



Trace Hub Debugger

Trace Hub Debugger

Document Revision 1.32

Jan 28, 2020



Confidential, NDA Required
Copyright © 2020

American Megatrends International LLC
5555 Oakbrook Parkway
Building 200
Norcross, GA 30093 (USA)

All Rights Reserved
Property of American Megatrends International LLC

Legal

Disclaimer

This publication contains proprietary information which is protected by copyright. No part of this publication may be reproduced, transcribed, stored in a retrieval system, translated into any language or computer language, or transmitted in any form whatsoever without the prior written consent of the publisher, American Megatrends International LLC. American Megatrends International LLC retains the right to update, change, modify this publication at any time, without notice.

For Additional Information

Call American Megatrends International LLC. at 1-800-828-9264 for additional information.

Limitations of Liability

In no event shall American Megatrends be held liable for any loss, expenses, or damages of any kind whatsoever, whether direct, indirect, incidental, or consequential, arising from the design or use of this product or the support materials provided with the product.

Limited Warranty

No warranties are made, either expressed or implied, with regard to the contents of this work, its merchantability, or fitness for a particular use. American Megatrends assumes no responsibility for errors and omissions or for the uses made of the material contained herein or reader decisions based on such use.

Trademark and Copyright Acknowledgments

Copyright © 2020 American Megatrends International LLC. All Rights Reserved.

American Megatrends International LLC

5555 Oakbrook Parkway

Building 200

Norcross, GA 30093 (USA)

All product names used in this publication are for identification purposes only and are trademarks of their respective companies.

Table of Contents

Document Information	6
Purpose	6
Overview	7
System Requirements	8
Software	8
Host software requirements	8
Hardware	8
Supported Platforms	9
Installation	11
Host side	11
Installing Trace Hub Collaterals	11
Installing Device Driver	12
Installing Intel® CCA Device Driver	12
Installing Intel® DBC Device Driver	13
Target Side	14
Active the Trace hub debugger	14
Using Trace Hub in VisualeBios	16
Setup Instructions:	17
Selecting Platform Configuration	17
Using Intel PVT collaterals	18
Updating Collaterals	19
Trace Hub Debugger (Intel platform specific) Features	20
Starting a Debug Session	23
Stopping a Debug Session	23
Working with Registers	24
Working with Disassembly	25
Working with Memory View	26



Working with Call Stack View	27
Working with Trace View	28
Working with Python Console	29
Working with CScripting	30
Working with IO View	31
Working with IIO View	32
Working with MSR View	33
Working with PCI.....	34
Getting NVRAM Variables Information.....	35
Working with Loaded driver view	36
Working with Halt at Module Entry	37
Working with Halt at Checkpoint.....	39
Working with Variables View	41
Editing a Variable	42
Additional Options	42
Watch Variables	43
Working with Expressions (Local and Global Variables)	44
Displaying CPU Information	45
Working with Progress Code View.....	46
Working with Source Lookup path Editor	49
Working with Break on Function Names	50
Execution Control.....	51
Resume.....	51
Break	51
Reset Target	52
Stepping Controls.....	52
Step Into.....	52
Step Over.....	52
Step Return	52



Run To Line.....	52
Set Next Statement	53
Understanding Breakpoint vs Inspection Point	54
Breakpoints - Setting & Usage	54
Inspection Points - Setting & Usage	61
Adding Conditions to a Breakpoint	67
Halting at Reset Vector	73
SMM Debugging.....	74
Working with Select Boot Phase to Halt	75
Working with Auto save user preferences.....	76
Perform Basic DCI Functional Test.....	77
General Notes	79
Troubleshooting.....	81
The Intel® CCA Hardware LED Indicators.....	82



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 versions of Windows OS 7\8\10
- VisualeBios (VeB). (BuildTools_32 or later versions)
- Java Runtime Environment (JRE) x86 - [\[Link\]](#)
- Python 2 X64 or Python 3 X64 [\[Link\]](#)
- Microsoft Visual C++ 2010 Redistributable x86 - [\[Link\]](#)
- Microsoft Visual C++ 2013 Redistributable x86 - [\[Link\]](#)
- Intel Trace hub Driver

