ADS 1.2 Multi-ICE Integrator Tutorial

Introduction

Aim

This tutorial provides the student with a basic introduction to the facilities provided with the Multi-ICE unit. This will include the use of the unit with an Integrator board, to build and debug example projects. Concepts such as breakpoints and remapping are also demonstrated.

The tutorial is split into two practical sessions:

Session 1 – Simple project
Session 2 – Full embedded project, including remapping

Pre-requisites

This tutorial is intended for use with a Microsoft Windows version of ADS v1.2 and Multi-ICE 2.2. The student should be familiar with Microsoft DOS/Windows, and have a basic knowledge of the C programming language.

This tutorial is intended for us with the Integrator/AP board. If you wish to use the Integrator/CP board, please see Appendix A for the differences.

Note: Explanation of File Extensions:

`.c`	C source file.
`.h`	C header file.
`.o`	object file.
`.s`	assembly language source file.
`.mcp`	project file, as used by the CodeWarrior IDE
`.axf`	ARM Executable file, as produced by **`armlink`**.
`.txt`	ASCII text file.

Additional information

This tutorial is not designed to provide detailed documentation of Multi-ICE. Full documentation is provided with the product in pdf form.

To view the pdf documentation, select *Programs* → *ARM Multi-ICE* → *User Guide* from the Windows *Start* menu.

Further help can be accessed by pressing *F1* when running AXD, from the help menu, or by using the -help switch for a command line tool. The documentation is also available in PDF format within the *ARM, Documentation* subdirectory.

Icon conventions

Various icons are used throughout the tutorial to clarify the purpose of text associated with them. Icons either signify the presence of information on a particular topic, or the requirement for user interaction.

The following icons all indicate that user interaction is required:

Indicates that command line input is required.

Indicates other keyboard or mouse input is required.

Indicates an action involving the hardware is required.

Button icon. This indicates that a corresponding button within the current application can be used to perform the operation currently being discussed.

Application icon. Suggests an application to be used to perform a given operation. This example shows 'Microsoft Notepad'.

To use Notepad from the command line type **`Notepad <filename>`**.

Alternatively click on the Notepad icon on the 'Start menu' and open the required file using the *File→Open* command.

The following icons show information:

Indicates a topic is also dealt with elsewhere in the tutorial.

Suggests that further help is available from other resources.

Identifies a user friendly hint or tip.

Highlights important information regarding the current topic.

Indicates the presence of a 'bug', logic or syntax error in code.

1. Simple demonstration

This section covers using Multi-ICE to examine a simple project running on the target.

Consider the following simple C program which calls a subroutine. This file is provided as **`hello.c`** in **`c:\ads_tutorial\mice\session1\`**

```
#include <stdio.h>

void subroutine (void)
{ printf ("Hello from subroutine\n");
}

int main (void)
{ printf ("Hello from Main\n");
  subroutine();
  printf ("And goodbye\n");
  return (0);
}
```

1.1 Setting up the target and the Multi-ICE Server

Firstly set up the Integrator board:

Set the Integrator switches 1 and 4 to the *ON* position. Connect the Multi-ICE to the PC and the Integrator board. Then power up the Integrator.

Start the Multi-ICE server on the PC by selecting *Programs* → *ARM Multi-ICE* → *Multi-ICE Server*

Configure the Multi-ICE server using the Auto-Configure icon. Note that auto-configuration may have already taken place if selected in *Settings* →*Startup Options*.

Notice that in the Multi-ICE server window, “ARMxxx” is shown in green to indicate that nothing is using the Multi-ICE and it is free for a debugger to connect to it. The [X] means that the core state is unknown, again because no debugger is connected.

1.2 Loading the simple project into CodeWarrior

Now we can load the simple project:

Navigate to the **`C:\ads_tutorial\mice\session1`** directory in Windows Explorer. Double-click on "hello.mcp" to load the project into the CodeWarrior IDE.

Build the image:

Make the project by selecting *Project* → *Make* from the menu, or press *F7*.

