American Megatrends

Trace Hub Debugger

Trace Hub Debugger

Document Revision 1.14

July 31, 2018

NDA Required (NDA)

Legal

Table of Contents

Document Information **5**
Purpose 5
Audience 5
Overview **6**
System Requirements **7**
Software 7
Host software requirements 7
Hardware 7
Supported Platforms 8
Installation **9**
Host side 9
Adding AMI Value Add Module and Debugger Plugins 9
Installing Trace Hub Collaterals 10
Installing Device Driver 10
Installing Intel© CCA Device Driver 10
Installing USB 3.0 Debug Cable Driver 11
Target Side 12
Installing Target Module 12
Using Trace Hub in VisualeBios **13**
Setup Instructions: 14
Selecting Platform Configuration 14
Updating Collaterals 15
Trace Hub Debugger (Intel platform specific) Features **17**
Starting a Debug Session 18
Stopping a Debug Session 19
Working with Registers 19
Working with Disassembly 20
Working with Call Stack View 21
Working with Trace View 22
Working with Python Console/ User scripts 23
Execution Control 24
Stepping Controls 24
Breakpoints - Setting & Usage 26
SMM Debugging 28

Perform Basic DCI Functional Test 30
General Notes 32
Known Issues 33
Limitations 34
Troubleshooting 35
The Intel® CCA Hardware LED Indicators 36

Document Information

Purpose

This document provides information about Trace hub feature integrated with VisualeBios (VeB).

Audience

The intended audiences are BIOS developers, Generic Chipset Porting Engineers, OEM Porting Engineers, and AMI OEM Customers.

Overview

Intel platforms From Skylake and later generation Intel CPU and platform supports hardware debugging and Trace hub infrastructure via USB 3 port.

Debug functionality controlled by CPU and Trace functionality controlled by PCH.

Need Hardware device called INTEL SVT Closed chassis adapter to be connected USB 3 Ports and target. Link: https://designintools.intel.com/product_p/itpxdpsvt.htm

Future platforms from Cannon Lake will include this CCA device in the platform itself and just USB 3 debug cable is enough to perform this operation.

System Requirements

Software

Host software requirements

- PC Host with one of the following version of Windows OS 7\8\10
- VisualeBios (VeB). (BuildTools_32)
- Java Runtime Environment (JRE) x86 - [Link]
- Microsoft Visual C++ 2010 Redistributable x86 - [Link]
- Microsoft Visual C++ 2013 Redistributable x86 - [Link]
- Intel Trace hub Driver (Driver available for download internally from BiosInternal repository: $/AptioV/Binary/Modules/Debugger/Intel_CCA_Device_Driver.7z)

Hardware

DCI supports two hardware configurations (depending on the Intel® platforms):

- Intel® DCI-OOB (DCI over an Intel® SVT Closed Chassis Adapter [Intel® CCA])
- Intel® DCI-USB (DCI over a USB3.0 Debug Cable)

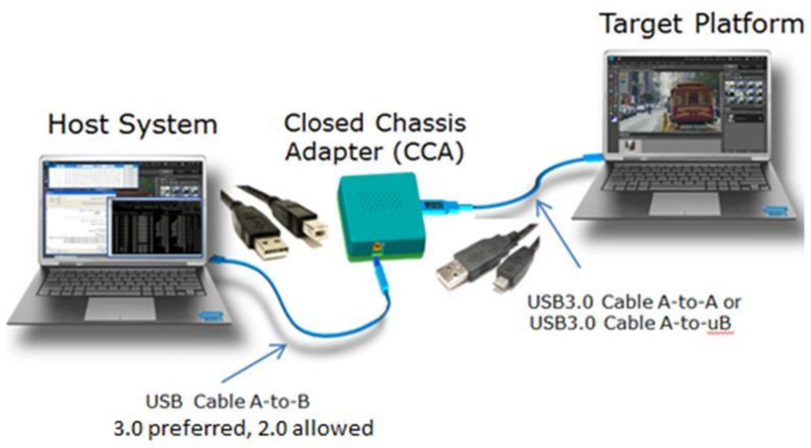

Target Platform
Host System
Closed Chassis Adapter (CCA)
USB3.0 Cable A-to-A or USB3.0 Cable A-to-uB
USB Cable A-to-B 3.0 preferred, 2.0 allowed

Figure 1: Hardware Configuration for Intel ® DCI-OOB with Intel® CCA

The Intel® CCA requires USB3 A-to-B cable to connect the host to the Intel® CCA, and a 6" A-to-A, A-to-uAB or A-to-C USB3 cable to connect the Intel® CCA to the target.

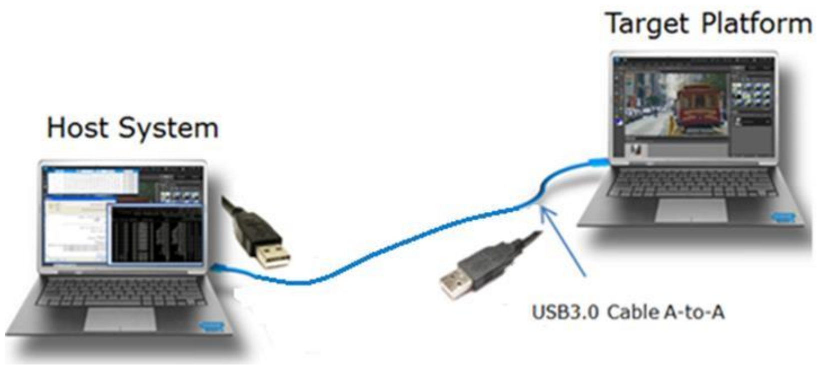


Figure 2: Hardware Configuration for USB3 Hosting DCI with USB3.0 Cable

USB-hosted DCI requires a USB3 Debug Cable (where the VBUS wires are isolated) to connect the debug host to the target. Using standard A to A USB3 cables can keep the target from transitioning through all power states correctly

Supported Platforms

The following are the supported platforms where Trace hub can be used

Table 1: Supported Platform as of July 2018

Platform	Processor	PCH	Debug	Trace	Connection Methods	Comments
Skylake	Any		✓	✓	OOB	DT RVP 8 Not supported
Kaby Lake	Any	KBP, SPT	✓	✓	OOB	
Coffee Lake	CFL-H, CFL-S, CFL-U, CFL-Y	SPT, KBP, CNP	✓	✓	OOB, USB3	Trace messages are not available when using USB 3.0 Connection
Canon Lake	CNL-U	SPT, CNP	✓	✓	OOB	
Purley	Skylake Server	LBG	✓	✓	OOB	
Purley Refresh	Skylake Server	LGB	✓		OOB	
Bakerville	Skylake Server	LGB	✓		OOB	
WolfPass	Skylake Server	LGB	✓		OOB	

Note: Connection to all platforms require NDA collaterals

Installation

Host side

Download VEB label Buildtools_32 or later and use this VEB to download Debugger module, and open project.

Download Debugger module (AptioV/trunk/AptioV/Binary/Modules/Debugger Labeled Debugger_41), reopen the project, AptioV debugger and trace hub debugger will be installed automatically.

Restart the VEB, you will able to see the “DCI Debug” and “Debug” menu. The “DCI Debug” is the trace hub debugger.

Adding AMI Value Add Module and Debugger Plugins

Ami Trace Hub debugger requires Host plugins to be installed in the VisualeBios (VeB) location. Adding an optional target module to the project allows users to enable AMI Value Adds.

The steps to install the Host plugins and the AMI value Add module are as follows.

- Launch VisualeBios (VeB) and open the project to be debugged
- In VEB select the context menu option “Add Component” from Module Explorer View
- Select button “Source Control” and Browse to folder `"SS:AptioV;$/AptioV/Binary/Modules/Debugger;"` and Click OK
- Select the label Debugger_41 or later
- Check the eModule components AMITraceHubDebugger and Debugger
- Click Select to Install the components to the project
- Finally reload project to install host plugins

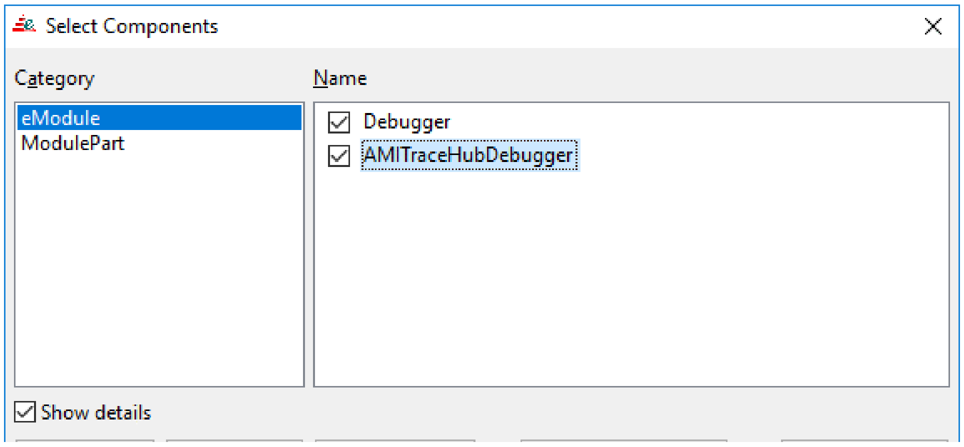


Figure 3: AMITraceHubDebugger and Debugger component selection

Installing Trace Hub Collaterals

Ami Trace Hub debugger requires special collaterals to communicate with the target platform. The collaterals are made available to the AMI customer once a three-way NDA agreement is signed.

Follow the steps below to install the collaterals once they have been made available.

- Launch VeB and open the project to be debugged
- In VEB select the context menu option "Add Component" from Module Explorer View
- Select button "Have Disk…", Browse to the folder that contains the collaterals and Click OK
- Check the checkbox for ModulePart "NDA Collaterals"
- Click Select to add the collaterals to the project

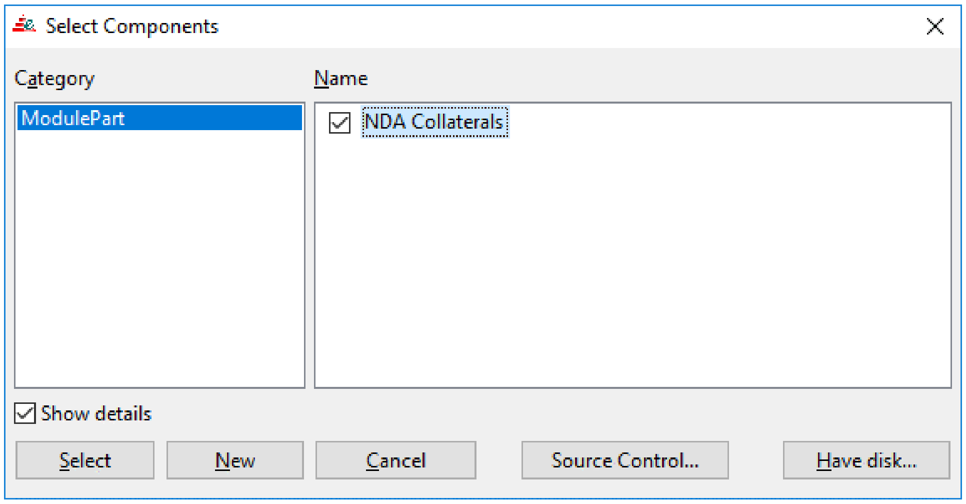


Figure 4: Installing Trace Hub Collaterals

Pl. note that when installing Collateral on VeB labeled BuildTools_32, the automatic installation will not be done. After the collateral component is installed to the project, the user will need to open the DCI configuration dialog found under DCI Debug->Configuration.