Note: Debugger only support Java Runtime Environment (JRE) x86 version 8.0

Hardware

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

- Intel® DCI-OOB (DCI over an Intel® SVT Closed Chassis Adapter [Intel® CCA])

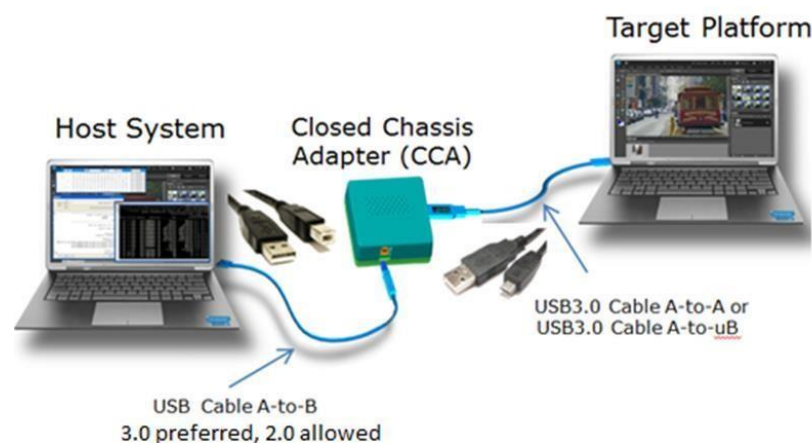


Figure 1-1. Hardware Configuration for Intel® DCI-OOB with Intel® CCA

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.

- Intel® DCI-USB (DCI over a USB3.0 Debug Cable)



Figure 1-2. 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

Platform	Debug	Trace	Connection Methods	comments
Skylake	✓	✓	OOB	DT RVP 8 Not supported
Kaby Lake	✓		OOB	No trace message support
Coffee Lake	✓	✓	OOB, USB3	
Canon Lake	✓	✓	OOB, USB3	NDA Collaterals required
Whiskey Lake	✓	✓	OOB, USB3	
Ice Lake	✓	✓	OOB, USB3	Tested stepping B CPU
Comet Lake	✓		OOB, USB3	
SkyLake server	✓		OOB, USB3	

Cascade Lake	✓		OOB, USB3	
Whitely	✓		OOB, USB3	NDA required
Tiger Lake	✓		OOB, USB3	NDA required
Snow Ridge	✓		OOB, USB3	NDA required

Note: Support available as of June 2019. Validated only in client platforms.



Installation

Host side

Download VEB label Buildtools_32 or later and use this VEB to download Debugger module, and open project. For AptioV projects, Download Debugger module (AptioV/trunk/AptioV/Binary/Modules/Debugger) labeled **Debugger_44** or later, reopen the project, AptioV debugger and trace hub debugger will be installed automatically.

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

Trace Hub Debugger requires execution of Python 2.7.x from command line. If Python 2.7.x is not installed in the system, it can be installed from [Here](#).

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 necessary 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 “DebuggerCollaterals”
- Click Select to add the collaterals to the project

