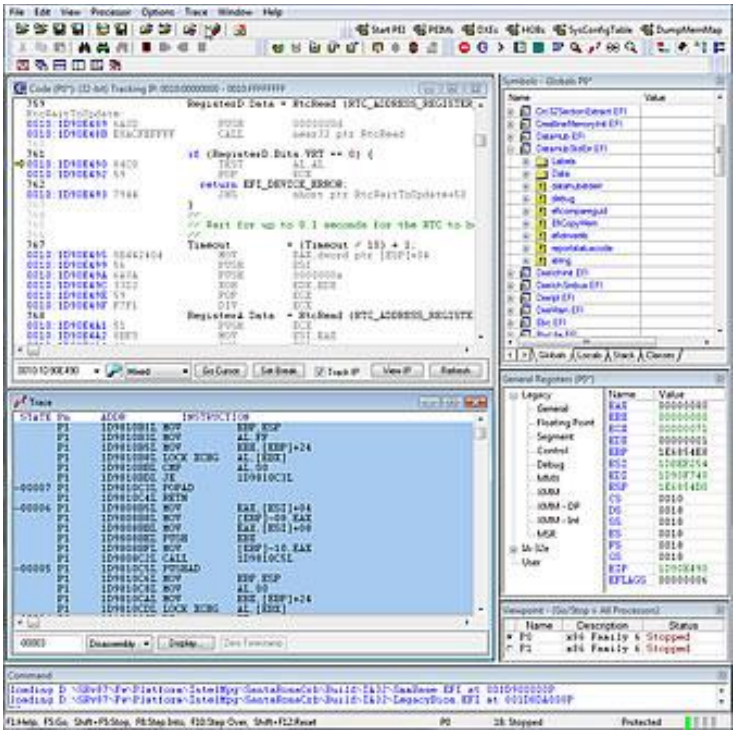
ASSET System Trace Library - User Guide (update 2144) October 24, 2021



For use with SourcePoint® debugger by ASSET InterTech Inc.

TABLE OF CONTENTS

Overview 4
General Description 4
Kit Contents 4
Implementation Overview 5
DEBUG 5
ASSERT_EFI_ERROR 6
MRCPRINTF 6
DEBUG_GPR_A, DEBUG_GPR_B, DEBUG_GPR_C, DEBUG_GPR_D 7
DEBUG_GPRS 7
DEBUG_RIP 8
EDKII Integration Steps 9
Extract and copy files 9
Amend Edk2 Source Tree 9
Target Specific Package 10
Option 1. Replace Existing Debug Library 11
Option 2. Compiler Option to include Trace Library 11
Pre-Memory Initialization 12
SourcePoint Integration Overview 13
Configure SourcePoint for Trace Hub 13
Setting Trace for Trace Hub 13
Configure Trace Hub Options 13
Validate Debug Output 17
Glossary 19
Related Documents 19
ASSET Contacts: 19
Appendix B – Example Status MetaData 21

TABLE OF FIGURES

Figure 1: Modules Package EDKII Source Tree Layout 9
Figure 2: Modules Package Description Modification 10
Figure 3: Existing Debug Library Entry. 11
Figure 4: SourcePoint System Trace Debug Library Entry. 11
Figure 5: Existing FSP Debug Library Entry. 11
Figure 6: SourcePoint FSP System Trace Debug Library Entry. 11
Figure 7: Conditional Compilation of Library Example. 12
Figure 8: Conditional FSP Compilation of Library Example. 12
Figure 9: Compiler Build Flags. 12
Figure 10: Include Trace Library for Pre-Memory. 12
Figure 11: Trace Icon. 13
Figure 12: Show SW/FW Trace Option. 13
Figure 13: Configure Trace Hub Options. 14
Figure 14: Trace Hub Configuration Tab. 14
Figure 15: Selecting Masters to Trace. 15
Figure 16: Viewing Master Names. 15
Figure 17: Trace Buffer Settings. 16
Figure 18: Metadata File Browse Icon. 16
Figure 19: Metadata File Selection. 16
Figure 20: Trace Hub Trace Output. 18

Overview

This document will outline the procedure for enabling the SourcePoint System Trace Library modules to be utilized within an EDKII build tree. It is assumed the end user has a full working knowledge of the EDKII build process and modification of files. This document does not cover integration of the Intel® silicon specific files into the EDKII build tree. Contact your local Intel® FAE for these files and integration instructions.

The SourcePoint System Trace Library takes advantage of a new feature available within Intel silicon, named Trace Hub.

The SourcePoint System Trace Library will allow re-direction of multiple debug sources (console output, management engine, UEFI debug messages, etc.) to be routed via the Trace Hub, eliminating the need of a console or serial debug port.

General Description

The Zip archive included with this document contains the files necessary to allow the integration of the SourcePoint System Trace Library. Some files are ©Copyright ASSET InterTech Inc. and are distributed under license.

Kit Contents

The ASSET System Trace Library kit has an existing folder structure to match the current EDKII source tree. At the time of writing, this is located on GitHub in a private repository as **CCG-Generic-EDK2Platforms** (contact your Intel® FAE for further details and access), and on the ASSET website at www.asset-intertech.com/sourcepoint-academy/at-speed-printf. The ASSET System Trace Library kit includes the following files:

MdePkg\Include\Library\BaseLib.h.merge**
MdePkg\Include\Library\DebugLibSystemTrace.h
MdePkg\Include\Library\DebugLibSystemTraceSmm.h
MdePkg\Include\Library\SerialPortLibSystemTrace.h
MdePkg\Library\BaseDebugLibSystemTrace\BaseDebugLibSystemTrace.inf
MdePkg\Library\BaseDebugLibSystemTrace\DebugLib.c
MdePkg\Library\BaseDebugLibSystemTraceSmm\BaseDebugLibSystemTraceSmm.inf
MdePkg\Library\BaseDebugLibSystemTraceSmm\DebugLib.c
MdePkg\Library\BaseLib\BaseLib.inf.merge**
MdePkg\Library\BaseLib\Ia32\ReadGprA.asm
MdePkg\Library\BaseLib\Ia32\ReadGprA.c
MdePkg\Library\BaseLib\Ia32\ReadGprA.nasm
MdePkg\Library\BaseLib\Ia32\ReadGprB.asm
MdePkg\Library\BaseLib\Ia32\ReadGprB.c
MdePkg\Library\BaseLib\Ia32\ReadGprB.nasm
MdePkg\Library\BaseLib\Ia32\ReadGprC.asm
MdePkg\Library\BaseLib\Ia32\ReadGprC.c
MdePkg\Library\BaseLib\Ia32\ReadGprC.nasm
MdePkg\Library\BaseLib\Ia32\ReadGprD.asm
MdePkg\Library\BaseLib\Ia32\ReadGprD.c

MdePkg\Library\BaseLib\Ia32\ReadGprD.nasm
MdePkg\Library\BaseLib\Ia32\ReadRip.asm
MdePkg\Library\BaseLib\Ia32\ReadRip.c
MdePkg\Library\BaseLib\Ia32\ReadRip.nasm
MdePkg\Library\BaseLib\X64\ReadGprA.asm
MdePkg\Library\BaseLib\X64\ReadGprA.nasm
MdePkg\Library\BaseLib\X64\ReadGprB.asm
MdePkg\Library\BaseLib\X64\ReadGprB.nasm
MdePkg\Library\BaseLib\X64\ReadGprC.asm
MdePkg\Library\BaseLib\X64\ReadGprC.nasm
MdePkg\Library\BaseLib\X64\ReadGprD.asm
MdePkg\Library\BaseLib\X64\ReadGprD.nasm
MdePkg\Library\BaseLib\X64\ReadRip.asm
MdePkg\Library\BaseLib\X64\ReadRip.nasm

** These files are provided to allow 'copy and paste' of the text from their respective sections, into the original file located in the source tree.

Implementation Overview

The EDKII framework has many debug hooks for displaying progress throughout the boot process. These hooks can direct the debug information via different interfaces, namely USB, UART (Serial Port) or Memory. Most debug information is defined using an inline macro that can format the output based on type or section of the EDKII code base.

With the current EDKII framework, three debug hooks have been identified, DEBUG, MRCPRINTF (Intel® specific), and ASSERT_EFI_ERROR. DEBUG and ASSERT_EFI_ERROR have a root-debug hook that calls SerialPortWrite(). The SourcePoint System Trace Library can utilize these hooks to route debug information via STM instead of the default path.

Each hook type is discussed in the following text:

DEBUG

Is used primarily to provide boot status information whilst running/executing the EDKII source. These include entry/exit to/from SEC, PEIMs, DXE, and various other source branches. Some typical entries are shown:

- DEBUG((EFI_D_ERROR, "Microcode not found.\n"));
- DEBUG((EFI_D_INFO,"BIOS Prog: lanePwrMask %05x\n", lanemask));
- DEBUG((EFI_D_ERROR, "IndexNumber:%d MemoryDataNumber%d \n", IndexNumber,DataSize/ sizeof (EFI_MEMORY_TYPE_INFORMATION)));

Although the EDKII framework has defined the DEBUG hook as a macro, it performs a function call to "DebugPrint". Currently this uses the serial port and presents a significant increase in the execution time of the boot process.

ASSERT_EFI_ERROR

ASSERT_EFI_ERROR is very similar to DEBUG, however, this hook is used for dual purposes.

Firstly, a comparison is performed based on the input parameters of the structure (Status). If the Status level parameter has its most significant bit (MSB) bit clear, then code execution returns to the calling procedure.

Secondly, if the MSB bit is set, a call to DEBUG is executed with the string “ASSERT_EFI_ERROR”, and then using the Platform Build Flag for Debug (PCD) options in the build process, the code would either cause a processor breakpoint, or a while(1) loop will be entered.

A typical entry is shown with ***Status*** as the only parameter. This parameter is of type EFI_STATUS:

- ASSERT_EFI_ERROR (Status);

MRCPRINTF

MRCPRINTF is used at a very early stage of the boot process before most of the EDKII infrastructure has been configured. This is primarily used in the Memory Reference Code (MRC) section of the pre-EFI phase. The MRCPRINTF is very similar to a standard printf function call, and as such, can be easily substituted with the desired At-Scale printf.

Some typical entries are shown:

- mrcPrintf (CurrentMrcData->ModMrcData.MrcDebugMsgLevel, SDBG_MAX, "C%dD%dR%d RTT_WR_60\n", Channel, Dimm, Rank);
- mrcPrintf (ModMrcData->MrcDebugMsgLevel, SDBG_MAX, "%03d:%d ", Center[Strobe], Test1[Strobe]);
- mrcPrintf (ModMrcData->MrcDebugMsgLevel, SDBG_MAX, " %03d   ", TempValue);

Unlike the other hooks, mrcPrintf is not a macro. To allow use with the Trace Hub, the function call needs to be changed to match with the SourcePoint System Trace Library. The equivalent entries are shown:

- mrcHprintf3 (CurrentMrcData->ModMrcData.MrcDebugMsgLevel, SDBG_MAX, "C%dD%dR%d RTT_WR_60\n", Channel, Dimm, Rank);
- mrcHprintf2 (ModMrcData->MrcDebugMsgLevel, SDBG_MAX, "%03d:%d ", Center[Strobe], Test1[Strobe]);
- mrcHprintf1 (ModMrcData->MrcDebugMsgLevel, SDBG_MAX, " %03d   ", TempValue);

DEBUG_GPR_A, DEBUG_GPR_B, DEBUG_GPR_C, DEBUG_GPR_D

This set of macros are not part of the EDKII framework. They are provided as part of the SourcePoint System Trace Library. These macros are used to provide the data associated with a general purpose processor register at a specific time in the code execution. These macros allow inspection of the GPRs to co-inside with any other trace data that may be targeted at the Trace Hub.