Installing Device Driver

Communicating with the target platform requires the proper host drivers be installed. The driver is named "Intel_CCA_Device _Driver.7z" and can be found in the NDA Collaterals ModulePart component.

To install the Device Driver please follow the below steps depending on the transport used.

Installing Intel© CCA Device Driver

- Extract "Intel_CCA_Device_Driver.7z" to a local folder
- Connect the CCA device to host system to either the USB 3.0 (preferred) or USB 2.0 port
  - Ensure CCA port marked "HOST" is connected to the host system

- Open Windows Device Manager and look for entry “Intel DCI Transports (USB-Based)”

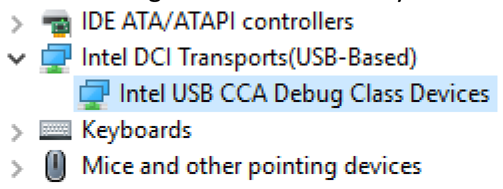

IDE ATA/ATAPI controllers
Intel DCI Transports(USB-Based)
Intel USB CCA Debug Class Devices
Keyboards
Mice and other pointing devices

Figure 5: Device Driver view with Intel© CCA Device Driver installed

If found, the necessary drivers are already installed

- If not found, look for entry “Other Devices” and expand that node to select “Closed Chassis Adapter”

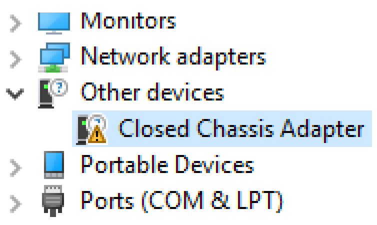


Figure 6: Device Manager view with Driver Not Installed

- Rt-Click on the node and select the context menu option “Update Driver”
- Select “Browse my computer for driver software” and browse to the location where the driver was extracted
- Click on Next to finish the installation of the driver

Installing USB 3.0 Debug Cable Driver

- Extract “Intel_CCA_Device_Driver.7z” to a local folder
- Connect the USB 3.0 debug cable to the USB 3.0 port of the host system and the target system
- Power On the target system
- Open Windows Device Manager and look for entry “Intel DCI Transports (USB-Based)”

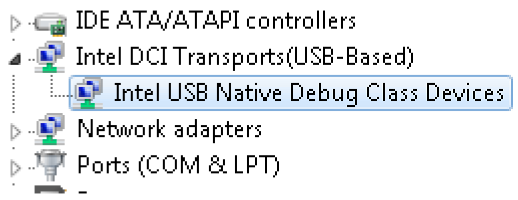

IDE ATA/ATAPI controllers
Intel DCI Transports(USB-Based)
Intel USB Native Debug Class Devices
Network adapters
Ports (COM & LPT)

Figure 7: USB Debug Cable View with Driver Installed

If found, the necessary drivers are already installed

- If not found, look for entry “Other Devices” and expand that node to select “Unknown device”

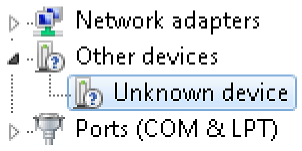


Figure 8: USB debug cable view without Driver Installed

- Rt-Click on the node and select the context menu option “Update Driver”
- Select “Browse my computer for driver software” and browse to the location where the driver was extracted
- Click on Next to finish the installation of the driver

Target Side

Usually when using a hardware debugger, target side footprint is not required. But this provides the user only the basic debugging features. Also, some features like loading of sources are only available after intensive operations leading to slowness and increased resource usage.

To distinguish itself AMI makes available a target module which provides Value Adds that augment the basic features that are available by default.

The AMI value add provides the following additional features

- Loaded Image View
- Support for breakpoint in code using __debugbreak()
- Detailed Callstack Information
- Faster loading of debug source
- SMM debugging using BSP

Installing Target Module

Use VisualeBios to add optional AMI module from AptioV repository. The “**AMITraceHubDebugger**” component can be downloaded from the Aptio V source control location **$/AptioV/Binary/Modules/Debugger** at label “Debugger_41” or later.

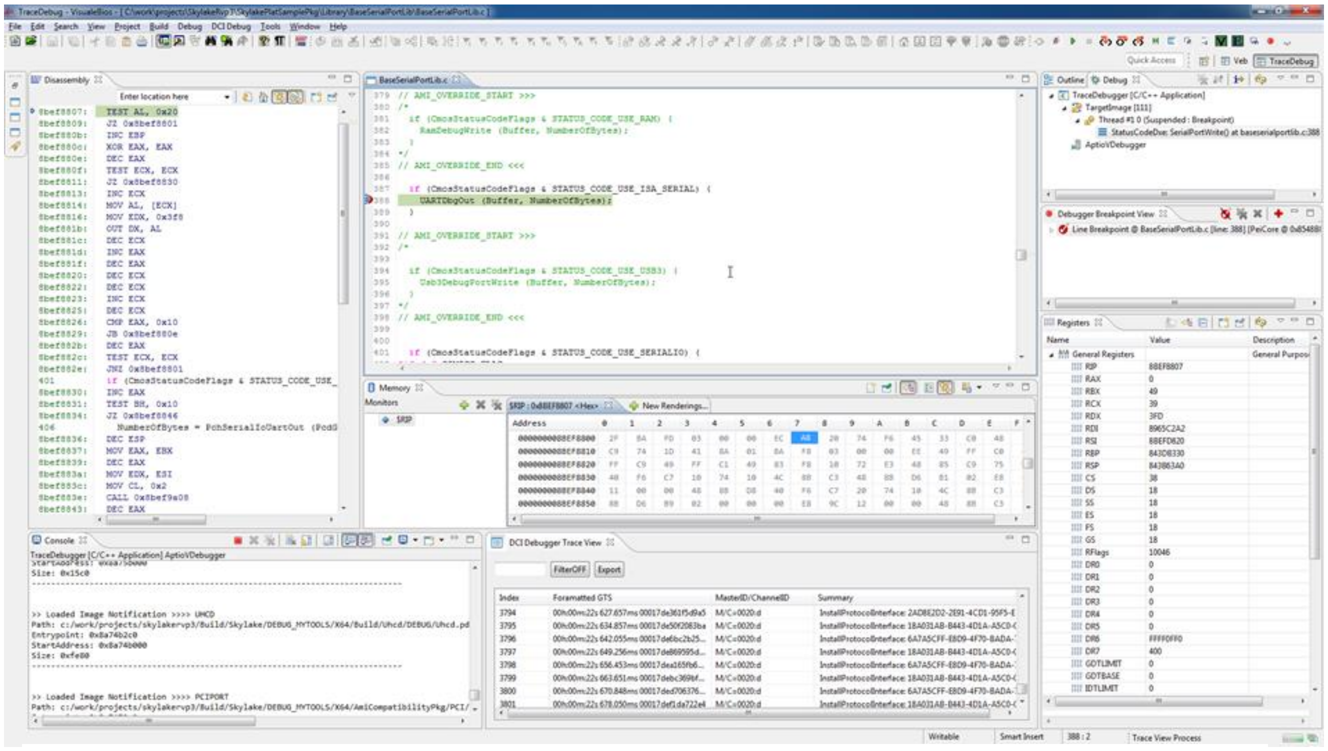
Enable the SDL token object “AMITraceHubDebugger_SUPPORT” and build the project.
The following value adds are available to the user once the target module is added.

Using Trace Hub in VisualeBios

Users can debug a target using the Intel Trace hub device from within VisualeBios (VeB). The users presented with a new perspective to view debug information. The following image depicts the Trace hub perspective a user will see.

If Target module is not present, User needs to select “Auto Load sources” from menu to load sources.

Currently Trace hub debugger support Connect to the target, disconnect from the target, halt the target, continue, reset the target, read register, and read memory. They can be accessed from the menu “DCI Debug”.



```
8bef8807:   TEST AL, 0x20
8bef8809:   JZ 0x8bef8801
8bef880b:   INC ESP
8bef880c:   XOR EAX, EAX
8bef880e:   DEC EAX
8bef880f:   TEST ECX, ECX
8bef8811:   JZ 0x8bef8830
8bef8813:   INC ECX
8bef8814:   MOV AL, [ECX]
8bef8816:   MOV EDX, 0x3f8
8bef881b:   OUT DX, AL
8bef881c:   DEC ECX
8bef881d:   INC EAX
8bef881f:   DEC EAX
8bef8820:   DEC ECX
8bef8822:   DEC ECX
8bef8823:   INC ECX
8bef8825:   DEC ECX
8bef8826:   CMP EAX, 0x10
8bef8829:   JB 0x8bef880e
8bef882b:   DEC EAX
8bef882c:   TEST ECX, ECX
8bef882e:   JNZ 0x8bef8801
401         if (CmosStatusCodeFlags & STATUS_CODE_USE_
8bef8830:   INC EAX
8bef8831:   TEST BH, 0x10
8bef8834:   JZ 0x8bef8846
406           NumberOfBytes = PchSerialIoUartOut (PcdG
8bef8836:   DEC ESP
8bef8837:   MOV EAX, EBX
8bef8839:   DEC EAX
8bef883a:   MOV EDX, ESI
8bef883c:   MOV CL, 0x2
8bef883e:   CALL 0x8bef9a08
8bef8843:   DEC EAX
```

```c
// AMI_OVERRIDE_START >>>
/*
  if (CmosStatusCodeFlags & STATUS_CODE_USE_RAM) {
    RamDebugWrite (Buffer, NumberOfBytes);
  }
*/
// AMI_OVERRIDE_END <<<

  if (CmosStatusCodeFlags & STATUS_CODE_USE_ISA_SERIAL) {
    UARTDbgOut (Buffer, NumberOfBytes);
  }

// AMI_OVERRIDE_START >>>
/*

  if (CmosStatusCodeFlags & STATUS_CODE_USE_USB3) {
    Usb3DebugPortWrite (Buffer, NumberOfBytes);
  }
*/
// AMI_OVERRIDE_END <<<


  if (CmosStatusCodeFlags & STATUS_CODE_USE_SERIALIO) {
```

```
TraceDebugger [C/C++ Application]
  TargetImage [111]
    Thread #1 0 (Suspended : Breakpoint)
      StatusCodeDxe: SerialPortWrite() at baseserialportlib.c:388
  AptioVDebugger
```

```
Line Breakpoint @ BaseSerialPortLib.c [line: 388] [PeiCore @ 0x85488
```

```
Address           0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
000000008BEF8800  2F 8A FD 03 00 00 EC A8 20 74 F6 45 33 C0 48
000000008BEF8810  C9 74 1D 41 8A 01 BA F8 03 00 00 EE 49 FF C0
000000008BEF8820  FF C9 49 FF C1 49 83 F8 10 72 E3 48 85 C9 75
000000008BEF8830  48 F6 C7 10 74 10 4C 8B C3 48 8B D6 B1 02 E8
000000008BEF8840  11 00 00 48 8B D8 40 F6 C7 20 74 10 4C 8B C3
000000008BEF8850  8B D6 89 02 00 00 00 E8 9C 12 00 00 48 8B C3
```

```
TraceDebugger [C/C++ Application] AptioVDebugger
StartAddress: 0x8a750000
Size: 0x15c0
--------------------------------------------------------------------------------


>> Loaded Image Notification >>>> UHCD
Path: c:/work/projects/skylakervp3/Build/Skylake/DEBUG_MYTOOLS/X64/Build/Uhcd/DEBUG/Uhcd.pd
Entrypoint: 0x8a74b2c0
StartAddress: 0x8a74b000
Size: 0xfe80
--------------------------------------------------------------------------------


>> Loaded Image Notification >>>> PCIPORT
Path: c:/work/projects/skylakervp3/Build/Skylake/DEBUG_MYTOOLS/X64/AmiCompatibilityPkg/PCI/
```