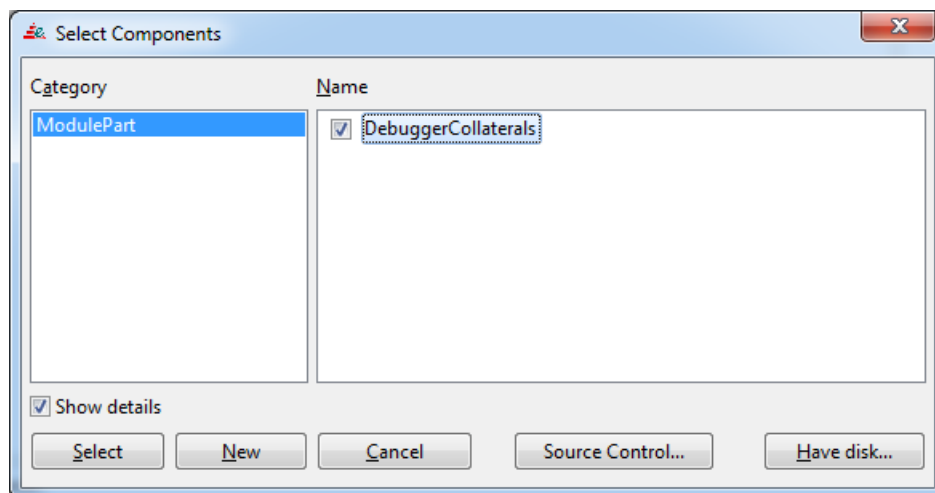


Figure 4: Installing Trace Hub Collaterals

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

NOTE: If the collateral is manually installed, VeB will not be able to identify this and the auto installation may not work for further updates. Hence, it is always recommended to install the collaterals (if you are using VeB later than BuildTools 32) by adding DebuggerCollateral component to the project.

Installing Device Driver

Communicating with the target platform requires the proper host drivers be installed. The driver is named "Intel_DCI_Driver_x64_1.10.msi" and can be found along with the DebuggerCollateral component.

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

Installing Intel® CCA Device Driver

- Copy "Intel_DCI_Driver_x64_1.10.msi" to a local folder
- Connect the CCA device to host system to either the USB 3.0 (preferred) or USB 2.0 port
- Open Windows Device Manager and look for entry "Intel DCI Transports (USB-Based)"

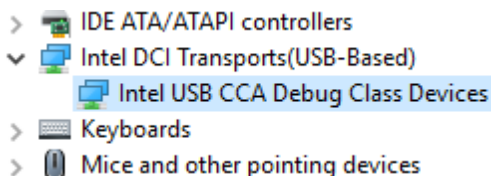


Figure 5: Device Driver view with Intel® CCA Device Driver installed

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”

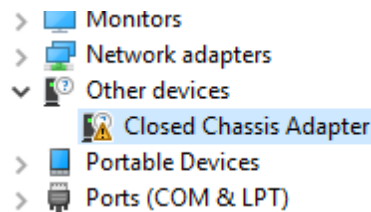


Figure 6: Device Manager view with Driver Not Installed

- Run the Intel_DCI_Driver_x64_1.10.msi to install the driver.

Installing Intel® DBC Device Driver

- Copy “Intel_DCI_Driver_x64_1.10.msi” to a local folder
- Connect a target with USB3 debug cable to host’s USB3 port.
- Open Windows Device Manager and look for entry “Intel DCI Transports (USB-Based)”

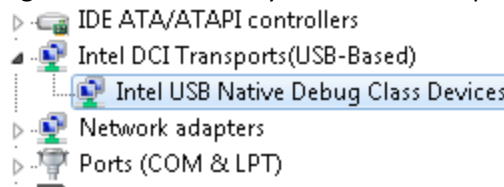


Figure 7: Device Driver view with Intel® DBC Device Driver installed

If found, the necessary drivers are already installed

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

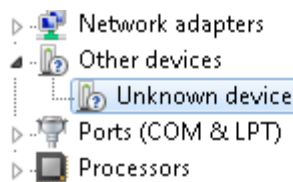


Figure 8: Device Manager view with Driver Not Installed

- Run the Intel_DCI_Driver_x64_1.10.msi to install the driver.

Note: for the host that does not has a USB3.0 port, user can purchase a PCI to USB3 card. Not all the PCI to USB3.0 card support Intel DBC. Suggest the [SHINESTAR USB 3.0 PCIe Expansion Card](#).

Target Side

Usually using a hardware debugger, target side change is not required. But to speed up the source look up process and other AMI Value adds function. We add this module.

Use VisualeBios to add optional AMI module from AptioV source control repository

- **\$/AptioV/Binary/Modules/Debugger (AMITraceHubDebugger component)**
Enable "AMITraceHubDebugger_SUPPORT"

Active the Trace hub debugger

Please Contact AMI For a license to active the trace hub debugger.

Then after install the debugger to VisualeBios.

Once get the license, select Menu Help->Manage Debugger license. The license info dialog will show.

On the incense info dialog, click "... " to find the license file.

