EFI Framework Debugging

Overview

First introduced on Intel® Itanium® processor systems, the Intel® Platform Innovation Framework for Extensible Firmware Interface (EFI), commonly known as the EFI Framework, is a new firmware architecture standard that defines a set of software interfaces and replaces the legacy BIOS found on traditional PC computers. This framework provides the kind of modularity, flexibility, and extensibility that were formerly unavailable with traditional BIOS. With EFI, BIOS developers can now write all their code in 'C', rather than assembly language. See Intel's website at http://www.intel.com/technology/framework for more information on EFI Framework.

Along with this new firmware architecture and the 'C' code that implements it comes the need for source-level debugging. Arium's debugger, SourcePoint™ (versions 7.0 and later) for Intel IA-32 and IA-64 processors, offers native debug support for EFI Framework platforms. Users can set breakpoints, single step, view variables, see the call stack, and access all of the feature-rich functionality SourcePoint normally provides. This includes source-level debugging during the PEI, DXE, and OS Boot phases of EFI. Below is a set of instructions for setting up SourcePoint to debug the EFI Framework.

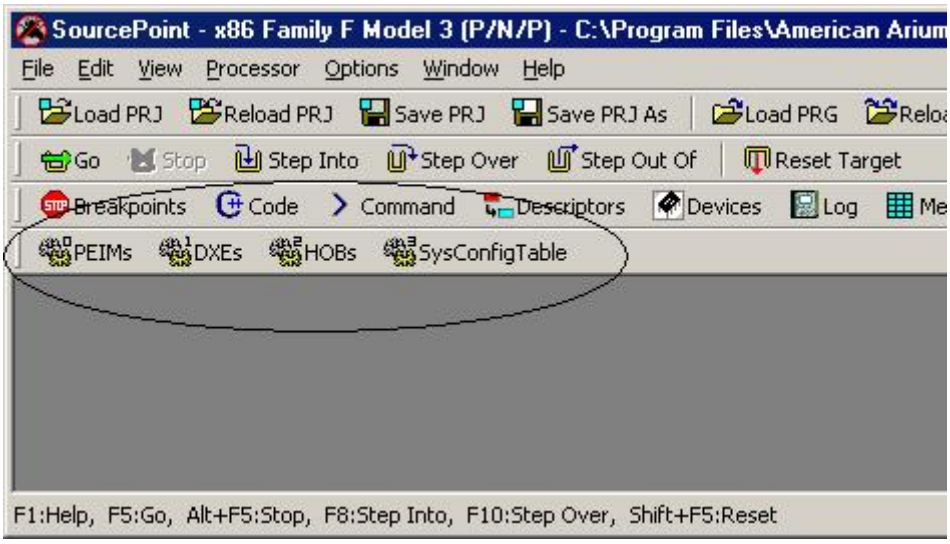
EFI Macros

Note: The macros described below macros are installed into the Macro\EFI sub-folder of the SourcePoint install path. Several of the EFI macro files contain directory paths to other macro files. If you move the macro files or change the current working directory in SourcePoint (via the 'cwd' command), you need to update the macros files with the new locations.

SetupEFI.mac

After installing SourcePoint, run the SetupEFI.mac macro file located in the Macro\EFI directory. This creates four custom toolbar buttons and associates each with their corresponding EFI macro file.

- The Pre-EFI Initialization Modules (PEIMs) icon loads the symbol files for the PEI modules found in target memory.
- The Driver Execution Environments (DXEs) icon loads the symbol files for the DXE modules found in target memory.
- The Hand-Off Blocks (HOBs) icon displays a list of EFI HOBs found in target memory.
- The SysConfigTable icon displays the contents of the EFI system configuration table.



EFI_Functions.mac

The EFI_Functions.mac macro file contains the support functions. Inside this file, a constant definition called gEfiSystemTablePointerStart may need to be changed depending on the location of the top of RAM in the target.

PEI Debugging

The PEI environment requires a specialized configuration of SourcePoint. PEI gets control shortly after target reset. PEI modules are dispatched and executed after cache RAM is mapped into system memory, and the stack is initialized. Having a stack this early allows 'C' language code to execute, but a special memory map must be configured to take advantage of it.

To configure SourcePoint for source-level debugging of PEI code, follow these steps.