Index	Foramatted GTS	MasterID/ChannelID	Summary
3794	00h:00m:22s 627.657ms 00017de361f5d9a5	M/C=0020:d	InstallProtocolInterface: 2AD8E2D2-2E91-4CD1-95F5-E
3795	00h:00m:22s 634.857ms 00017de50f2083ba	M/C=0020:d	InstallProtocolInterface: 18A031AB-B443-4D1A-A5C0-(
3796	00h:00m:22s 642.055ms 00017de6bc2b25...	M/C=0020:d	InstallProtocolInterface: 6A7A5CFF-E8D9-4F70-BADA-
3797	00h:00m:22s 649.256ms 00017de869595d...	M/C=0020:d	InstallProtocolInterface: 18A031AB-B443-4D1A-A5C0-(
3798	00h:00m:22s 656.453ms 00017dea165fb6...	M/C=0020:d	InstallProtocolInterface: 6A7A5CFF-E8D9-4F70-BADA-
3799	00h:00m:22s 663.651ms 00017debc369bf...	M/C=0020:d	InstallProtocolInterface: 18A031AB-B443-4D1A-A5C0-(
3800	00h:00m:22s 670.848ms 00017ded706376...	M/C=0020:d	InstallProtocolInterface: 6A7A5CFF-E8D9-4F70-BADA-
3801	00h:00m:22s 678.050ms 00017def1da722e4	M/C=0020:d	InstallProtocolInterface: 18A031AB-B443-4D1A-A5C0-(

Name	Value	Description
General Registers		General Purpos
RIP	8BEF8807	
RAX	0	
RBX	49	
RCX	39	
RDX	3FD	
RDI	8965C2A2	
RSI	8BEFD820	
RBP	843D8330	
RSP	843B63A0	
CS	38	
DS	18	
SS	18	
ES	18	
FS	18	
GS	18	
RFlags	10046	
DR0	0	
DR1	0	
DR2	0	
DR3	0	
DR4	0	
DR5	0	
DR6	FFFF0FF0	
DR7	400	
GDTLIMIT	0	
GDTBASE	0	
IDTLIMIT	0	

Figure 9: Trace Hub Debugger Views

Setup Instructions:

For any AptioV project that supports Trace hub, to enable trace hub support, build the project with token TRACE_HUB_STATUS_CODE_SUPPORT enabled. This SDL token can be found in the generic module “IntelClientCommonPkg”.

Note: the token may be cloned in another module and turned off.

Flash the generated firmware image and boot the target, and enter into the BIOS setup and enable the following setup questions.

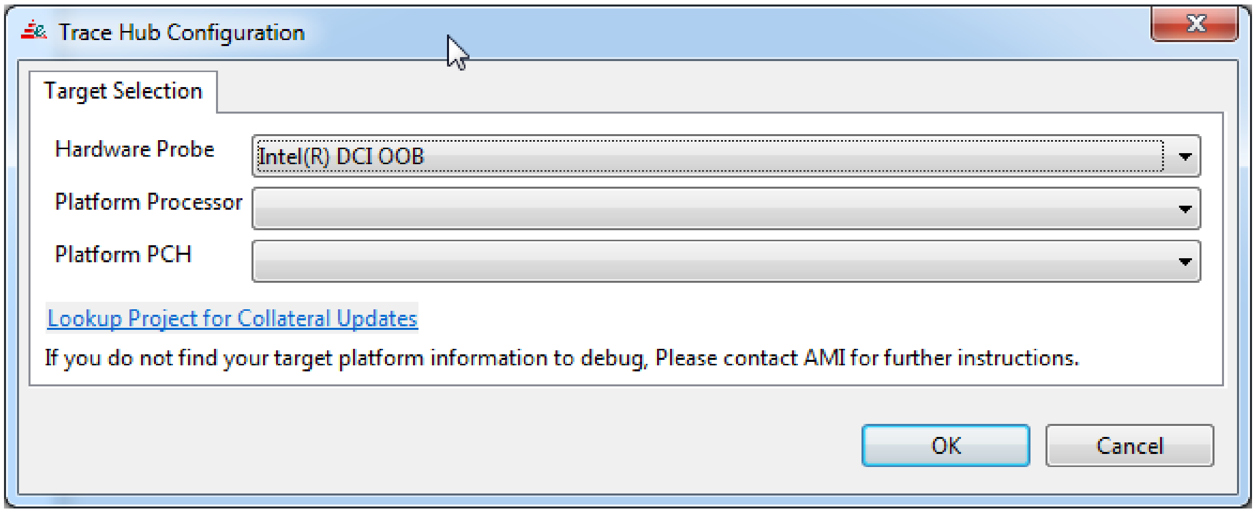
Setup Question	Value
PCH-IO Configuration->Trace Hub Configuration Menu->Trace Hub Enable Mode	***Host Debugger***
PCH-IO Configuration->DCI Enable (HDCIEN)	**Enabled**
CPU Configuration->Debug Interface	**Enabled**
CPU Configuration->Direct Connect Interface	**Enabled**

Then connect the target and host by using the INTEL SVT Closed chassis adapter, for the target side, need to use the USB 3 debug port. For the host side, USB2 or USB3 connection are fine.

Selecting Platform Configuration

Before connecting, need to select the correct configuration file, to connect to the target. Use menu “**DCI Debug**-> **Configuration**” or by clicking on menu icon

You will see the following dialog.



Trace Hub Configuration

Target Selection

Hardware Probe: Intel(R) DCI OOB

Platform Processor:

Platform PCH:

Lookup Project for Collateral Updates

If you do not find your target platform information to debug, Please contact AMI for further instructions.

Figure 10: Trace hub Configuration

NOTE: If the platform is not available as part of the default configuration collaterals. Download and Execute Install.exe from BiosInternal: $/AptioV/Binary/Modules/Debugger/ to update the collateral list.

- The configuration dialog will display the hardware probes available for debugging
- Select the connection type as either “**Intel(R) DCI OOB**” or “**Intel(R) DCI USB 3.x Debug Class**”.
- When you use the INTEL SVT Closed chassis adapter, please always select “**Intel(R) DCI OOB**”.
- Select the Processor Information from the list to enable debug of the platform. For example, Skylake – SKL_SPT-DCI_OOB
- The Platform PCH is now populated for selection
- Select the target PCH information for configuration from the list. For example, Skylake CPU / SunrisePoint PCH-LP C1
- If target is running, use menu “**DCI Debug**-> **Connect**”. The INTEL SVT Closed chassis adapter should be starting to connect to the target, after it connect the INTEL SVT Closed chassis adapter target light should be green. If there are trace message comes, you should be able to see it in the “Trace Message view”.
- Use menu DCI Debug-> halt. The target should be halt. You can read memory, read register, and see the disassemble code.

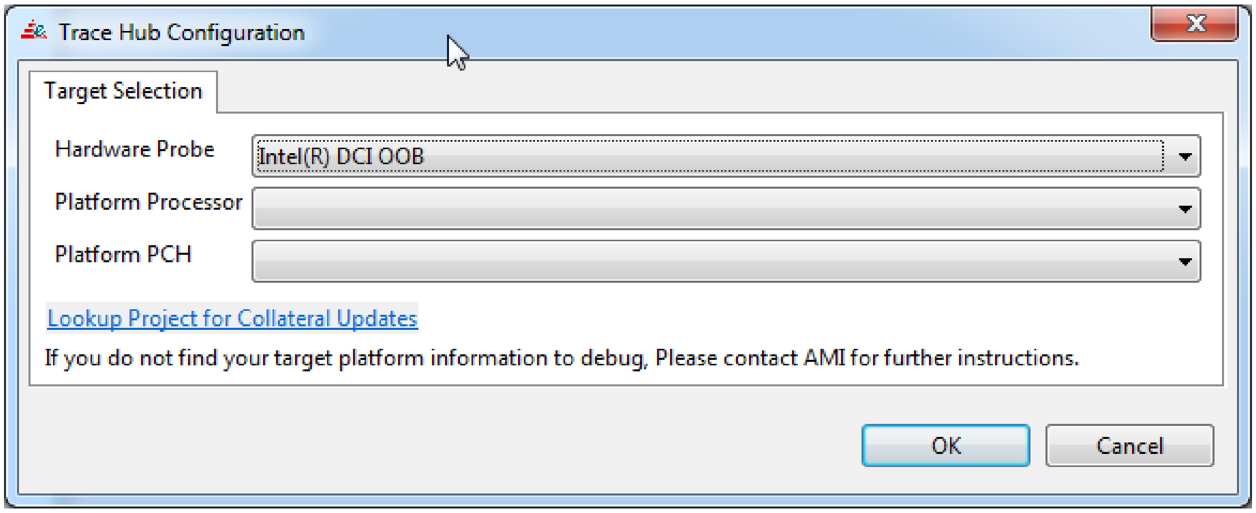
Currently Trace hub debugger has following support available for the debug function and full functionality will be added in future.

Updating Collaterals

When new collaterals are available, follow the process to add the updated NDA Collateral component to the project. VeB will automatically identify if new version of the collaterals are available and install them to the project.

If user is using VeB labeled BuildTools_32 then the collaterals are not automatically installed. The user will need to perform additional steps to install the collaterals. User is required to open the Trace Hub Configuration from the menu “DCI Debug->Configuration” or by clicking on menu icon

Once the Trace Hub Configuration dialog is opened the user can then click on the link “Lookup Project for Collateral Updates”. Clicking on the link allows Debugger to scan the project for updated collaterals and if found will automatically install them. Reopen the dialog to see the updated configurations.



Hardware Probe: Intel(R) DCI OOB

Platform Processor:

Platform PCH:

Lookup Project for Collateral Updates

If you do not find your target platform information to debug, Please contact AMI for further instructions.