Then click "Add" button, to add the license.

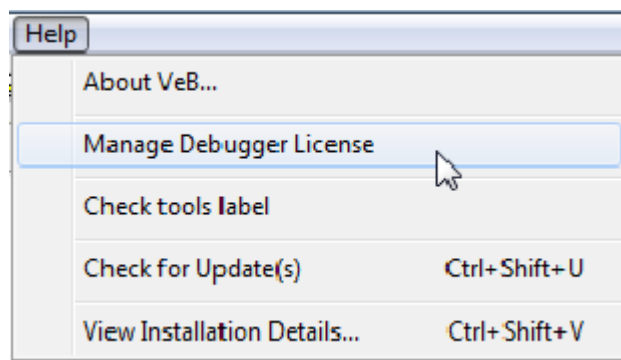


Figure 9: "Manage Debugger License" menu of the Trace Hub debugger

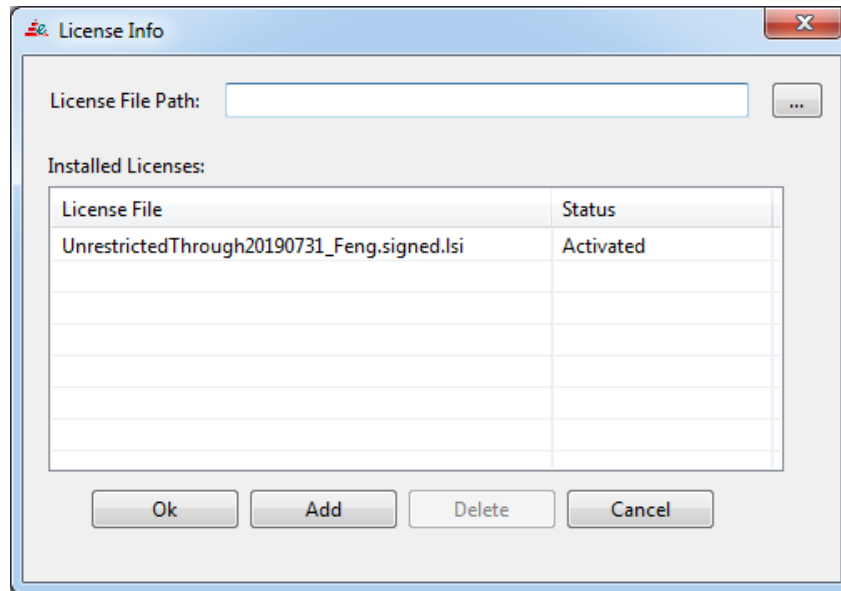


Figure 9: Active the Trace Hub debugger

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 “Trace Hub”.