Very similar to DEBUG, these macros can be inserted at any point in the source tree to allow debugging of code with data.

Some typical entries are shown using general purpose registers. This code assumes that variables regAddr and regData have been placed into GPR_A and GPR_D respectively by the compiler. This can be confirmed by using SourcePoint to view the GPR's at the relevant section of the code. The macro does not have any parameters:

```
uint32_t *regAddr = 0x10000000;
uint32_t regData= 0;

for (n = 0; n < 0x100; n++)
{
        *(volatile UINT32 *) regAddr = regData++;

        // display current values for GPR A and D via System Trace Library
        DEBUG_GPR_A();
        DEBUG_GPR_D();
}
```

DEBUG_GPRS

This macro is not part of the EDKII framework. It is provided as part of the SourcePoint System Trace Library. This macro is used to provide the data associated with the general-purpose processor registers A through D at a specific time in the code execution.

A typical entry is shown using the macro for the general-purpose registers A to D. This code will allow GPRs A through D for a 32 and 16 bit write, but not an 8 bit write. Again, this code assumes that variables are mapped to GPRs A through D respectively by the compiler. This can be confirmed by using SourcePoint to view the GPR's at the relevant section of the code. The macro does not have any parameters:

```
Size = (uint8_t)RegOffset.Bits.size;
RegAddr = GetCpuPciCfgAddress (host, SocId, BoxInst, Offset, &Size);

//
// Check register size and write data
//
switch (Size) {
case sizeof (uint32_t):
        *(volatile uint32_t *) RegAddr = Data;
        DEBUG_GPRS();
        break;
```

```
case sizeof (uint16_t):
        *(volatile uint16_t *) RegAddr = (UINT16) Data;
        DEBUG_GPRS();
        break;

case sizeof (uint8_t):
        *(volatile uint8_t *) RegAddr = (UINT8) Data;
        break;

default: Stop;
}
```

DEBUG_RIP

This macro is not part of the EDKII framework. It is provided as part of the SourcePoint System Trace Library. This macro is used to provide the data associated with the instruction pointer at a specific time in the code execution.

A typical entry is shown using the macro for the Instruction Pointer (RIP). This code will allow displaying of the Instruction Pointer via the System Trace Library for a 16 bit write only. The macro does not have any parameters:

```
Size = (uint8_t)RegOffset.Bits.size;
RegAddr = GetCpuPciCfgAddress (host, SocId, BoxInst, Offset, &Size);

//
// Check register size and write data
//
switch (Size) {
case sizeof (uint32_t):
        *(volatile uint32_t *) RegAddr = Data;
        break;

case sizeof (uint16_t):
        *(volatile uint16_t *) RegAddr = (uint16_t) Data;
        DEBUG_RIP();
        break;

case sizeof (uint8_t):
        *(volatile uint8_t *) RegAddr = (uint8_t) Data;
        break;

default: Stop;
}
```

EDKII Integration Steps

It is assumed the end user has a full working knowledge of the EDKII build process and modification of files. The following steps allow the integration of the SourcePoint System Trace Library; however, they do not cover the building of the source tree for specific Intel® silicon. Contact your local Intel® FAE for instructions on how to include silicon specific files.

Extract and copy files

1. Extract the Kit contents into a temporary folder of your choice.

2. Copy the *Edk2\MdePkg* folder to the *<root>\Edk2* folder of the source tree where you will be building the target image. The EDKII source tree should have a sub-folder from the <root> called *Edk2* of which the *MdePkg* folder should be available (as seen in Figure 1).

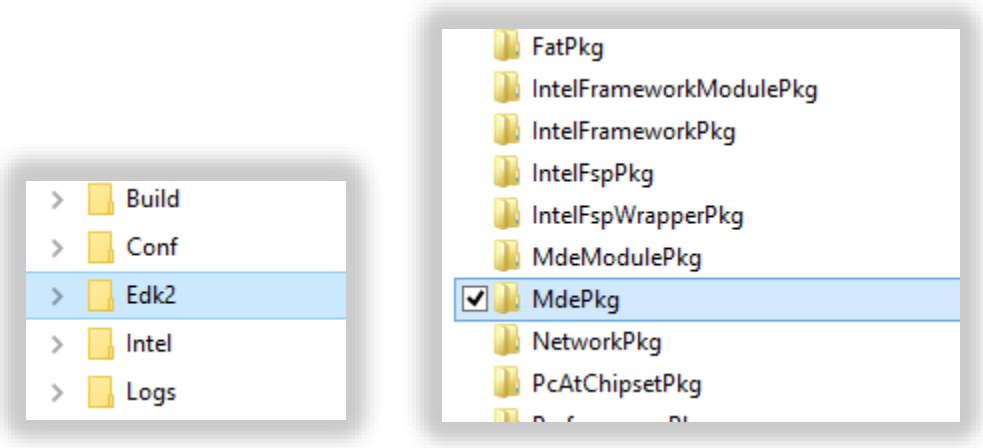


Figure 1: Modules Package EDKII Source Tree Layout

Note: The supplied Kit has a merge file included that can be used to copy the content.