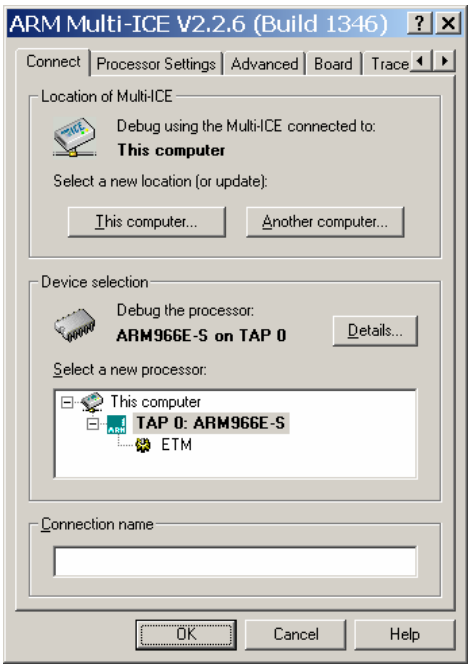
The DebugRel image that we create is equivalent to using the **`armcc`** command-line options **`-g+ -O1`**. These provide adequate debugging along with good optimisation.

1.3 Starting the Debugger

Start AXD by selecting *Project* → *Debug* from the menu, or press *F5*.

Connect to the Multi-ICE target:

Configure the target connection by selecting *Options* → *Configure Target* from the menu. Select Multi-ICE from the list of available target types, then click *Configure*.



Ensure that *This computer* is selected as the location for the Multi-ICE. Highlight the connection as shown above, then click *OK* to close the window and return to the *Choose Target* window. Close this by clicking the *OK* button.

You may be asked to reload the last image. If so, click *Yes*.

Now see the changes that are shown in the Multi-ICE server window:

Switch to the Multi-ICE server window from the Windows taskbar.

Notice that the processor is now red. This shows that it is in use and cannot be connected to by another client. The [S] indicates that the core is stopped. Finally, the lower pane of the window shows communication with the Multi-ICE.

Return to AXD using the Windows taskbar.

The final thing to prepare is to set the top of memory as appropriate for the Integrator:

Select *System Views → Debugger Internals* (or press *Alt+D*) to display the list of AXD internal variables. Double-click the value for $top_of_memory and change the value to 0x00040000.
Close this pane by right-clicking inside it and selecting *Close*.

1.4 Setting Breakpoints

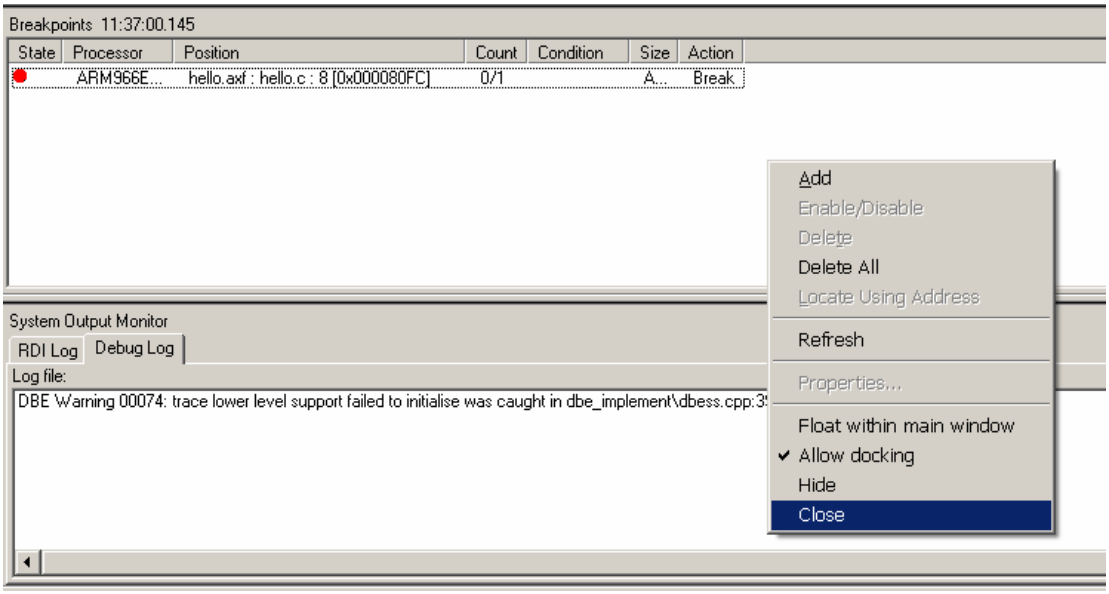
Click on the *Go* button on the toolbar, or press *F5*.