Figure 11: Updating Collaterals

Trace Hub Debugger (Intel platform specific) Features

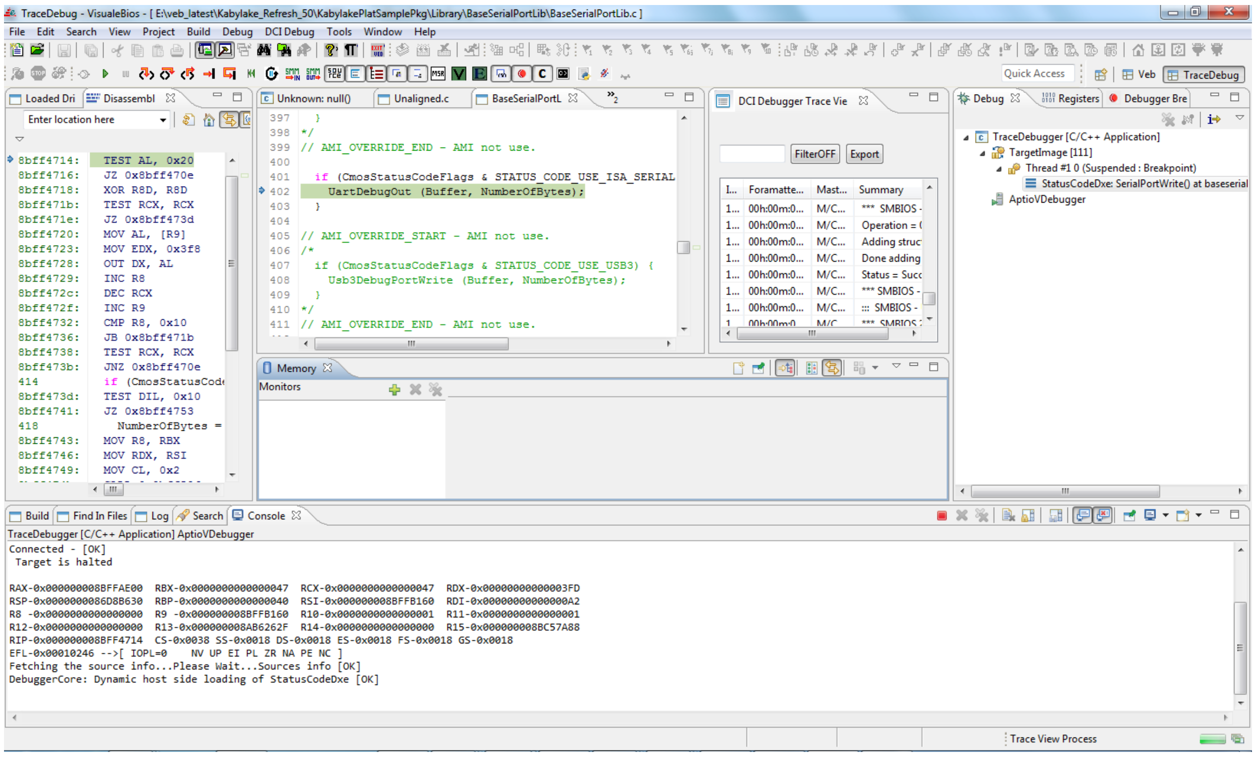
Features Technology / Feature	Supported	Comments
Real Mode Debugging (disassembly)	✓	
x32 and x64 mode debugging	✓	
CCA (OOB) Debugging Support	✓	Connection between Target and host using Intel SVT hardware
USB 3 (DbC) Debug Class Debugging	✓	Connection between Target and host using USB 3 Debug Cable.
Asynchronous Break	✓	
Asynchronous Reboot	✓	
Resume / Go	✓	
Source level debugging - Source view with current line highlighted. - Disassembly View - Mixed View support for source and disassembly	✓	
Source level debugging - Stepping into - Step over Operation - Step Out Operation - Step return / Step Out - Run to Line - Set Next statement	✓	
Call Stack Display	✓	
Debugger Toolbar	✓	
Local Variable view	✓	
Memory - View and Edit Memory content	✓	
Multiple address monitoring	✓	Open multiple memory views at a time
Breakpoint	✓	Set breakpoints on IO, Memory access, Execution

Features Technology / Feature	Supported	Comments
– Hardware – Software – Reset Break		
Register view	✓	
Console logs	✓	Log Messages from Host Application
View Loaded Image Notifications	✓	
Toggle Breakpoints with source view	✓	Set Breakpoint by Double click on source Line
Trace Messages View	✓	Time log, Export, Search and filtering.
ITP (Python) Script Support	✓	Support of ITP console and Script from VEB.

Starting a Debug Session

Follow the steps outlined below to start a new Debug Session,

1. Build the BIOS project, flash the BIOS Image on the Target platform and set the Setup tokens (Refer Setup Instructions)
2. Connect the target and host by using the INTEL SVT Closed chassis adapter.
3. Launch VisualeBios (with AptioVDebugger installed).
4. Select the Target configuration (Refer Selecting Platform Configuration)
5. Now start a Debugging Session by selecting "DCI Debug→Connect" or select the Connect [ ] from Toolbar (Target should be running)
6. Now target and host will be connected.



```
8bff4714:   TEST AL, 0x20
8bff4716:   JZ 0x8bff470e
8bff4718:   XOR R8D, R8D
8bff471b:   TEST RCX, RCX
8bff471e:   JZ 0x8bff473d
8bff4720:   MOV AL, [R9]
8bff4723:   MOV EDX, 0x3f8
8bff4728:   OUT DX, AL
8bff4729:   INC R8
8bff472c:   DEC RCX
8bff472f:   INC R9
8bff4732:   CMP R8, 0x10
8bff4736:   JB 0x8bff471b
8bff4738:   TEST RCX, RCX
8bff473b:   JNZ 0x8bff470e
414         if (CmosStatusCode
8bff473d:   TEST DIL, 0x10
8bff4741:   JZ 0x8bff4753
418           NumberOfBytes =
8bff4743:   MOV R8, RBX
8bff4746:   MOV RDX, RSI
8bff4749:   MOV CL, 0x2
```

```
  }
*/
// AMI_OVERRIDE_END - AMI not use.

  if (CmosStatusCodeFlags & STATUS_CODE_USE_ISA_SERIAL
    UartDebugOut (Buffer, NumberOfBytes);
  }

// AMI_OVERRIDE_START - AMI not use.
/*
  if (CmosStatusCodeFlags & STATUS_CODE_USE_USB3) {
    Usb3DebugPortWrite (Buffer, NumberOfBytes);
  }
*/
// AMI_OVERRIDE_END - AMI not use.
```

I...	Foramatte...	Mast...	Summary
1...	00h:00m:0...	M/C...	*** SMBIOS -
1...	00h:00m:0...	M/C...	Operation = (
1...	00h:00m:0...	M/C...	Adding struc
1...	00h:00m:0...	M/C...	Done adding
1...	00h:00m:0...	M/C...	Status = Succ
1...	00h:00m:0...	M/C...	*** SMBIOS -
1...	00h:00m:0...	M/C...	::: SMBIOS -
1	00h:00m:0	M/C	*** SMBIOS

```
TraceDebugger [C/C++ Application]
  TargetImage [111]
    Thread #1 0 (Suspended : Breakpoint)
      StatusCodeDxe: SerialPortWrite() at baseserial
  AptioVDebugger
```

```
TraceDebugger [C/C++ Application] AptioVDebugger
Connected - [OK]
 Target is halted

RAX-0x000000008BFFAE00  RBX-0x0000000000000047  RCX-0x0000000000000047  RDX-0x00000000000003FD
RSP-0x0000000086D8B630  RBP-0x0000000000000040  RSI-0x000000008BFFB160  RDI-0x00000000000000A2
R8 -0x0000000000000000  R9 -0x000000008BFFB160  R10-0x0000000000000001  R11-0x0000000000000001
R12-0x0000000000000000  R13-0x000000008AB6262F  R14-0x0000000000000000  R15-0x000000008BC57A88
RIP-0x000000008BFF4714  CS-0x0038 SS-0x0018 DS-0x0018 ES-0x0018 FS-0x0018 GS-0x0018
EFL-0x00010246 -->[ IOPL=0    NV UP EI PL ZR NA PE NC ]
Fetching the source info...Please Wait...Sources info [OK]
DebuggerCore: Dynamic host side loading of StatusCodeDxe [OK]
```

Figure 12: AMI Trace Hub Debugger

Stopping a Debug Session

To stop the target's Debugging Session,

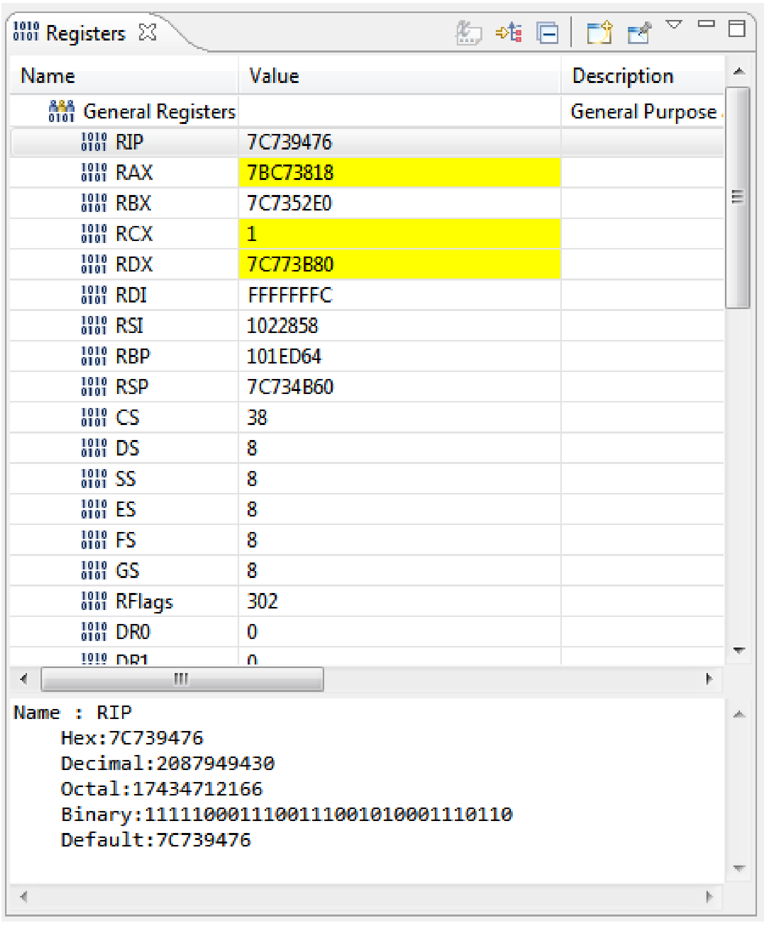
1. Click "DCI Debug→Disconnect" on the Debug menu or select the Toolbar Icon [ ]
This action enables you to end a previous debug session and start a New Debug Session if required.

Working with Registers

Users can Open or Close the Registers View using:

- VeBMenu: Debug →View →Registers
- Or using Debugger Toolbar Icon [AX]

The Registers view will only be updated when the Target is in a halted state. The Registers window contains two columns. Name - Register Name & Value - the current value of the Register in Hexadecimal.



Name	Value	Description
General Registers		General Purpose
RIP	7C739476	
RAX	7BC73818	
RBX	7C7352E0	
RCX	1	
RDX	7C773B80	
RDI	FFFFFFFC	
RSI	1022858	
RBP	101ED64	
RSP	7C734B60	
CS	38	
DS	8	
SS	8	
ES	8	
FS	8	
GS	8	
RFlags	302	
DR0	0	
DR1	0	

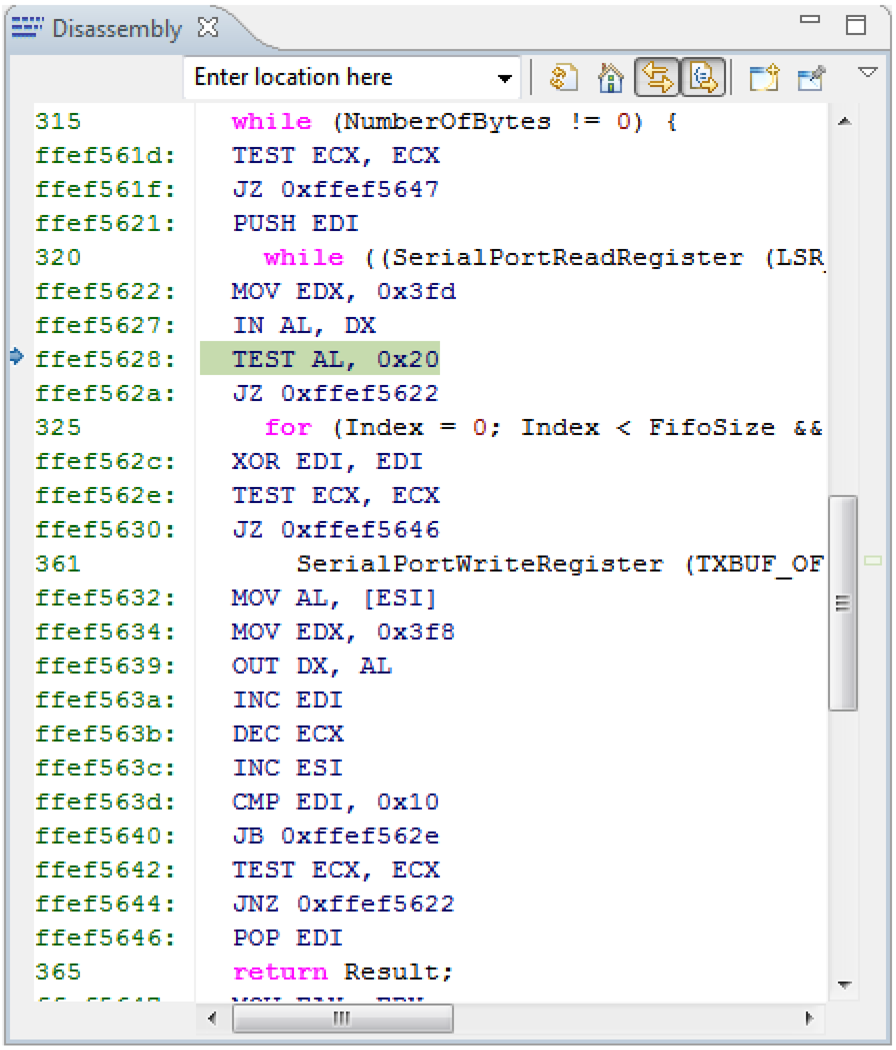
```
Name : RIP
    Hex:7C739476
    Decimal:2087949430
    Octal:17434712166
    Binary:1111100011100111001010001110110
    Default:7C739476
```