3. Copy the *Intel\IceLakeFspPkg* folder and *Intel\IceLakePlatSamplePkg* to the *<root>\Intel* folder of the source tree. The ***Intel*** folder contained with the Kit, contains the following files:

   IceLakeFspPkg\IceLakeFspPkg.dsc.merge
   IceLakePlatSamplePkg\PlatformPkg.dsc.merge

Amend Edk2 Source Tree

1. Within the *Edk2\MdePkg* folder, the *MdePkg.dsc* file needs to be edited to allow inclusion of the SourcePoint System Trace Library. Search for the section heading *[Components]* and add two entries as shown in Figure 2 (use the *MdePkg.dsc.merge*, if required)

MdePkg.dsc

```
55  ##############################################################################################################
56
57  [Components]
58    MdePkg/Library/BaseCacheMaintenanceLib/BaseCacheMaintenanceLib.inf
59    MdePkg/Library/BaseCpuLib/BaseCpuLib.inf
60    MdePkg/Library/BaseDebugLibNull/BaseDebugLibNull.inf
61    MdePkg/Library/BaseDebugLibSerialPort/BaseDebugLibSerialPort.inf
62    MdePkg/Library/BaseDebugLibSystemTrace/BaseDebugLibSystemTrace.inf                    # ASSET System Trace Library
63    MdePkg/Library/BaseDebugLibSystemTrace/BaseDebugLibSystemTraceSmm.inf                 # ASSET System Trace Library
64    MdePkg/Library/BaseDebugPrintErrorLevelLib/BaseDebugPrintErrorLevelLib.inf
65    MdePkg/Library/BaseLib/BaseLib.inf
```

Figure 2: Modules Package Description Modification

2. Edit ***<root>\Edk2\MdePkg\Include\Library\BaseLib.h***

   Insert the support ASM files (use the BaseLib.h.merge, if required)

```
451 [Sources.X64]
452   X64/ReadGprA.nasm| MSFT
453   X64/ReadGprA.asm | MSFT
454   X64/ReadGprB.nasm| MSFT
455   X64/ReadGprB.asm | MSFT
456   X64/ReadGprC.nasm| MSFT
457   X64/ReadGprC.asm | MSFT
458   X64/ReadGprD.nasm| MSFT
459   X64/ReadGprD.asm | MSFT
460   X64/ReadRip.nasm| MSFT
461   X64/ReadRip.asm | MSFT
462   X64/ReadGprA.nasm| INTEL
463   X64/ReadGprA.asm | INTEL
464   X64/ReadGprB.nasm| INTEL
465   X64/ReadGprB.asm | INTEL
466   X64/ReadGprC.nasm| INTEL
467   X64/ReadGprC.asm | INTEL
468   X64/ReadGprD.nasm| INTEL
469   X64/ReadGprD.asm | INTEL
470   X64/ReadRip.nasm| INTEL
471   X64/ReadRip.asm | INTEL
```

```
69  [Sources.Ia32]
70    Ia32/ReadGprA.c | MSFT
71    Ia32/ReadGprB.c | MSFT
72    Ia32/ReadGprC.c | MSFT
73    Ia32/ReadGprD.c | MSFT
74    Ia32/ReadRip.c | MSFT
75    Ia32/ReadGprA.asm | INTEL
76    Ia32/ReadGprA.nasm| INTEL
77    Ia32/ReadGprB.asm | INTEL
78    Ia32/ReadGprB.nasm| INTEL
79    Ia32/ReadGprC.asm | INTEL
80    Ia32/ReadGprC.nasm| INTEL
81    Ia32/ReadGprD.asm | INTEL
82    Ia32/ReadGprD.nasm| INTEL
83    Ia32/ReadRip.asm | INTEL
84    Ia32/ReadRip.nasm| INTEL
```

3. The Trace Library can now be included in the build.

Target Specific Package

Target specific packages will differ from platform to platform. Choose the target package required for your build and edit the associated DSC file. The SourcePoint System Trace Library can be used with any EDKII source tree, however, for the library to function correctly, the target hardware must support the trace feature. This document was created using a IceLake Client target therefore, the following file needs to be edited:

<root>\Intel\IceLakePlatSamplePkg\PlatformPkg.dsc

There are multiple ways of including the SourcePoint System Trace Library in the build process. Two of these options are described in this document.

Option 1. Replace Existing Debug Library

1. Search for any occurrence of *BaseDebugLibDebugPort.inf* in the description file. A typical entry would be as shown in Figure 3

```
DebugLib|ClientCommonPkg/Library/BaseDebugLibDebugPort/BaseDebugLibDebugPort.inf
```

Figure 3: Existing Debug Library Entry.

2. Replace each entry with the SourcePoint System Trace Library as shown in Figure 4 (use *PlatformPkg.dsc.merge* if required)

```
DebugLib|MdePkg/Library/BaseDebugLibSystemTrace/BaseDebugLibSystemTrace.inf
```

Figure 4: SourcePoint System Trace Debug Library Entry.

3. Edit the file: ***<root>\Intel\IceLakeFspPkg\IceLakeFspPkg.dsc***
4. Search for any occurrence of *BaseDebugLibSerialPort.inf*. A typical entry would be as follows:

```
DebugLib|MdePkg/Library/BaseDebugLibSerialPort/BaseDebugLibSerialPort.inf
```

Figure 5: Existing FSP Debug Library Entry.

5. Replace each entry with the SourcePoint System Trace Library as shown in Figure 6 (use *IceLakeFspPkg.dsc.merge* if required):

```
DebugLib|MdePkg/Library/BaseDebugLibSystemTrace/BaseDebugLibSystemTrace.inf
```

Figure 6: SourcePoint FSP System Trace Debug Library Entry.

Option 2. Compiler Option to include Trace Library

Condition code can be used to allow compiler options for inclusion/exclusion in the build process. This has the advantage of allowing certain modules within the EDKII source tree to use the SourcePoint System Trace Library or not. A typical conditional code example is shown in Figure 7.

1. Edit the file ***<root>\Intel\IceLakePlatSamplePkg\PlatformPkg.dsc***

```
538 !if $(ASSET_SYSTEM_TRACE) == TRUE
539     DebugLib|MdePkg/Library/BaseDebugLibSystemTrace/BaseDebugLibSystemTrace.inf
540 !else
541     DebugLib|ClientCommonPkg/Library/BaseDebugLibDebugPort/BaseDebugLibDebugPort.inf
542 !endif
```

Figure 7: Conditional Compilation of Library Example.

2. Edit the file: ***<root>\Intel\IceLakeFspPkg\IceLakeFspPkg.dsc***

```
349 !if $(ASSET_SYSTEM_TRACE) == TRUE
350     DebugLib|MdePkg/Library/BaseDebugLibSystemTrace/BaseDebugLibSystemTrace.inf
351 !else
352     DebugLib|MdePkg/Library/BaseDebugLibSerialPort/BaseDebugLibSerialPort.inf
353 !endif
```

Figure 8: Conditional FSP Compilation of Library Example.