The debugger will run through the C library initialisation code and stop at main(). This breakpoint was placed there automatically by the debugger.

View the list of breakpoints by selecting *System Views → Breakpoints*, or type *br* in the command window.

You will notice the breakpoint that is set at main().

Close the Breakpoints pane by right-clicking inside it and selecting *Close*.



Highlight the C source window, if necessary by selecting it from the *Window* menu. Right-click in this window and select *Interleave Disassembly*.

The ARM instructions that map onto each C statement are shown.

Set a breakpoint on the call to subroutine() on line 9 by double-clicking in the grey area to the left of the code.

This sets up a new breakpoint. You can check this from the breakpoint view again:

Select *System Views →Breakpoints*, or type *br* in the command window.

Finish running the example:

Select *Execute →Go* (*F5*) to continue to the new breakpoint, followed by *Execute → Step In* (*F8*) to follow the call into the subroutine. Then step through the example using *Execute → Step* (*F10*) until it terminates.

Notice that the output from the program still appears in the AXD *Console* window. This is because many of the input/output functions from the C library are *semihosted*. Semihosting is a mechanism for I/O requests to be passed to the host system, rather than run on the target. This means that, for example, the keyboard and mouse of a debugging host can be used.

Close AXD and CodeWarrior by selecting *File → Exit* in each to finish the exercise.

2. Full Embedded Application

This session uses a more complex embedded example

Ensure that the Integrator motherboard switches S1-1 and S1-4 are on. Then reset the board.

These settings cause the boot monitor to run and then poll waiting for an input following a reset.

2.1 Load project into CodeWarrior

Start by loading the project:

Navigate to the **`C:\ads_tutorial\mice\session2`** directory in Windows Explorer. Double-click on "ledflash.mcp" to load the project into the CodeWarrior IDE.

Now look at the scat.scf file to see an example of scatter loading:

Double-click scat.scf in the project window to open it.

The scatter file shows:

-One Load Region	Start 0x24000000, size 0x4000000
-Two Execution Regions	
-Flash	Start 0x24000000, size 0x4000000
-Ram	Start 0x00000000, size 0x003FFFF

Now build the project, using the *DebugRel* settings.

Remove any existing object code by selecting *Project → Remove Object Code* and selecting *All Targets*. Make sure that DebugRel is selected in the drop-down menu at the top of the project window.
Finally, select *Project →Make* (*F7*) to build the image.

2.2 Downloading the binary image to flash

If it is not already running, start the Multi-ICE server from the Windows start menu.

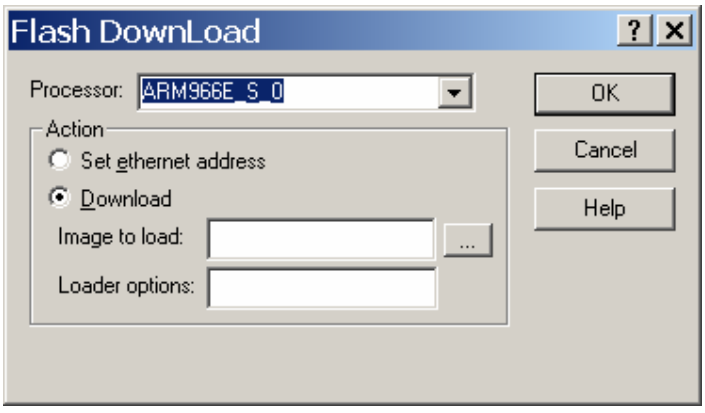
Click the auto-configure button to set up the connection to the target.