Figure 13: Register View

Working with Disassembly

Users can Open\Close the Disassembly View using:

- VeB Menu: DCI Debug→View→Assembler View
- Or using Debug Toolbar Icon [ASM]



```
315          while (NumberOfBytes != 0) {
ffef561d:    TEST ECX, ECX
ffef561f:    JZ 0xffef5647
ffef5621:    PUSH EDI
320            while ((SerialPortReadRegister (LSR
ffef5622:    MOV EDX, 0x3fd
ffef5627:    IN AL, DX
ffef5628:    TEST AL, 0x20
ffef562a:    JZ 0xffef5622
325            for (Index = 0; Index < FifoSize &&
ffef562c:    XOR EDI, EDI
ffef562e:    TEST ECX, ECX
ffef5630:    JZ 0xffef5646
361              SerialPortWriteRegister (TXBUF_OF
ffef5632:    MOV AL, [ESI]
ffef5634:    MOV EDX, 0x3f8
ffef5639:    OUT DX, AL
ffef563a:    INC EDI
ffef563b:    DEC ECX
ffef563c:    INC ESI
ffef563d:    CMP EDI, 0x10
ffef5640:    JB 0xffef562e
ffef5642:    TEST ECX, ECX
ffef5644:    JNZ 0xffef5622
ffef5646:    POP EDI
365          return Result;
```

Figure 14: Disassembly View

The Disassembly window displays executable code in assembly language. This allows for debugging

- Real mode code
- Protected mode code
- Long mode code

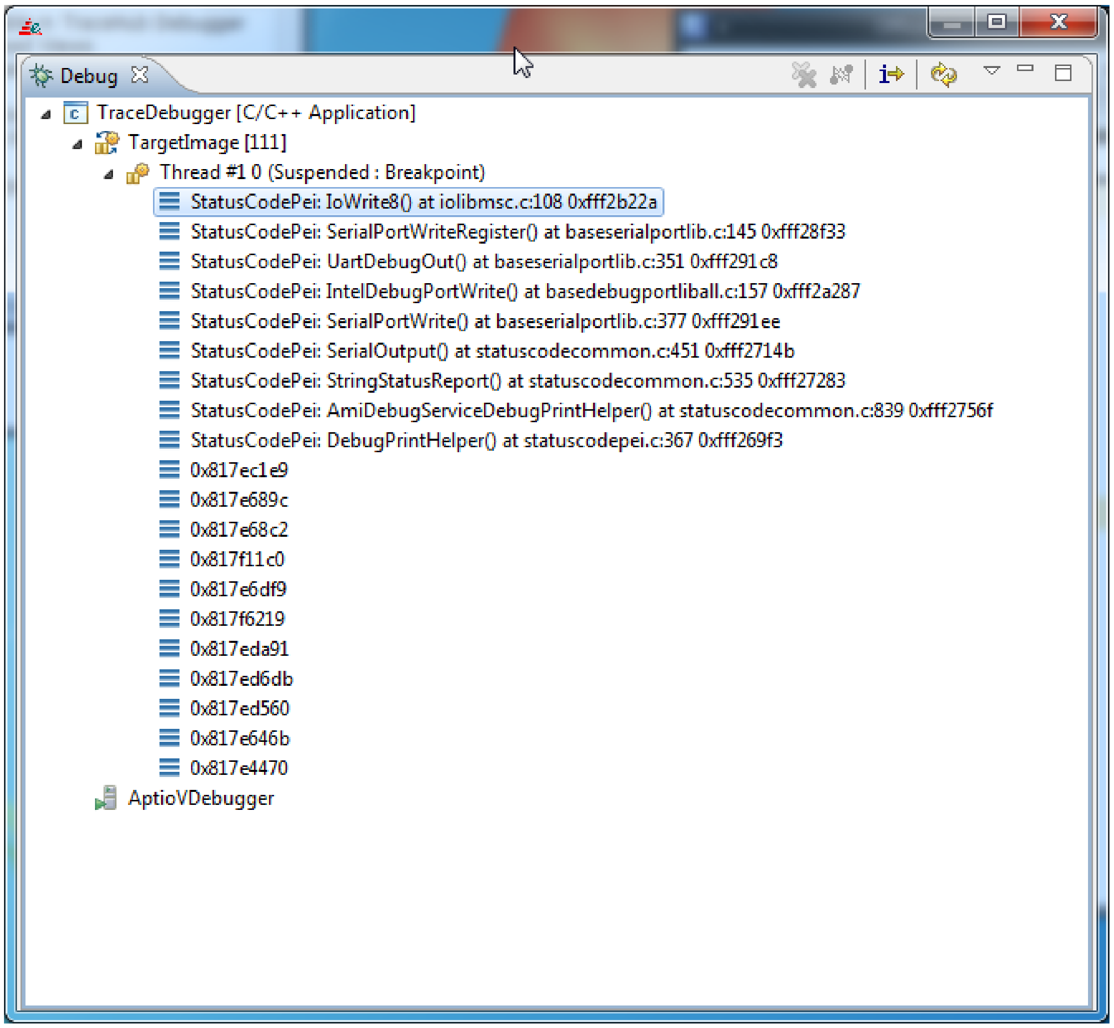
Working with Call Stack View

The Debug View contains the Call stack Displayed under TargetImage Thread.

We need to add following SDL Token to the project to get the call stack.

TOKEN
Name = "export EXTERNAL_GENFW_FLAGS"
Value = "--keepexceptiontable"
TokenType = Expression
Target AK = Yes

End



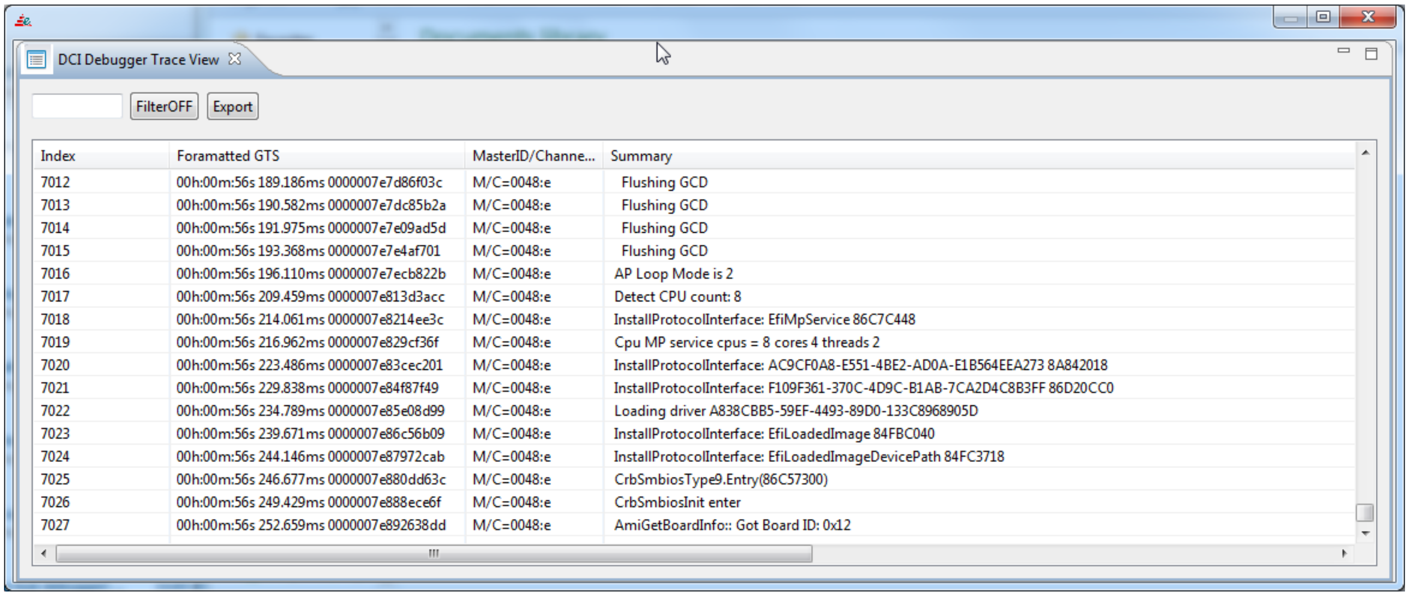
```
TraceDebugger [C/C++ Application]
  TargetImage [111]
    Thread #1 0 (Suspended : Breakpoint)
      StatusCodePei: IoWrite8() at iolibmsc.c:108 0xfff2b22a
      StatusCodePei: SerialPortWriteRegister() at baseserialportlib.c:145 0xfff28f33
      StatusCodePei: UartDebugOut() at baseserialportlib.c:351 0xfff291c8
      StatusCodePei: IntelDebugPortWrite() at basedebugportliball.c:157 0xfff2a287
      StatusCodePei: SerialPortWrite() at baseserialportlib.c:377 0xfff291ee
      StatusCodePei: SerialOutput() at statuscodecommon.c:451 0xfff2714b
      StatusCodePei: StringStatusReport() at statuscodecommon.c:535 0xfff27283
      StatusCodePei: AmiDebugServiceDebugPrintHelper() at statuscodecommon.c:839 0xfff2756f
      StatusCodePei: DebugPrintHelper() at statuscodepei.c:367 0xfff269f3
      0x817ec1e9
      0x817e689c
      0x817e68c2
      0x817f11c0
      0x817e6df9
      0x817f6219
      0x817eda91
      0x817ed6db
      0x817ed560
      0x817e646b
      0x817e4470
  AptioVDebugger
```

Figure 15: Callstack View

Working with Trace View

Debugger Supports capturing of Trace messages over DCI interface. Allows for filtering of trace messages and the following features.