1. Select **Options|Target Configuration|Memory Map** from the menu, and set it up as follows:

Start	End	Type
00000000P	000FFFFFP	DRAM
FEF00000P	FFEFFFFFP	SRAM
FFF00000P	FFFFFFFFP	FLASH

   Enable Safe mode.

   The first entry in the table designates the first 1MB of system memory. The middle entry designates the location of the cache RAM mapped into system memory. The third entry designates the firmware ROM. These parameters may vary depending on the platform.

2. Select **Options|Preferences|Breakpoints** and set the default breakpoint type to **Processor**.
3. Set a breakpoint at a location shortly after the cache RAM is mapped into system memory, but before it reaches the first 'C' code. The address FFFFFDD1P worked for our initial target, but this location will vary.
4. Click the PEIMs toolbar icon to load the PEIM symbols.
5. Set a breakpoint at the symbol SecStartup.
6. Run to the next breakpoint.

You should now be at a source code window showing the start of the PEIM code at SecStartup.

DXE Debugging

Once system RAM is initialized and the PEI phase completes, the DXE environment is entered. This is less specialized than PEI, but, nevertheless, requires a few SourcePoint parameters to be set.

To configure SourcePoint for source-level debugging of DXE code, follow these steps:

1. Select **Options|Target Configuration|Memory Map** from the menu, and set it up as follows:

Start	End	Type
00000000P	FFFFFFFFP	DRAM

   The entry in the table designates the entire 4GB address space as DRAM.
2. Set Safe mode to disabled.
3. Run the target to the EFI shell, or as far as it will go in DXE.
4. Stop the target.
5. Click the DXEs toolbar icon to load the DXE symbols.
6. Browse the source code files using the **Symbols** window and set breakpoints in your code.
7. Reset the target, and go until you hit a breakpoint.

HOBs

Open the **Command** window, and then click the HOBs toolbar icon to display the hand-off blocks on the target.

System Configuration Table

Open the **Command** window, and then click the **SysConfigTable** toolbar icon to display the contents of the EFI system configuration table on the target.

Notes

1. Multiple Project Files Tip

   It may be convenient to create one project file for PEI debugging and another project file for DXE debugging. That way you can quickly pull up the SourcePoint configuration you need for that debugging session.
2. DXE Debugging Tip

   To stop the target and load symbols just before a DXE module is dispatched, open the **Symbols** window, choose the **Globals** tab, and drill down to:

   program: DXEMAIN.efi

module: image (image.c)

function: CoreStartImage()

Right click on **CoreStartImage** and select **Open Code Window** from the pop-up menu.

Set a processor breakpoint in **CoreStartImage()** where **Image|EntryPoint()** is called. This hits before each DXE module is dispatched, but afterwards its entry is placed in tables. Each time you hit this breakpoint, click the DXEs toolbar icon to load the DXE symbols.

Now you can load symbols just before your DXE module runs instead of running to the EFI shell, then loading symbols, then resetting the target, then running to your breakpoint.

3. Watchdog Timer on Intel Platforms

   Some motherboards with Intel processors have a TCO timer that will assert RESET independent of the emulator. See the Arium application note titled, "Disabling the TCO Timer in an Intel I/O Controller Hub" for details. Resetting the target from SourcePoint can cause a **Target state undefined** error message to appear because the timer asserts RESET and confuses the emulator. The solution to this problem is to configure the ICH_TCO_Timer_Disable.mac macro to run at every target reset.

4. The EFI firmware on the target contains strings that hold the paths to the program symbol files on your hard drive. Our macros read target memory, find these strings, then load the symbol files specified in these paths. The symbol files must be located in the path specified in the EFI firmware.

   For example, one path might look like this:

   "Z:\Platform\IntelSsg\D845GRG\Build\IA32\DxeMain.efi"

   This architecture, defined by Intel, presents a requirement for EFI debugging. You must have the EFI symbol files on the host computer in the same directories as specified in the firmware on the target. This should not be a problem if you build the EFI firmware on the same host from which you run SourcePoint.

   If the drive letter or path doesn't match exactly, you can use the 'subst' command from the Windows command prompt to map a drive letter to a desired path.

14811 Myford Road, Tustin, CA 92780 Voice: 877-508-3970 in the US, 714-731-1661 outside the US Fax: 714-731-6344 E-mail: info@arium.com Web www.arium.com **A0163C**