc -LC:\Andestech\AndeSight\ide\workspace\MP3 -oMP3.adx ./mp3play.o -lmad
Finished building target: MP3.adx

Build complete for project MP3
```

Figure 4-3: Source Code Editor and Outline View

4.1.3 VEP Builder

- Double click the **audio.vep**. *VEP builder* will be invoked presenting the configuration of the file and letting you edit or modify.

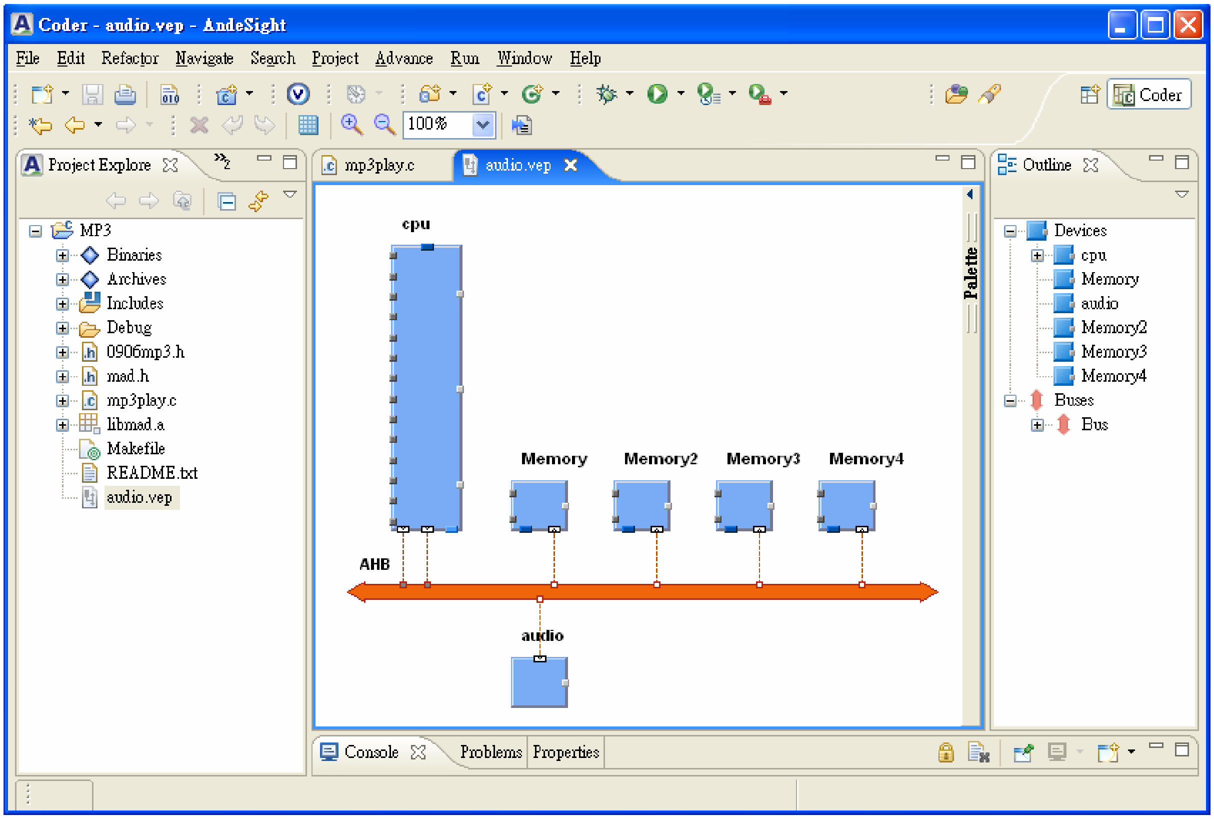


Figure 4-4: VEP Builder

- The *VEP builder* represents the hardware configuration in the high level perspective. It's of functional level connecting IPs like cpu, memory, bus, and an audio device (audio should be connected to APB instead of AHB? Thus APB and AHB-APB bridge should be there). The CPU is simulated by the ISS and the compiled programs will be loaded into Memory. The MP3 will be decoded into PCM format and saved into memory, then sent to the audio device played.