Auto-configuration may have already taken place if selected in *Settings->Start-up options*. As in Session One, in the Multi-ICE server window, 'ARMxxxx' is shown in green. The [X] box means that the core state is unknown, i.e. that no debugger is connected.

Start AXD from the Windows Start menu, under *Programs* → *ARM Developer Stuie* → *AXD Debugger*

Select *System Views* → *Debugger Internals*, and set $top_of_memory to 0x00040000.

Select *File* →*Flash Download*. Click the "…" button next to the "Image to load" text box, and navigate to:
`C:\ads_tutorial\mice\session2\ledflash_data\DebugRel`
and select **`ledflash.bin`**



In the *Loader options* text box, type "a0x24000000 i0 *LedFlash" (without quotes)

This loads the image starting at 0x24000000, with an image number ‘0’ and named ‘LedFlash’. The image number and name are only used by the boot monitor.

Click *OK*. The flash downloader will ask you to confirm the details in the console window. Type “*y*” (no quotes) and press *Enter*. If there is already an image in the flash, you will be asked to overwrite it. Say yes here also.

The image is downloaded to the flash, and a confirmation message appears. Click *OK* to close it.

2.3 Running the program from flash without Multi-ICE attached

Power down the Integrator board.

Close AXD, CodeWarrior and the Multi-ICE server, by selecting *File* →*Exit* in each.

Unplug the Multi-ICE from the Integrator board, and set switch S1-1 off. Power up the board.

The Integrator LEDs should light up according to the switch settings.

Try changing the settings of switches S1-1 to S1-4. These alter the speed and pattern of the lights.

2.4 Reconnect Multi-ICE and AXD

Power down the Integrator board again, and reconnect the Multi-ICE unit to the board. Ensure switch S1-1 is down, and re-power the board.

The board LEDs should begin to flash again.

Re-start the Multi-ICE server from the Windows start menu.

Click the auto-configure button to set up the connection to the target.

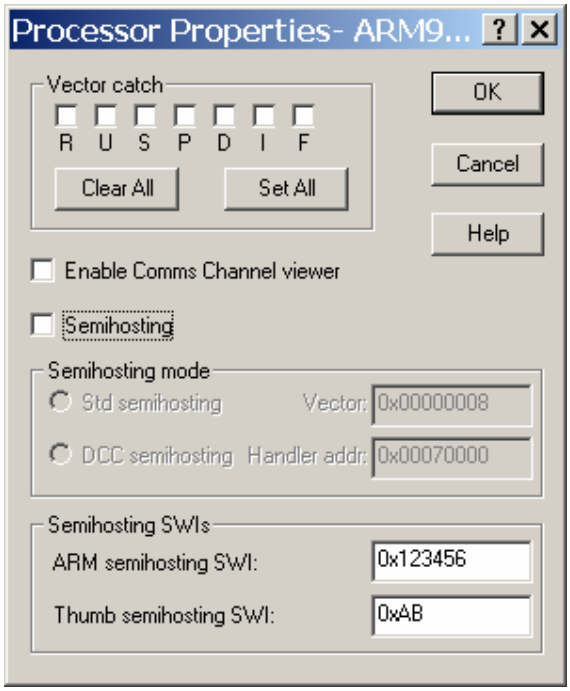
In the Multi-ICE server window, the processor ‘ARMxxxx’ is shown in green again.

Start AXD from the Windows Start menu, under *Programs* → *ARM Developer Stuie* → *AXD Debugger*

The LEDs will stop flashing as the core is stopped by AXD.
If you look at the Multi-ICE server window, the processor is now red showing that it is in use and cannot be connected to by another client. The [S] signifies that the core is stopped.

We now need to disable vector catch so that our own exception handlers are used, and stop semihosting breakpoints being set.

In the AXD window, ensure that the *Target* tab is selected. Right-click the processor name and select *Properties*. In the dialog box that appears, click the *Clear all* button. Also deselect the semihosting checkbox. Finally, click the *OK* button to apply the settings and close the window.



2.5 Start debugging from 0x0

Ensure that switch S1-1 remains down. Now we can set a breakpoint to allow us to debug the embedded application from reset.