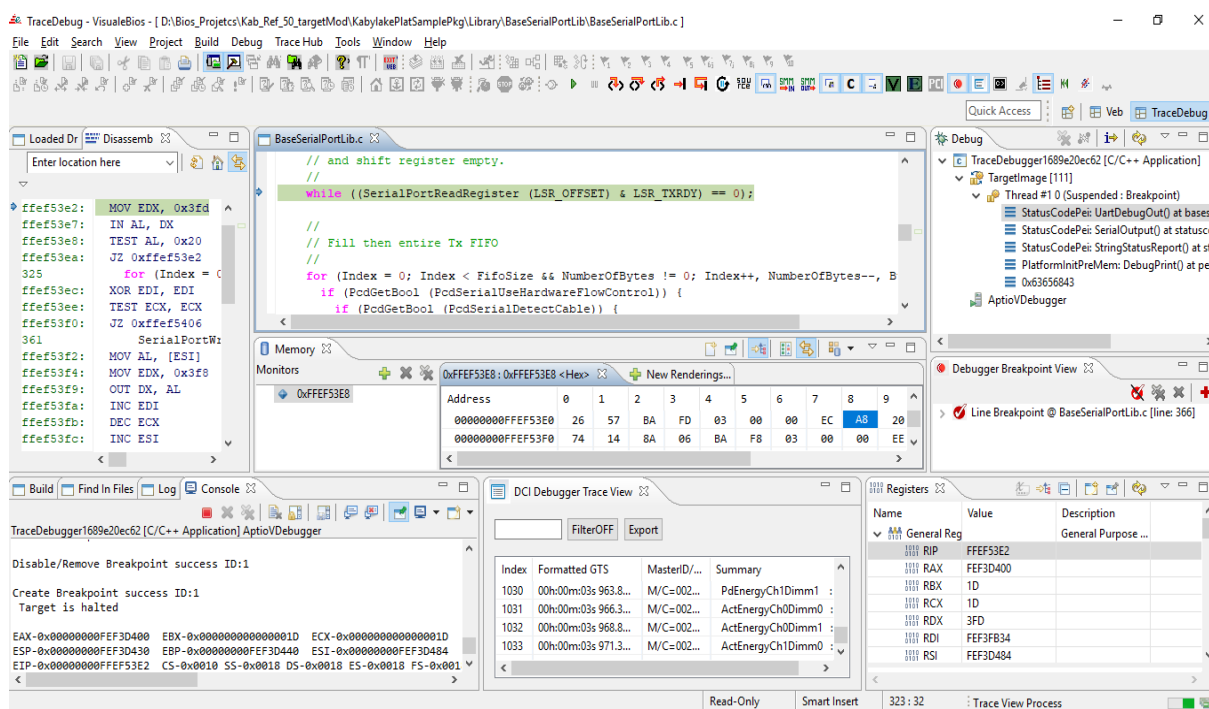


Figure 1: Trace Hub Debugger Views

Setup Instructions:

1. Use the fit tool (fit.exe) to enable the trace hub debugger
 - Please enable the following items
 - Debug->Direct Connect Interface (DCI) Enabled
 - Platform Protection-> CPU Debugging
 - Even if the trace hub is enabled by the fit tool, the BIOS may disable it. So, may need to config the BIOS also.
2. Enable the trace hub by BIOS setup
 - Please refer the specific start guide for that platform [here](#).

Selecting Platform Configuration

Before connecting, need to select the correct configuration file, to connect to the target.
Use menu **“Trace Hub-> Configuration”**

You will see the following dialog.