- Export trace messages
- Tab separated text file
- Comma separated Value
- Allows for selecting start and end index to export


DCI Debugger Trace View

FilterOFF Export

Index	Foramatted GTS	MasterID/Channe...	Summary
7012	00h:00m:56s 189.186ms 0000007e7d86f03c	M/C=0048:e	Flushing GCD
7013	00h:00m:56s 190.582ms 0000007e7dc85b2a	M/C=0048:e	Flushing GCD
7014	00h:00m:56s 191.975ms 0000007e7e09ad5d	M/C=0048:e	Flushing GCD
7015	00h:00m:56s 193.368ms 0000007e7e4af701	M/C=0048:e	Flushing GCD
7016	00h:00m:56s 196.110ms 0000007e7ecb822b	M/C=0048:e	AP Loop Mode is 2
7017	00h:00m:56s 209.459ms 0000007e813d3acc	M/C=0048:e	Detect CPU count: 8
7018	00h:00m:56s 214.061ms 0000007e8214ee3c	M/C=0048:e	InstallProtocolInterface: EfiMpService 86C7C448
7019	00h:00m:56s 216.962ms 0000007e829cf36f	M/C=0048:e	Cpu MP service cpus = 8 cores 4 threads 2
7020	00h:00m:56s 223.486ms 0000007e83cec201	M/C=0048:e	InstallProtocolInterface: AC9CF0A8-E551-4BE2-AD0A-E1B564EEA273 8A842018
7021	00h:00m:56s 229.838ms 0000007e84f87f49	M/C=0048:e	InstallProtocolInterface: F109F361-370C-4D9C-B1AB-7CA2D4C8B3FF 86D20CC0
7022	00h:00m:56s 234.789ms 0000007e85e08d99	M/C=0048:e	Loading driver A838CBB5-59EF-4493-89D0-133C8968905D
7023	00h:00m:56s 239.671ms 0000007e86c56b09	M/C=0048:e	InstallProtocolInterface: EfiLoadedImage 84FBC040
7024	00h:00m:56s 244.146ms 0000007e87972cab	M/C=0048:e	InstallProtocolInterface: EfiLoadedImageDevicePath 84FC3718
7025	00h:00m:56s 246.677ms 0000007e880dd63c	M/C=0048:e	CrbSmbiosType9.Entry(86C57300)
7026	00h:00m:56s 249.429ms 0000007e888ece6f	M/C=0048:e	CrbSmbiosInit enter
7027	00h:00m:56s 252.659ms 0000007e892638dd	M/C=0048:e	AmiGetBoardInfo:: Got Board ID: 0x12

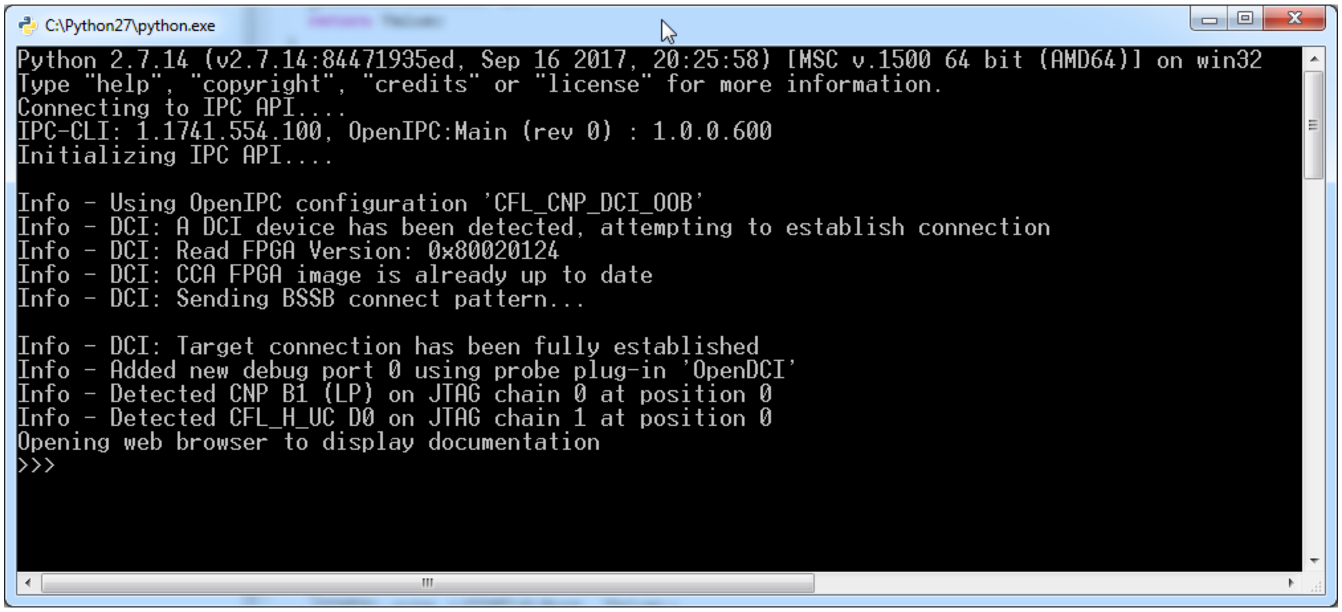
Figure 16: AMI Debugger Trace View

Working with Python Console/ User scripts

Debugger supports executing ITP / IPC commands. User can open the python scripting console using:

- VeB Menu: DCI Debug→View→Script
- Or using Debug Toolbar Icon [▣]

It launches the python console and can run individual commands. It can be used simultaneously with Debug session.


C:\Python27\python.exe

```
Python 2.7.14 (v2.7.14:84471935ed, Sep 16 2017, 20:25:58) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
Connecting to IPC API....
IPC-CLI: 1.1741.554.100, OpenIPC:Main (rev 0) : 1.0.0.600
Initializing IPC API....

Info - Using OpenIPC configuration 'CFL_CNP_DCI_OOB'
Info - DCI: A DCI device has been detected, attempting to establish connection
Info - DCI: Read FPGA Version: 0x80020124
Info - DCI: CCA FPGA image is already up to date
Info - DCI: Sending BSSB connect pattern...

Info - DCI: Target connection has been fully established
Info - Added new debug port 0 using probe plug-in 'OpenDCI'
Info - Detected CNP B1 (LP) on JTAG chain 0 at position 0
Info - Detected CFL_H_UC D0 on JTAG chain 1 at position 0
Opening web browser to display documentation
>>>
```

Figure 17: Python console

Debugger also supports to execute Scripts provided by silicon vendor.

User can access this using menu: DCI Debug View User Script or using Toolbar icon [ ]

Script automatically sets up connection with target and executes commands.

Execution Control

Resume

If process is halt at breakpoint, you can resume the execution by selecting Resume on Debug Menu or by pressing F8 or by selection tool bar icon []. The execution of the application continues until another breakpoint is encountered.

Break

When the target is running, most debugger actions are unavailable. If you want to stop a running target, you can issue a Break command by clicking on Break on Debug Menu or by selecting tool bar icon [ ] or by using F10. This command causes the debugger to break into the target. That is, the debugger stops the target and all control is given to the debugger. If a running target encounters an exception, if certain events occur, if a breakpoint is hit, or if the application closes normally, the target breaks into the debugger and a message appears in the Debugger Command window and describes the error, event, or breakpoint.

Reset Target

Click Reset Target on the Debug menu or select debugger tool bar icon [] or the hotkey F12 to reset the target

Stepping Controls

Step Into

Click Step Into on the Debug menu or select debugger tool bar icon [] or use the hot key F5 to execute a single instruction on the target.

If a function call occurs, Step Into enters the function and continues stepping through each instruction.

Step Over

Click Step Over on the Debug menu or select debugger tool bar icon [] or use the hot key F6 to execute a single instruction on the target. If the instruction is a function call, the whole function is executed .Step over treats the function call as a single step. When the debugger is in Assembly Mode, stepping occurs one machine instruction at a time. When the debugger is in Source Mode, stepping occurs one source line at a time.

Step Return

Click Step Return on the Debug menu or select debugger tool bar icon [] or use the hot key F7 to execute Step Return. It is used to return from a method, which has been stepped into. Even though we return from the method, the remainder of the code inside the method will be executed normally.

Run To Line

In the Source View, the user can select on a valid line in the execution control and select the Run to Line option from the Debug Menu or use the hot key CTRL+R or using tool bar icon [ ]. The Disassembly view provides the option to 'Run to Line' using its Right click menu option Like in below image. Selecting this option will runs code until the selected line encounters and halt at the selected line.

Run to Line is very useful when it comes to skipping over parts of a function that you do not want to step through. For example, if you have a loop in your method and you do not want to step through the loop, select a line just after the loop and select Run to Line (CTRL+R).

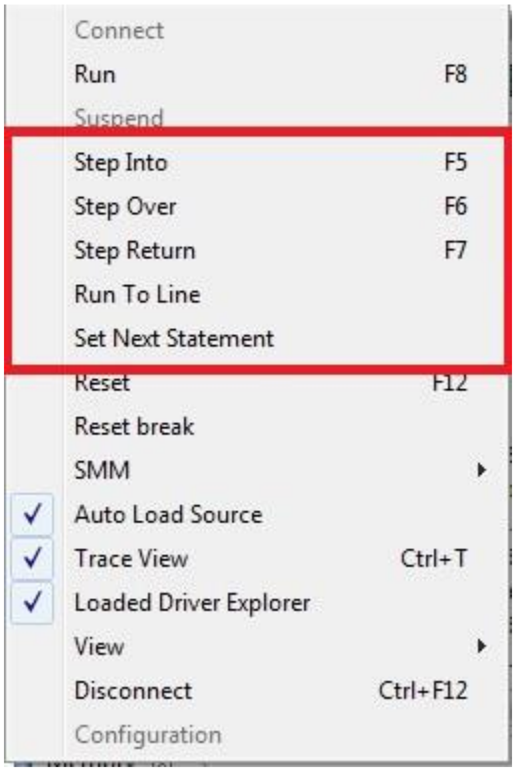


Figure 18: Run control Menu

Set Next Statement

Set Next Statement is frequently used in conjunction with edit and continue. That is, after changing some code during the debug session, you can set the next statement to the location before the change and re-execute the code to see the effect of your changes. Set Next Statement allows you to jump around in a method without executing any code in between. You can also use set next statement to jump over code that you believe is buggy, or to jump past a conditional test on an if statement to execute a path of code that you want to debug without having to make the condition true.

In the Source View, the user can select on a valid line in the execution control and select the Set Next Statement option from the Debug Menu or select debugger icon [ ]or use the hot key CTRL+SHIFT+N. The Disassembly view provides the option to 'Move to Line' using its Right click menu option, like in above image. Move to Line works similar to Set Next Statement option.

Breakpoints - Setting & Usage

Setting a breakpoint can be done at Boot time or Compile time using any of the methods mentioned below.

Breakpoint @ Compile Time (In Code)

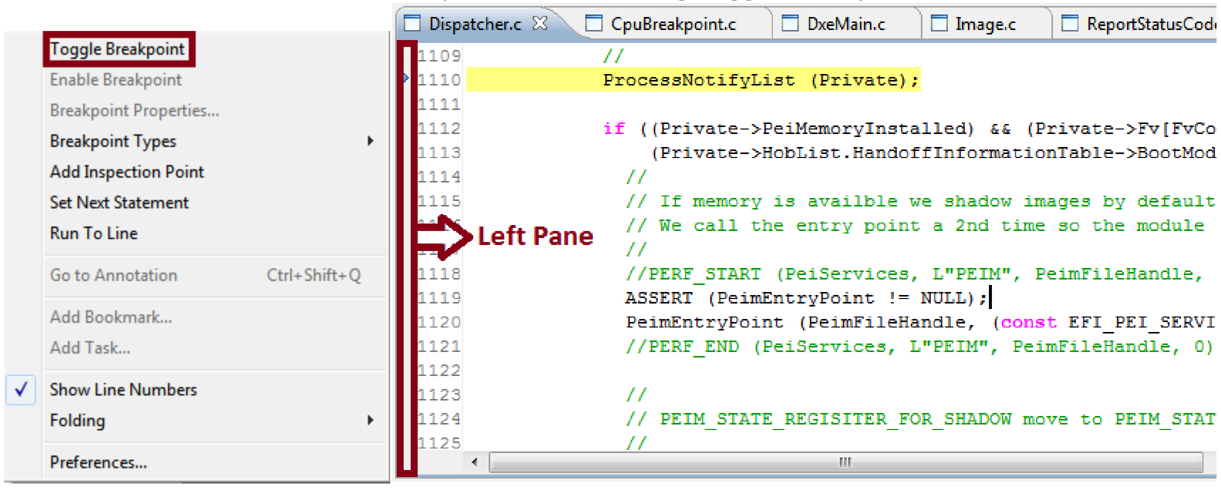
Setting a breakpoint in the code can be done by adding the below mentioned code in the place where the user wishes the target to break –

- __debugbreak ();

Breakpoint @ Boot Time

Using Editor Pane (Source & Disassembly Views)

One way to set a breakpoint on any source or Disassembly line is by right clicking on the Left Pane of Source View or Disassembly view and selecting Toggle Breakpoint as shown below.