If the command-line interface window is not visible in AXD, show it by selecting *System Views* → *Command Line Interface* (*Alt+K*) from the menu.

Type "break 0x0" at the prompt to set a new breakpoint. Then select *Execute* → *Go* (or press F5) to resume execution.

The LEDs continue to flash as the core resumes execution.

Press the reset button on the Integrator board.

Only reset the board whilst it is running. Resetting it when stopped causes connection to be lost.

A dialog box appears saying "The target board has been reset, attempting to Recover." Click *OK*.
If another message saying "Processor ARMxxx raised an exception" is Displayed, click *OK* in this dialog box also.

The breakpoint at 0x0 will be hit and control passed back to the debugger.

2.6 Demonstrating Memory Remap

If the Disassembly window is already open, close it as it will not update automatically.

Now examine the memory on the board:

Select *Processor Views* → *Memory* from the menu to display the memory pane. Enter 0x0 as the start address.

Now right-click in the memory pane. Select *Format → ARM* to show a disassembly of the memory being viewed.

The disassembly shows the code in flash –this is the same as code at 0x24000000.

Select *Execute → Step* (*F10*). The execution jumps to 0x24000004.

Step a further three instructions, stopping on the STR instruction.

Step the STR instruction, and notice how the whole memory display changes.

Remap has now taken place. The contents of 0x0 are now RAM-based code rather than ROM-based code. The exception vectors may already be in place because they were installed when we powered the board up to show the flashing LEDs code running from Flash memory.

2.7 Loading Debug Symbols

Select *Execute → Go* (*F5*) from the menu; note that the Multi-ICE server window shows [R] to denote the core running.

Stop execution using *Execute → Stop* (*Shift+F5*). Perform a single step of the next instruction by pressing *F10*.

Although this allows us to single step through disassembly, it is much more useful to be able to step through source code, including labels in assembly.

Select *File → Load Debug Symbols*. Navigate to the **`C:\ads_tutorial\mice\session2\ledflash_data\DebugRel`** and select ledflash.axf. The debugging symbols are loaded and the source code is shown in the AXD window.

Single step the C source by pressing *F10*.

Close AXD and the Multi-ICE server by selecting *File* → *Exit* in each. Power down the Integrator board to finish the exercise.

Appendix A: Differences with the Integrator/CP

The Integrator/AP and Integrator/CP boards are very similar. However, there is one important difference as far as this example is concerned. The timer on the AP has only a single enable bit in its control register. In the CP boards, this has been extended and there is an additional interrupt enable bit.

In order for the embedded example in the tutorial to function correctly, the value sent to the timer's control register must be changed. To do this manually, follow the instructions below:

- Navigate to the **`C:\ads_tutorial\mice\include`** directory. Open the **`intgrt.h`** header file.
- Scroll down to line 96, which reads:
  `#define TimerEnable 0x80`
- Change the value of **`TimerEnable`** to **`0xA0`**.
- Save the changes to the file.

The change has already been made in the additional file **`intgrt_cp.h`**; to use it, first rename the original file (for example, to **`intgrt.h.backup`**) then rename the new file to **`intgrt.h`**. Note also that you may need to set switches 1 and 2 on while the flashing LEDs example is running for the flash rate to be suitably fast.

For more information on the differences between the two Integrator boards, please see Appendix A in the Integrator/CP User Guide.

Appendix B: Motherboard Settings at Boot

Configuration 1:	S1-1 **off** causes the board to boot from Application Flash. Flash at 0x24000000 is mapped to 0x0. The application code will normally remap this, so that RAM appears at zero.
Configuration 2:	S1-1 **on** and S1-4 **on** causes boot monitor to run and give command prompt. Jumps to 'Boot ROM' at 0x20000000 then Remaps
Configuration 3:	S1-1 on and S1-4 off causes boot monitor to run then jump to Application flash. The boot ROM takes care of remapping ROM/RAM.

If a flash download fails

To enable a flash download to work, memory remap must have occurred. This can be performed by the boot monitor or the LED flash program. In the example, the boot monitor is used to perform this. In addition, **`$top_of_memory`** must be set to **`0x00040000`**.