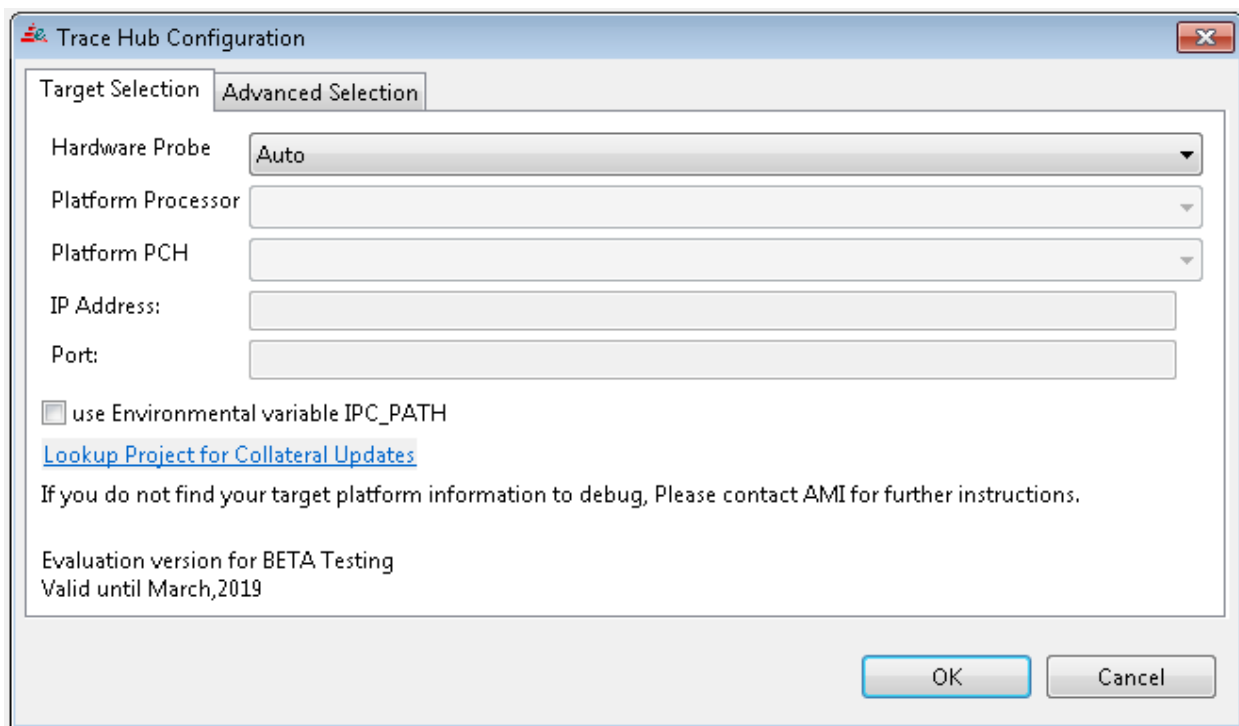


Figure 2: Trace hub Configuration



NOTE: If the platform is not available as part of the default configuration collaterals request collateral installation deliverable from AMI customer contact.

- The configuration dialog will display the hardware probes available for debugging
- Select the connection type as either “Auto”, “Intel(R) DCI OOB” or “Intel(R) DCI USB 3.x Debug Class”.

“Auto” is recommended.

Using Intel PVT collaterals

Additionally, users can specify the location of the installed Intel PVT to make use of Intel platform debug and trace collaterals.

- If Intel PVT is already installed ensure that the environment variable IPC_PATH is set to the correct path
- Click on the Checkbox “Use Environment Variable IPC_PATH”
- VeB will proceed to parse the collaterals found in the IPC_PATH and populate the Platform Processor and Platform PCH entries
- If IPC_PATH is not set, click on Advanced Tab
- Set Path value for the of IPC_PATH control
- Select Target Selection Tab and check the checkbox “Use Environment Variable IPC_PATH”
- VeB will proceed to parse and populate the controls

Note: If you are not sure about the correct configuration, you can select “Auto” in the “Hardware Probe” combo box. Debugger will select the correct configuration automatically.

Updating Collaterals

When new collaterals are available, follow the process to add the updated DebuggerCollateral 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.