Note: If conditional code has been used to include the SourcePoint System Trace Library, modify the VAR_BUILD_FLAGS= option in the build file provided by your BIOS vendor.

In the case of IceLake, edit the file *<root>\Intel\IceLakePlatformPkg\BasicBuild.bat* (or if it exists, *<root>\Intel\IceLakePlatformPkg\BinBuild.bat*, as shown in Figure 9

```
VAR_BUILD_FLAGS= …….. -DASSET_SYSTEM_TRACE=TRUE
```

Figure 9: Compiler Build Flags.

Pre-Memory Initialization

As mrcPrintf is essentially printf, a simplified trace library can easily be utilized. However, as the pre-EFI phase has not initialized all modules, the SourcePoint System Trace Library will not be available. To circumvent this restriction, a modification to the EDK2 printf.c definition will be required see Figure 10.

```
#include <Library\DebugLibSystemTrace.h>
```

Figure 10: Include Trace Library for Pre-Memory.

This completes the section on EDKII modifications required to use the SourcePoint System Trace Library

SourcePoint Integration Overview

SourcePoint has many features that are described in more detail in the accompanying documentation of the debugger software installation (help files and SourcePoint.pdf). These include hardware descriptions of the run-control hardware provided by ASSET, and all software aspects of SourcePoint, from GUI operation to command line options and macros.

This user guide targets the Trace Hub feature of the SourcePoint environment to provide an example for use within a project for a Skylake reference platform

Configure SourcePoint for Trace Hub

Setting Trace for Trace Hub

To enable the viewing of the SourcePoint System Trace Library output via the Trace Hub, click on the trace icon in the main window. See Figure 11

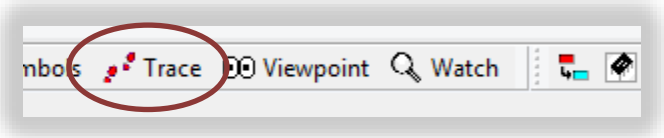


Figure 11: Trace Icon.

Depending on how SourcePoint has been used in this session, if a window popup does not display "Trace Hub", right-click on the window and select "Show SW/FW Trace" as shown in Figure 12.

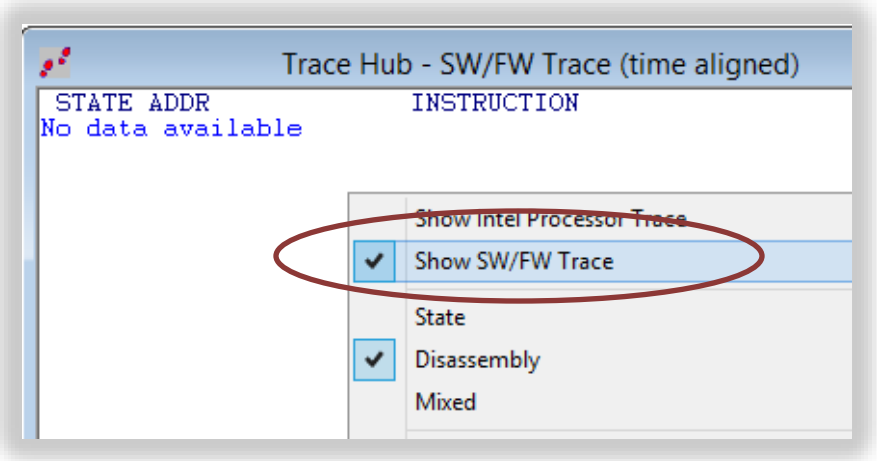


Figure 12: Show SW/FW Trace Option.

Configure Trace Hub Options

Once the Trace Hub view is displayed, the settings need to be configured to match the SourcePoint System Trace Library integrated into the build process. The Trace Hub is complex, and beyond the remit of this user guide. Please contact your local Intel® FAE if you require more information.

To change the settings, click on the “Configure” button as shown in Figure 13.

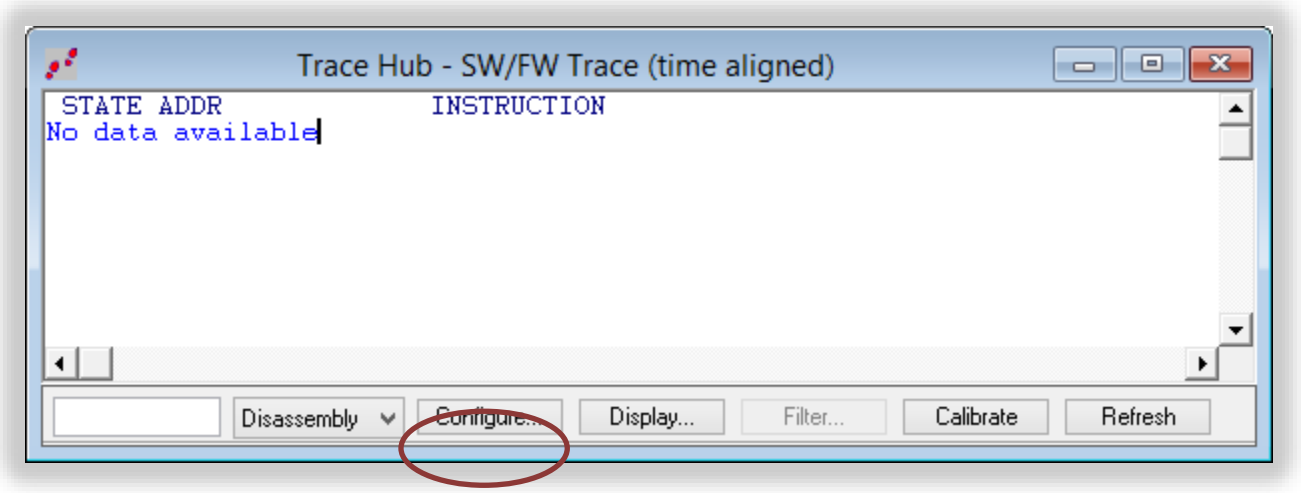


Figure 13: Configure Trace Hub Options.