4.1.4 Start GDBAgent and VEP

- Start *VEP* as described in the section 3.4
- Drag the **audio.vep** file from *Project Explore* and drop to the space next to **Select VEP Config**.
- Press the button of **Start Simulator.** Then the simulator and correspond frontend will be invoked as shown in *Figure 4-5*.

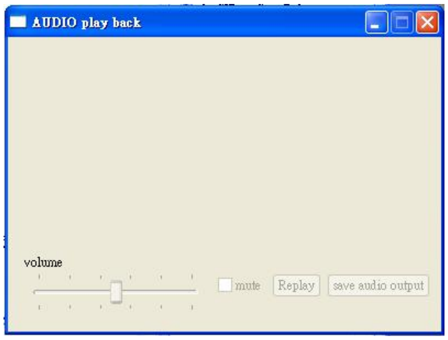
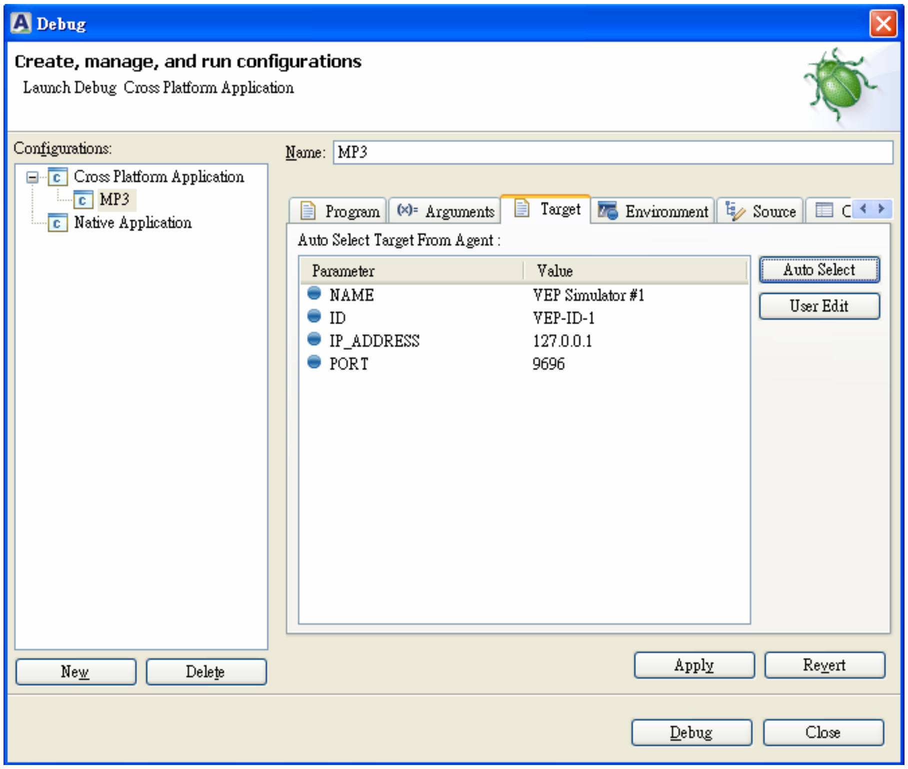


Figure 4-5: MP3 Decoder Front-end

- Set and Select the target as described in the section 3.5

4.1.5 Debugging

- Select the project **MP3** right-clicking the mouse, and from the menu invoked, select **Debug As > Debug…**
- Select the **Cross Platform Applications**, and then press the **New** button.
- AndeSight will automatically setup most execution options for the target and scan the available VEPs in network and list them for you to select. Click the **Auto Select** button as shown in *Figure 4-6*.