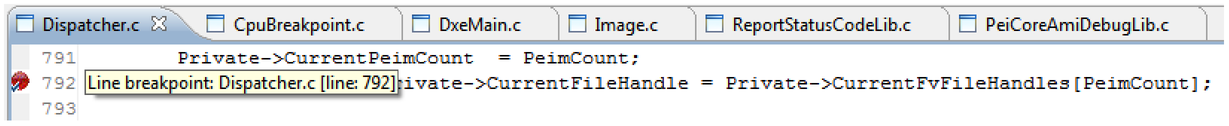
```
        //
        ProcessNotifyList (Private);

        if ((Private->PeiMemoryInstalled) && (Private->Fv[FvCo
            (Private->HobList.HandoffInformationTable->BootMod
          //
          // If memory is availble we shadow images by default
          // We call the entry point a 2nd time so the module
          //
          //PERF_START (PeiServices, L"PEIM", PeimFileHandle,
          ASSERT (PeimEntryPoint != NULL);
          PeimEntryPoint (PeimFileHandle, (const EFI_PEI_SERVI
          //PERF_END (PeiServices, L"PEIM", PeimFileHandle, 0)

          //
          // PEIM_STATE_REGISITER_FOR_SHADOW move to PEIM_STAT
          //
```

Figure 19: Breakpoint Context Menu

Another way to quickly Toggle a breakpoint is by "double clicking" on the left pane, next to the desired source line in Source View or Disassembly View.



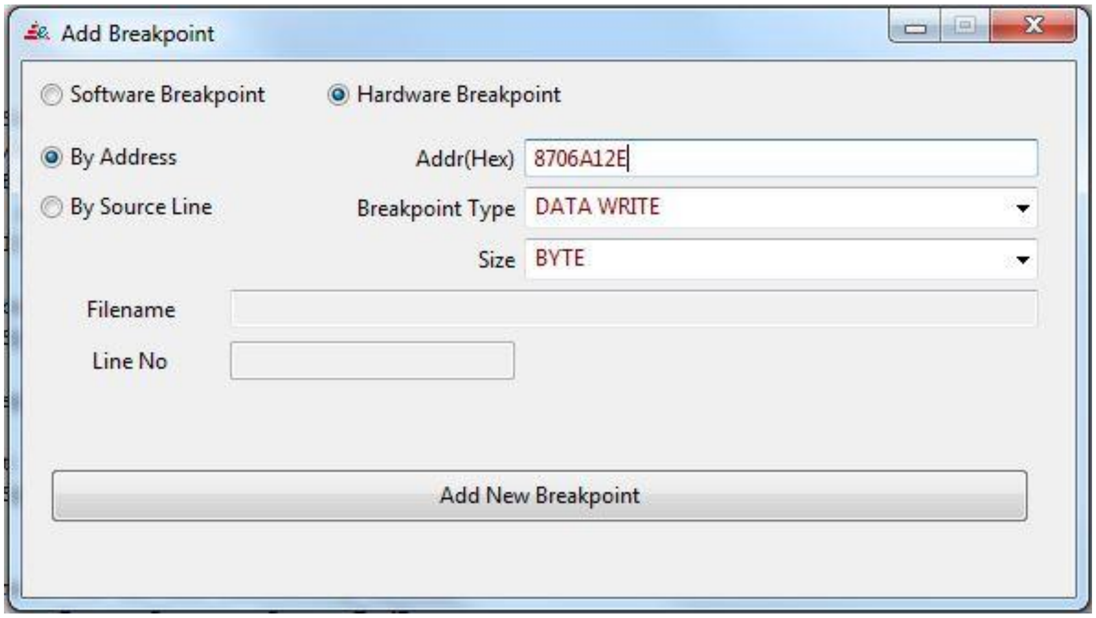
```
791         Private->CurrentPeimCount  = PeimCount;
792         Private->CurrentFileHandle = Private->CurrentFvFileHandles[PeimCount];
793
```

Figure 20: Source Breakpoint

Setting a Custom Breakpoint:

Users can set custom Breakpoints (HW or SW), to halt on a known desired location in the BIOS boot process. In order to add a new custom breakpoint user can do by using:

- VeB Menu: *DCI Debug→View → Breakpoint View*
- Toolbar: using icon [✚]



Software Breakpoint [✓ Hardware Breakpoint]

[✓ By Address] By Source Line

Addr(Hex) 8706A12E

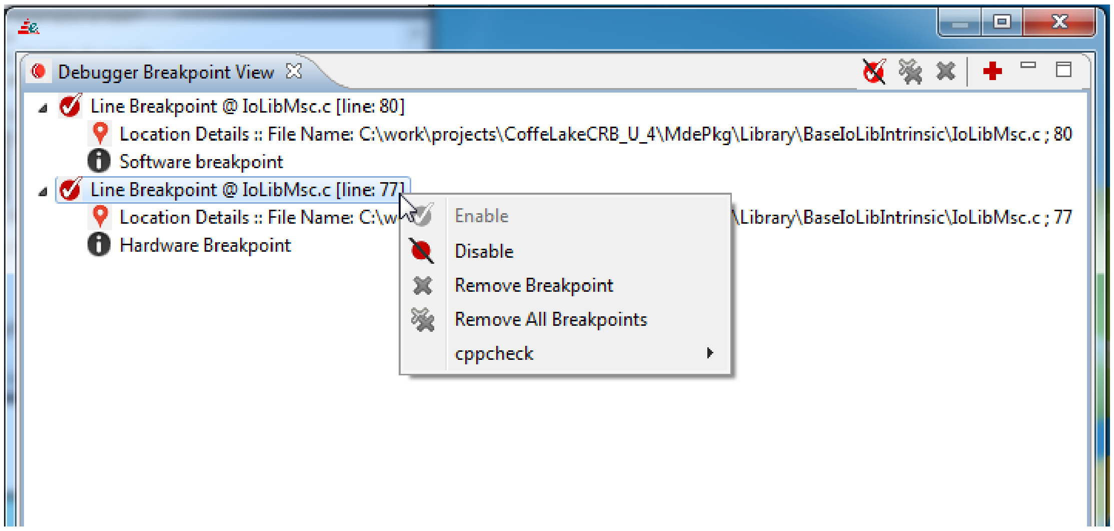
Breakpoint Type DATA WRITE

Size BYTE

Filename

Line No

Figure 21: Breakpoint Wizard



Line Breakpoint @ IoLibMsc.c [line: 80]
Location Details :: File Name: C:\work\projects\CoffeLakeCRB_U_4\MdePkg\Library\BaseIoLibIntrinsic\IoLibMsc.c ; 80
Software breakpoint
Line Breakpoint @ IoLibMsc.c [line: 77]
Location Details :: File Name: C:\w \Library\BaseIoLibIntrinsic\IoLibMsc.c ; 77
Hardware Breakpoint

Figure 22: Breakpoint View

Halting at Reset Vector:

Debugger allows to halt target at Reset Vector and debug through Reset Vector by using

- VeB Menu: *DCI Debug→Reset Break*

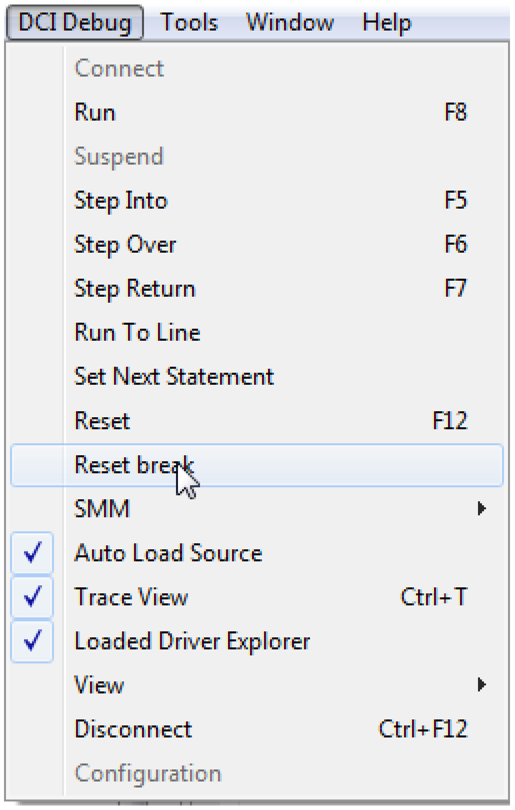


Figure 23: DCI Menu

- Toolbar: using icon []

SMM Debugging

Halting on SMM Entry\Exit

Select: DCI Debug□SMM□Break on SMM Entry/Exit from Menu item or Select SMM Entry/Exit from Toolbar [].

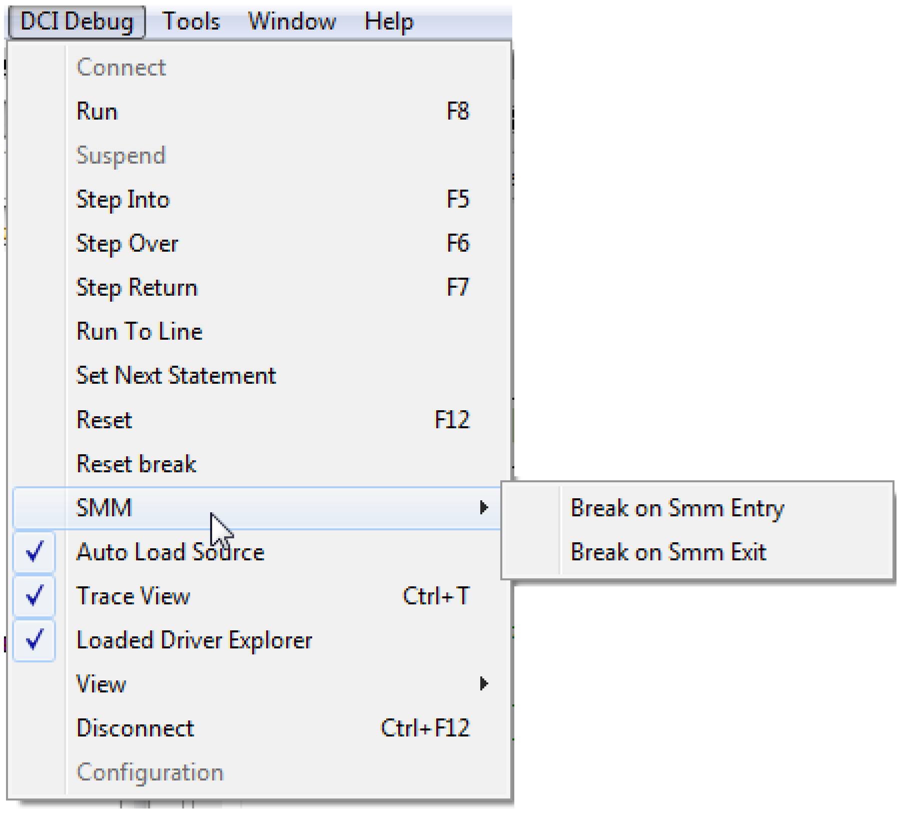


Figure 24: SMM Menu

NOTE: *Requires PCD "gUefiCpuPkgTokenSpaceGuid.PcdCpuSmmEnableBspElection" to be set to FALSE*