Select the “Trace Hub” tab from the available options at the top of the window. This will display the settings for the Trace Hub. See Figure 14.

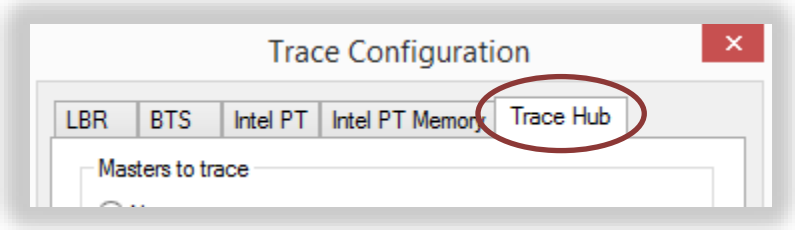


Figure 14: Trace Hub Configuration Tab.

The current Trace Hub implementation supports FW Masters 0x08 to 0x1F, SW Masters from 0x20 to 0xFF and 128 Channels per Master. This allows the Trace Hub to relay information across many different channels. These can be separated for individual processes or combined into a single channel for simplicity. The SourcePoint System Trace Library uses Master number 0x75 for communication with the Trace Hub.

Source Point can be used to take advantage of the multi-source capability of the Trace Hub. This could be used to not only display messages from the BIOS (UEFI enabled SourcePoint System Trace Library), but possible system event sources such as the Management Engine (ME). The ME uses Masters 0x10, 0x11 and 0x12.
If individual Masters are required, select “LIST” and type in the required Master numbers. To view the Master numbers and their associated names, click on the Master icon (see Figure 15).

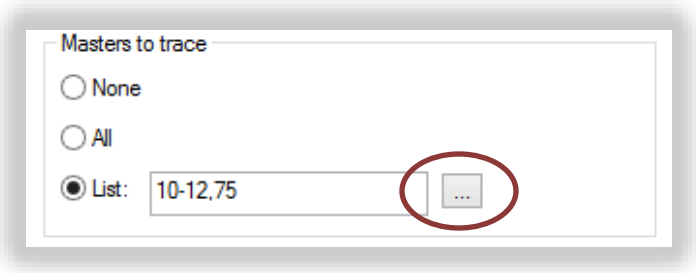


Figure 15: Selecting Masters to Trace.

Each Master that is defined in the MetaData will have an associated name against it (see Figure 16). For details of the metadata please refer to Appendix A – Example MetaData.

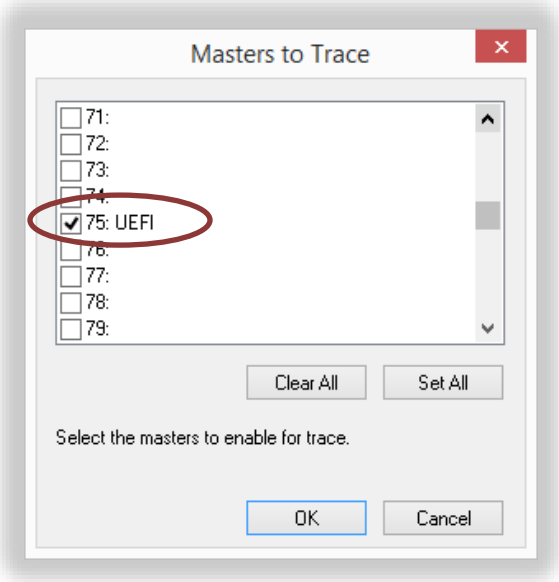


Figure 16: Viewing Master Names.

Using Trace Hub to memory requires the configuration of the buffer address. If the BIOS has configured this, then the "Use BIOS settings" can be selected. If however, you wish to override the default address, select the "Use SourcePoint settings" option and type the buffer address (see Figure 17).
Note: To use target memory, the memory controller must have been configured before attempting any Trace Hub output.

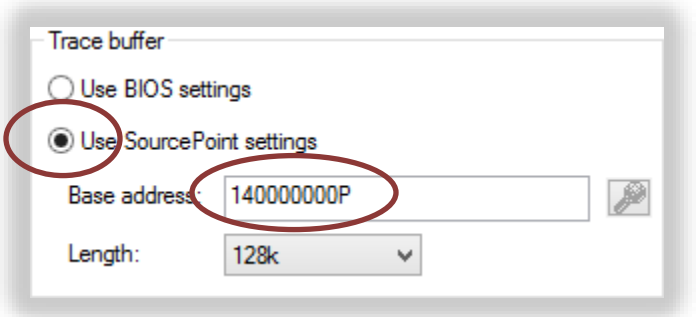


Figure 17: Trace Buffer Settings.

The Trace Hub uses the STPv2 protocol as a transport for the different trace sources. SourcePoint needs to be configured to allow the trace data to be displayed in a human readable format. By selecting the metadata file the Masters and Channels can be defined, along with display options (data type, colors, and use leading zero's etc.) To select the metadata file, click the browse icon next to the Filename box, as shown in Figure 18.

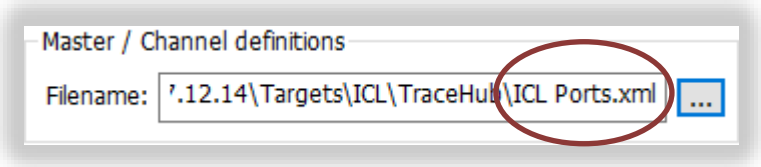


Figure 18: Metadata File Browse Icon.

Navigate to the folder location that the metadata file is located and select from the available XML file types (see Figure 19).This file is provided by ASSET upon request. However, a sample XML file is shown in Appendix A – Example MetaData.