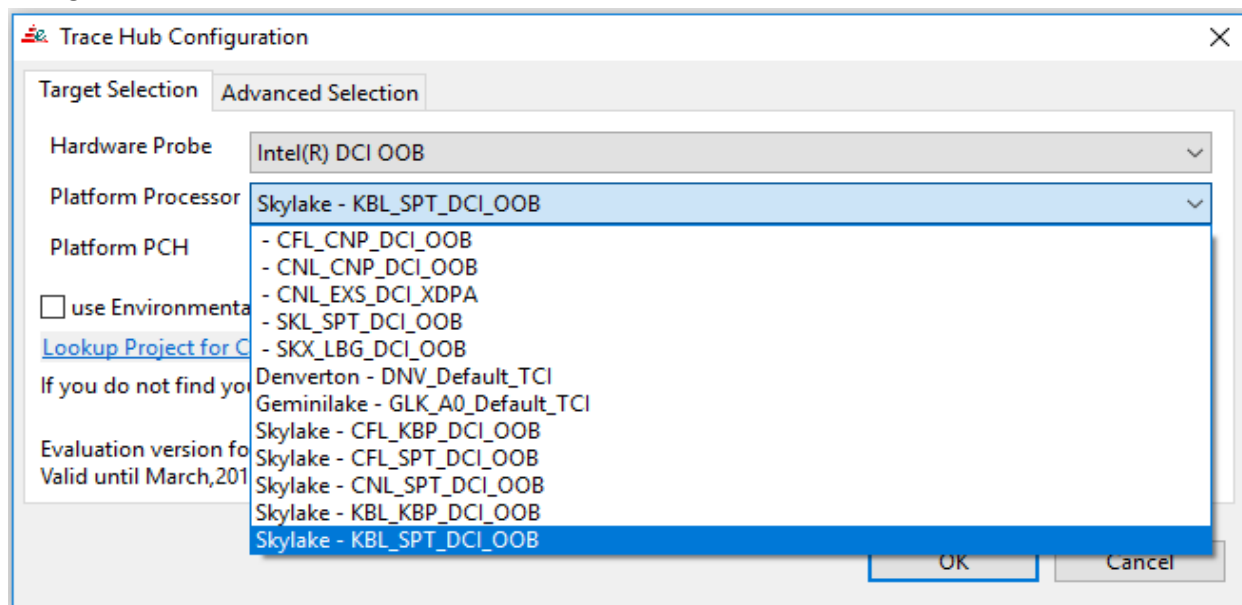


Figure 11: Updating Collaterals

NOTE: If the collateral is manually installed, VeB will not be able to identify this and the auto installation may not work for further updates. Hence, it is always recommended to install the collaterals (if you are using VeB later than BuildTools 32) by adding DebuggerCollateral component to the project.

Trace Hub Debugger (Intel platform specific) Features

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


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 <ul style="list-style-type: none"> - Source view with current line highlighted - Disassembly View - Mixed View support for source and disassembly 	✓	
Source level debugging <ul style="list-style-type: none"> - Stepping into - Step over Operation - Step Out Operation - Step return / Step Out - Run to Line - Set Next statement 	✓	
Call Stack Display	✓	

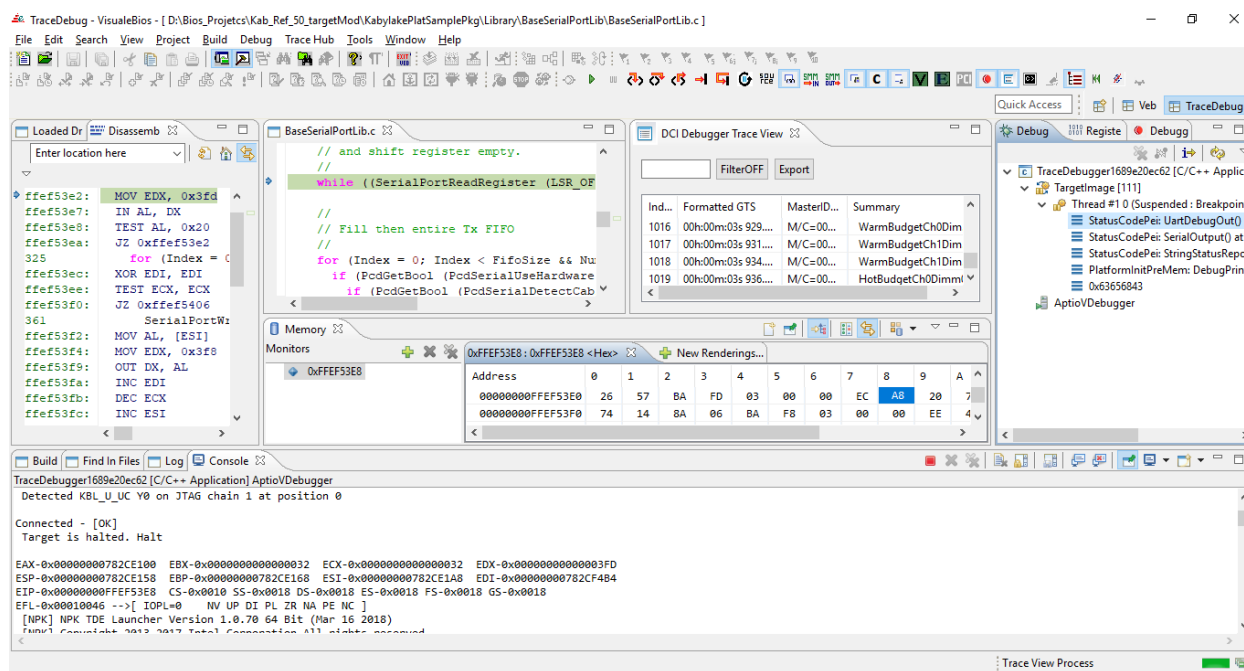
Debugger Toolbar	✓	
Features Technology / Feature	Supported	Comments
Memory - View and Edit Memory content	✓	
Local Variable view	✓	
Expression View	✓	
Multiple address monitoring	✓	Open multiple memory views at a time
IO View	✓	
IIO View	✓	
MSR View	✓	
CPUID View	✓	
Breakpoint	✓	Set breakpoints on IO, Memory access, Execution
<ul style="list-style-type: none"> – Hardware – Software – Reset Break 	✓	
– Conditional breakpoint	✓	PCI/Variable/Register/IO/Memory/Hit Count
– Inspection breakpoint	✓	PCI/Register/IO/Memory/Variable
Register view	✓	
PCI Wizard	✓	
NVRAM Variables	✓	
Source Editor	✓	
Halt at checkpoint	✓	
Halt at Module Entry	✓	