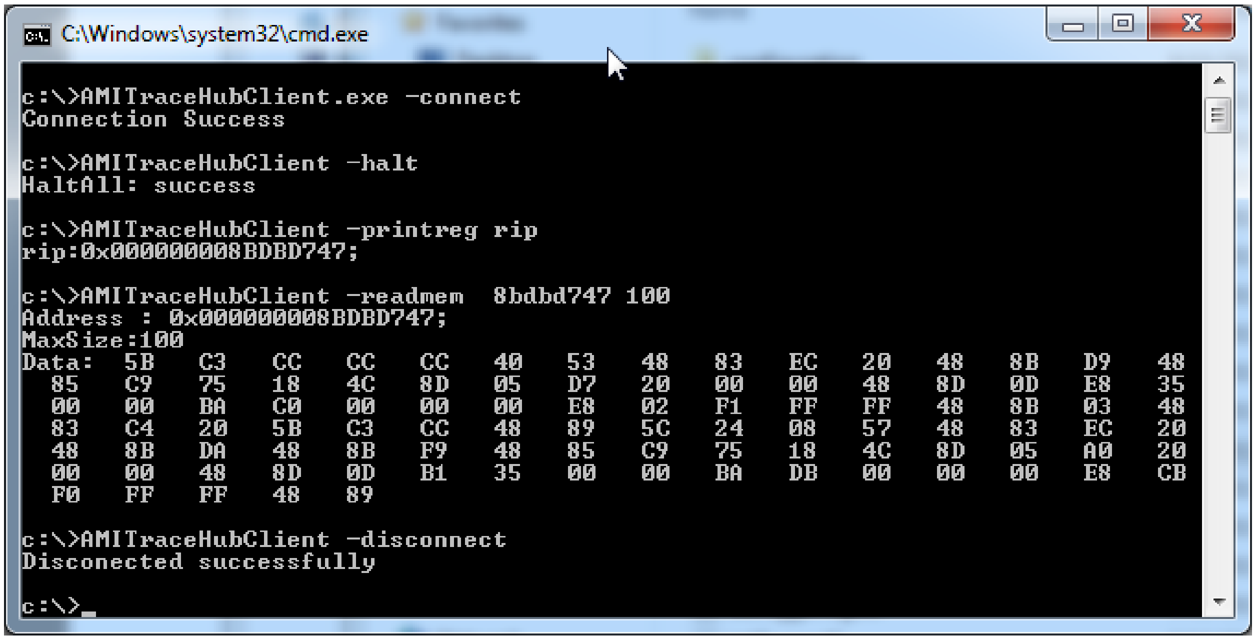
Perform Basic DCI Functional Test

Once the DCI hardware setup is ready, software is installed, and the Platform configuration is selected (through VisualeBios), perform the following steps to establish basic access and verify basic functionality through DCI.

Make sure the AMITraceHubClient.exe and AMITraceHubServer.exe are exist in the VisualeBios folder.

- Using the VEB make sure that to select the correct platform and connection method
- Open command prompt window and change directory to VisualeBios location
- Run command “**AMITraceHubClient -connect**” to connect to the platform
- Wait for the platform to complete the DCI connectivity sequence
- If needed refer to the section “The Intel® CCA Hardware LED Indicators” for details
- After the software finishes the initialization sequence, use the following basic commands (in the order listed) to test out the basic function of DCI:

Command	Function
AMITraceHubClient -halt	The CPU enters probe mode and saves states.
AMITraceHubClient -printreg <Register>	Print the value of register mentioned
AMITraceHubClient -readmem <address> <length>	Read <length> byte memory from <address>
AMITraceHubClient -run	The CPU exits probe mode and restores states.
AMITraceHubClient -disconnect	Ends the current session and terminate the server



C:\Windows\system32\cmd.exe

```
c:\>AMITraceHubClient.exe -connect
Connection Success

c:\>AMITraceHubClient -halt
HaltAll: success

c:\>AMITraceHubClient -printreg rip
rip:0x000000008BDBD747;

c:\>AMITraceHubClient -readmem  8bdbd747 100
Address : 0x000000008BDBD747;
MaxSize:100
Data:  5B   C3   CC   CC   CC   40   53   48   83   EC   20   48   8B   D9   48
  85   C9   75   18   4C   8D   05   D7   20   00   00   48   8D   0D   E8   35
  00   00   BA   C0   00   00   00   E8   02   F1   FF   FF   48   8B   03   48
  83   C4   20   5B   C3   CC   48   89   5C   24   08   57   48   83   EC   20
  48   8B   DA   48   8B   F9   48   85   C9   75   18   4C   8D   05   A0   20
  00   00   48   8D   0D   B1   35   00   00   BA   DB   00   00   00   E8   CB
  F0   FF   FF   48   89

c:\>AMITraceHubClient -disconnect
Disconected successfully

c:\>_
```

Figure 25: Example of a console after the DCI is connected

General Notes

1. To identify the supported commands to execute within python console, execute the command "dir (<ipc command>)"
2. Target module is required to get the loaded driver notification to host.
3. Disassembly window must be in viewable area to be updated during debug session
4. Driver Load Information will not be listed if trace messages are not received from target, allowing only debugging using disassembly
5. To debug a specific module, add "__debugbreak()" code to break target before continuing with debug operation.
6. Breakpoint on IO Read and IO Write on the same address are displayed as two separate Breakpoints in the UI while they are treated as a single breakpoint by the core.
7. Edit and Continue of Files during the debug session is not supported.

Known Issues

1. Trace Messages on Skylake RVP11 platform are not supported
2. Tool tip using mouse hover in debug session does not working consistently
3. Step operations sometime takes time. Clicking 'Suspend' before the step out operation completes, will cause read memory error
4. Editing Variable in Variable view or Expression View does not refresh memory view of the modified variable requiring a step operation to refresh
5. Value edited in Variable window is not reflected in Memory window immediately
6. With Software debugger Auto Load of debugger available in Project when higher version is installed with VeB.
7. Reset break operation sometimes displays error occurred message in console.
8. After reset the target, immediately pressing the halt button to halt the target will cause the target error (Red light turn on).
9. Coffee Lake - While booting if user do reset, halt, resume continuously, sometimes getting read memory error.
10. Coffee Lake – Suspend operation fails sometimes when USB 3 debugging is used.
11. After setting SMM entry giving an error sometimes, but halting at SMM entry.
12. On Purley Refresh – Reset Break is not working as expected. It halts at different address. Every time the address is the same.
13. ipc.perfreport()' command is giving error occurred message sometimes.
14. ipc.threads[0].asm("$") is not working as expected in Purley.
15. Running command 'ipc.threads[0].port(0x80)' is always returning '00'.
16. IO breakpoint does not work properly with Trace Hub debugger.

Limitations

1. On target restart to enable User breakpoints or SMM breaks, the user needs to halt and resume the target once after restart.
2. Due to Open IPC limitation, if user tries to read invalid memory address, the target will give read memory error continuously and user need to restart the debug session to proceed with the debugging.
3. Maximum of 3 Hardware Breakpoints allowed. Since Debugger uses 1 Hardware breakpoint for internal operations, user can use maximum 3 hardware breakpoints.
4. Step Out is not expected to work if the RSP is modified by code, e.g. SMM Exit.
5. CPU Reset will clear all debug features, except for reset break. This means for example that user-specified breakpoints will be invalid until the target halts once after reset. Note that this halt can be due to either a reset-break, or due to a user-initiated halt. In either case the debugger will restore the necessary debug features.
6. On Coffee Lake, if the platform enters an Error state (identified by glowing red light) power cycle the platform after removing all connection between target and host
7. If using external python script to halt target, manual synchronization of VeB host and target is required
8. Step operations once the target has stopped at a HLT instruction is not possible. Set a breakpoint after the HLT and run to it instead.
9. SMM Entry/exit will disable/re-enable breakpoints, this means you cannot specify a breakpoint in SMRAM while halted outside of SMRAM. If you wish the break within SMRAM, you must first halt at the SMM entry-break and manually apply the breakpoint.
10. Read memory at reset vector, will cause the target error due to Jtag limitation (Red light turn on).
11. Trace hub debugger, No trace message through USB3 port (DCI USB).

Troubleshooting

1. **Issue**: The Intel® CCA must be connected to the target through the appropriate 6” Intel® CCA cable.
   Solution: Refer to section 2.4.1. for the to select the appropriate 6” cable to connect between the Type-A target USB connector of the Intel® CCA (labeled as Target) and the target’s USB3 capable Type-A, Type-uAB, or Type-C connector.

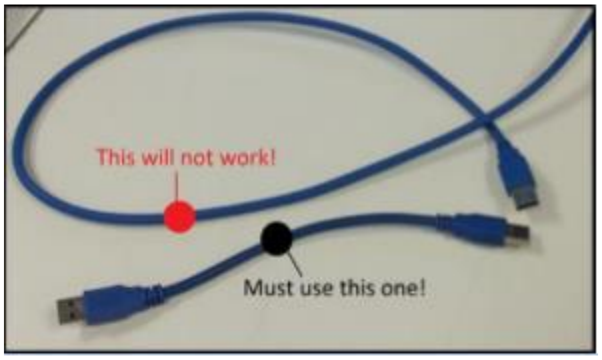


Figure 26: DCI CCA Cables

2. **Issue:** Intel® DAL connects to Intel® In-Target Probe (Intel®ITP) instead of DCI when both Intel® In-Target Probe (Intel® ITP) and DCI are present.
   Solution: Don’t have Intel® In-Target Probe (Intel® ITP) and DCI connected to the debug host at the same time.

3. Issue: VeB Trace view stops receiving Trace messages
   Solution: Stop Trace hub debugger by disconnecting and connect again.

The Intel® CCA Hardware LED Indicators

The Intel® CCA provides two LED indicators labelled as FIRMWARE and DCI CONNECT. The following table shows the Intel® CCA status depending on the LED color that displays.

Firmware LED	DCI Connect LED	Intel CCA Status
OFF	Off	Intel® CCA driver not found or failed to load correctly, or cable not connected to port.
Flashing Green	Off	The FPGA firmware of Intel® CCA is updating. This happens when a new Intel® DAL version is installed and run for the first time, if the new Intel® DAL also contains a new Intel® CCA FPGA update. The FW update may take 3–5 minutes to complete.
Off	Red	Driver loaded and USB host f/w uploaded. Ports' SSXTx is in e-Idle (USB Phy not responding—may be powered off or disabled).
Green	Red	Firmware moved from above state to the BSSB idle state (PCH exited eIdle on SSTx and Connection Status in BSSB = "Connected").
Green	Flashing Red	BSSB idle state (above state) and Intel® DAL s/w is attempting to connect with s/w driven connection patterns.
Green	Alternating Red / Orange	Same as above, but firmware is attempting to connect with the connection pattern (without s/w intervention).
Green	Orange	From the above two states, Intel® CCA then detects ExI status packet.
Green	Green	Target connection established—s/w sets the "Do Enable" (read IO enable) bit in the Intel® CCA.

		The Intel® CCA is ready for debugging!
Green	Flashing green	Target connection established. Tx traffic in progress.