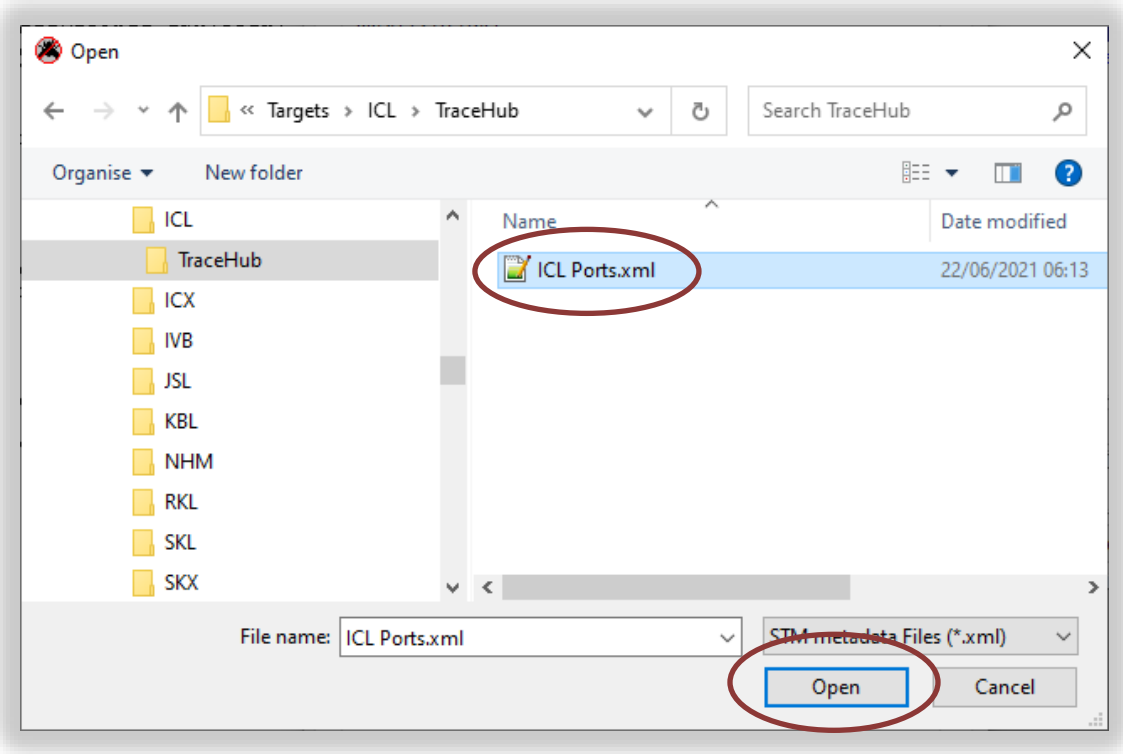


Figure 19: Metadata File Selection.

The example metadata file also includes another file to decode and display UEFI status messages. This is "included" by the <files> tag. The UEFI status decode metadata file is provided by ASSET upon request, however, an example is shown in Appendix B – Example Status MetaData

Validate Debug Output

To verify that the build process has completed successfully, and the SourcePoint System Trace Library has been included correctly, the following steps need to be performed.

1. Build the source tree with the appropriate tools (Visual Studio)

2. Program the resultant binary onto the target platform with a SPI/Flash programming tool.

3. Using the targets BIOS setup screen, enable the Trace Hub output via the BIOS settings (if not enabled by default in the build process)

4. Using SourcePoint, power on and run the target to the EFI shell

5. Using SourcePoint, load the EFI macro set (Macro->Load->"Macros\EFI\EFI.mac"). This will add macro buttons to the SourcePoint GUI.

6. Click on the button "PEIMs", wait for completion, and then click on "DXEs", and wait for completion. This will load all symbols for all modules in the build tree.

7. Use SourcePoint to add a breakpoint at DXEMAIN

8. Power cycle or reset the target platform

9. Use SourcePoint to run to the breakpoint where the memory controller has been configured. Use the breakpoint setup in (7.)

10. Follow the steps in the section: Configure Trace Hub Options

11. Run the target and stop at an appropriate location (possibly another breakpoint if desired), or the user can configure a macro to perform this step automatically.