Create, manage, and run configurations

Launch Debug Cross Platform Application

Name: MP3

Auto Select Target From Agent :

Parameter	Value
NAME	VEP Simulator #1
ID	VEP-ID-1
IP_ADDRESS	127.0.0.1
PORT	9696

Figure 4-6: Debug Configuration Setup

- Select the target listed and press OK in the next dialog.
- Click Debug button to start debugging. While program is being debugged, the perspective will be switched to *Debug* perspective and debug process will be shown in *Console* view.
- After debugging, the result will be shown from the front-end: Audio simulator.

4.2 JPEG

JPEG decoder is another common program in SoC. This section will guide you into the JPEG project demonstrating decoding eight jpeg images in a set and displaying the result on the emulated LCD panel in the end. The application will use LCD controller and two memory sections to show how to debug between C source code and memory.

Follow the steps below to accomplish the project:

4.2.1 Import the project and start VEP

1. Follow the same steps as the section 3.2 to import the prepared project *JPEG.zip* from <AndeSight-install-path>/demo/
2. Build the project.
3. Start the GDBAgent and SID simulator as described in the section 3.4, but drag the different configuration file from *Project Explore*: **jpeg.vep** to the space next to **Select VEP Config**. Then you will find that VEP front-end - LCD simulator is invoked.
4. Select the target as described the section 3.5.
5. Start debugging and the *Debug* perspective will be open.

4.2.2 Debugging

1. The top left view shows the process, thread, and core stack of current debugging application. And, on the right hand side, there are 4 major views stacked together – *Variables*, *Breakpoints*, *Modules*, and *Registers.*
2. The debugger will initially stop at the first executable code in main function.
3. Variables of current scope will be automatically queried from VEP and listed in the *Variables* view.
4. To see all registers of AndeScore, you could click the plus sign "+" to expand **Main** processor in the *Registers* view.
5. Press Ctrl-F (or click **Edit > find/replace** from the menu bar) or scroll downward for 10 ~ 20 lines to find *djpeg_main*. Right-click on the left side bar (or click **Run > Toggle Line Breakpoint** on the menu bar) to add a breakpoint on this line. You could see all the breakpoints set in the *Breakpoints* view.

6. Press **Resume** button. The execution will continue and stop at the calling of djpeg_main. Click **Step into** , then the execution will go into *djpeg_main()*.
7. Check the *Variables* view, and you will find the variable set is different now.
8. In the *Registers* view, there are several registers changed to red, which means the registers are loaded by different values between breaks.
9. Try to add one more breakpoint on line 744 as the following highlighted:

```
while (cinfo.output_scanline < cinfo.output_height)
 {
   num_scanlines = jpeg_read_scanlines(&cinfo, dest_mgr->buffer,
                                       dest_mgr->buffer_height);
   (*dest_mgr->put_pixel_rows) (&cinfo, dest_mgr, num_scanlines);
 }
```

10. Resume again, you will see the picture is decoded line by line on the virtual LCD panel.
11. Disable all breakpoints by un-checking the checkbox, and press **Resume** button. The application will run with full speed and you will soon see the picture is displayed on the virtual LCD panel.

Note:

1. Above steps show the general breakpoint usage. You could track the current scan line by adding a variable in the code, such as:

```
int current_line = 0;
while (cinfo.output_scanline < cinfo.output_height)
   {
       num_scanlines = jpeg_read_scanlines(&cinfo, dest_mgr->buffer,
                                           dest_mgr->buffer_height);
     (*dest_mgr->put_pixel_rows) (&cinfo, dest_mgr, num_scanlines);
       current_line ++;
   }
```

2. Right-click the breakpoint in the *Breakpoints* view and select **Properties…**. Then input *current_line* == *40* in the **Condition:**
3. Compile and run again. The program will be suspended on the breakpoint only when current_line == 40. This could help debugging more efficient.
4. Eight JPEG pictures are decoded and presented now on the virtual LCD panel.