12. SourcePoint will display a window similar to Figure 20

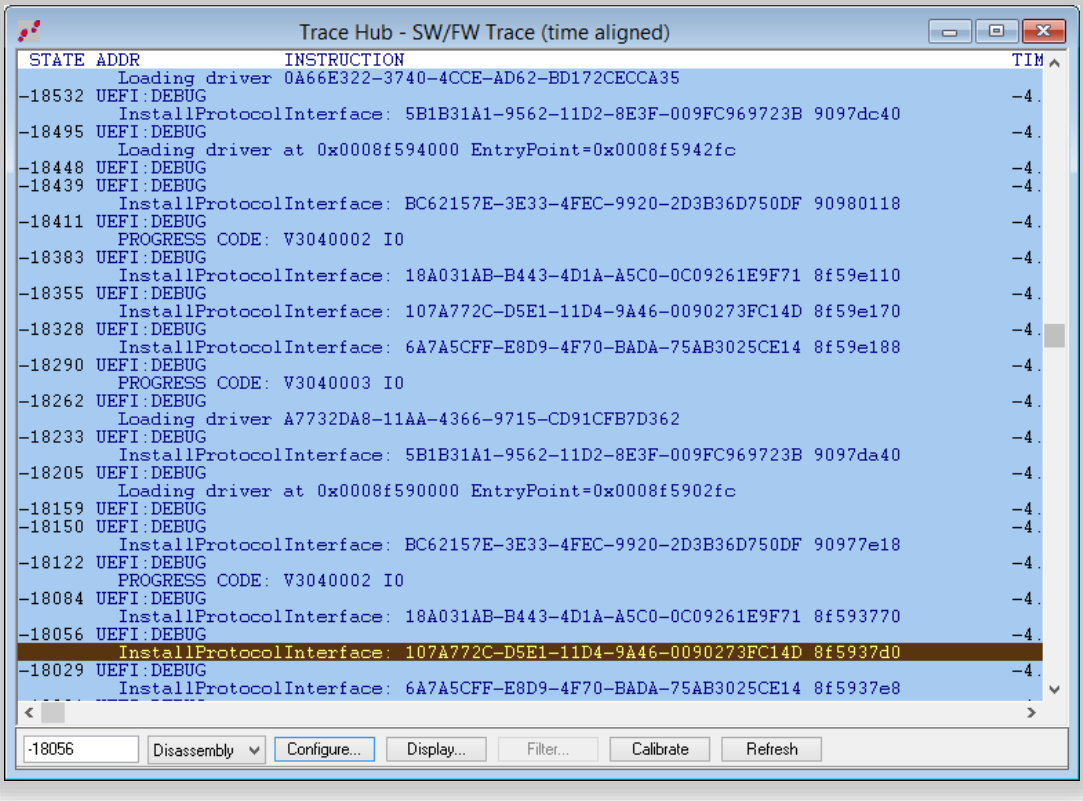


Figure 20: Trace Hub Trace Output.

Glossary

Abbreviation	Meaning
STM	System Trace Macrocell
STPv2	System Trace Protocol Version 2
SoC	System-on-a-Chip
EDKII	Embedded Development Kit
UEFI	Unified Extensible Firmware Interface

Related Documents

The following documents may be of interest in using the Trace Hub.

- System Trace Protocol (STP), v2.00.01, MIPI Alliance
  http://mipi.org/specifications/debug#STP
- STM Metadata Functional Specification, ASSET (Available upon request)
- Intel Trace Hub Developers Manual
  https://software.intel.com/sites/default/files/managed/d3/3c/intel-th-developer-manual.pdf

ASSET Contacts:

Please contact your SourcePoint tools sales representative for more information.

ASSET InterTech, Inc.
7161 Bishop Road, Suite 250
Plano, TX USA 75024
+1 888 694-6250 or +1 972 437-2800
http://asset-intertech.com

Appendix A – Example MetaData

```
<?xml version="1.0" encoding="utf-8"?>

<!--
   File:    HPRINTF.xml
   Purpose: SourcePoint sample Tool Hosted Printf metadata
-->

<hprintf_mappings>
   <mapping type="%g" value="guid"/>
</hprintf_mappings>

<masters>
   <master id="0x75" name="UEFI:HPRINTF" format="ASCII" color="0">
      <channels>
         <channel id="0x20" name="DEBUG" format="Hosted printf"/>
         <channel id="0x26" name="SMM"/>
      </channels>
   </master>
   <master id="0x76" name="UEFI:REGS" format="ASCII" color="1">
      <channels>
         <channel id="0x20" name="----RAX/RIP--- -------RBX------ -------RCX------ -------RDX------  " format="Tabular Hex" columns="4"/>
         <channel id="0x22" name="RAX" format="bit field">
            <bitfields size="64" format="RAX=%016llX">
               <bitfield start="0" size="64"/>
            </bitfields>
         </channel>
         <!--
                  <channel id="0x22" name="RAX" format="hex" size="32"/>
                  <channel id="0x23" name="RBX" format="hex" size="32"/>
         -->
         <channel id="0x21" name="RIP" format="tabular hex" columns="1"/>
         <!--
                  <channel id="0x22" name="RAX" format="tabular hex" columns="1"/>
         -->
         <channel id="0x23" name="RBX" format="tabular hex" columns="1"/>
         <channel id="0x24" name="RCX" format="tabular hex" columns="1"/>
         <channel id="0x25" name="RDX" format="tabular hex" columns="1"/>
      </channels>
   </master>
</masters>
```

Appendix B – Example Status MetaData

```
<?xml version="1.0" encoding="utf-8"?>

<!--
    File:    UefiStatus.xml
    Purpose: UEFI status strings
-->

 <messagetables>
     <messagetable name="UEFI Status">
         <message id="0x00000000" format="Success"/>
         <message id="0x00000001" format="Warning Unknown Glyph"/>
         <message id="0x00000002" format="Warning Delete Failure"/>
         <message id="0x00000003" format="Warning Write Failure"/>
         <message id="0x00000004" format="Warning Buffer Too Small"/>
         <message id="0x80000001" format="Load Error"/>
         <message id="0x80000002" format="Invalid Parameter"/>
         <message id="0x80000003" format="Unsupported"/>
         <message id="0x80000004" format="Bad Buffer Size"/>
         <message id="0x80000005" format="Buffer Too Small"/>
         <message id="0x80000006" format="Not Ready"/>
         <message id="0x80000007" format="Device Error"/>
         <message id="0x80000008" format="Write Protected"/>
         <message id="0x80000009" format="Out of Resources"/>
         <message id="0x8000000a" format="Volume Corrupt"/>
         <message id="0x8000000b" format="Volume Full"/>
         <message id="0x8000000c" format="No Media"/>
         <message id="0x8000000d" format="Media changed"/>
         <message id="0x8000000e" format="Not Found"/>
         <message id="0x8000000f" format="Access Denied"/>
         <message id="0x80000010" format="No Response"/>
         <message id="0x80000011" format="No mapping"/>
         <message id="0x80000012" format="Time out"/>
         <message id="0x80000013" format="Not started"/>
         <message id="0x80000014" format="Already started"/>
         <message id="0x80000015" format="Aborted"/>
         <message id="0x80000016" format="ICMP Error"/>
         <message id="0x80000017" format="TFTP Error"/>
         <message id="0x80000018" format="Protocol Error"/>
         <message id="0x80000019" format="Incompatible Version"/>
         <message id="0x8000001a" format="Security Violation"/>
         <message id="0x8000001b" format="CRC Error"/>
         <message id="0x8000001c" format="End of Media"/>
         <message id="0x8000001f" format="End of File"/>
     </messagetable>
</messagetables>
```