Progress code view	✓	Display the Progress code checkpoints
License Support	✓	
Special Breakpoints <ol style="list-style-type: none"> 1. PEI Entry point 2. On Memory Initialize 3. End of Pei 4. DXE Entry Point 5. SMM Init 6. SMM Entry 7. SMM Exit 8. Ready to Boot 9. Exit Boot Services 	✓	
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 “**Trace Hub→Connect**” or select the Connect [] from Toolbar (Target should be running)
6. Now target and host will be connected.




Stopping a Debug Session

To stop the target's Debugging Session,

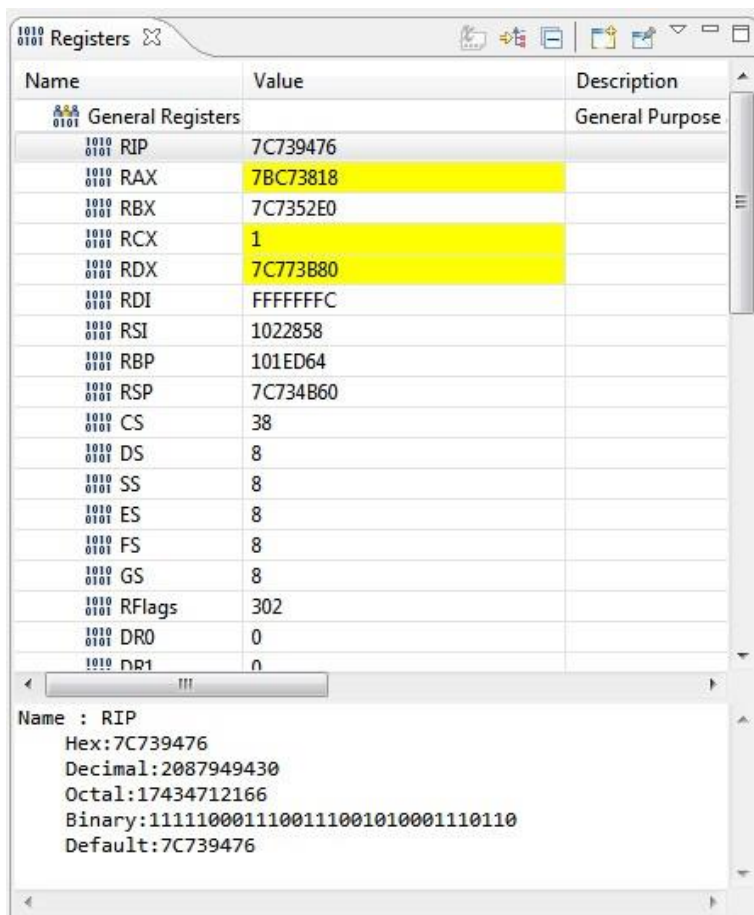
1. Click “**Trace Hub→Disconnect**” on the Debug menu or select the Toolbar Icon []
- This action enables you to end a previous debug session and start a New Debug Session if required.

Working with Registers

Users can Open or Close the Registers View using:


- VeBMenu: **Trace Hub** → **Windows** → **Registers**
- Or using Debugger Toolbar Icon []

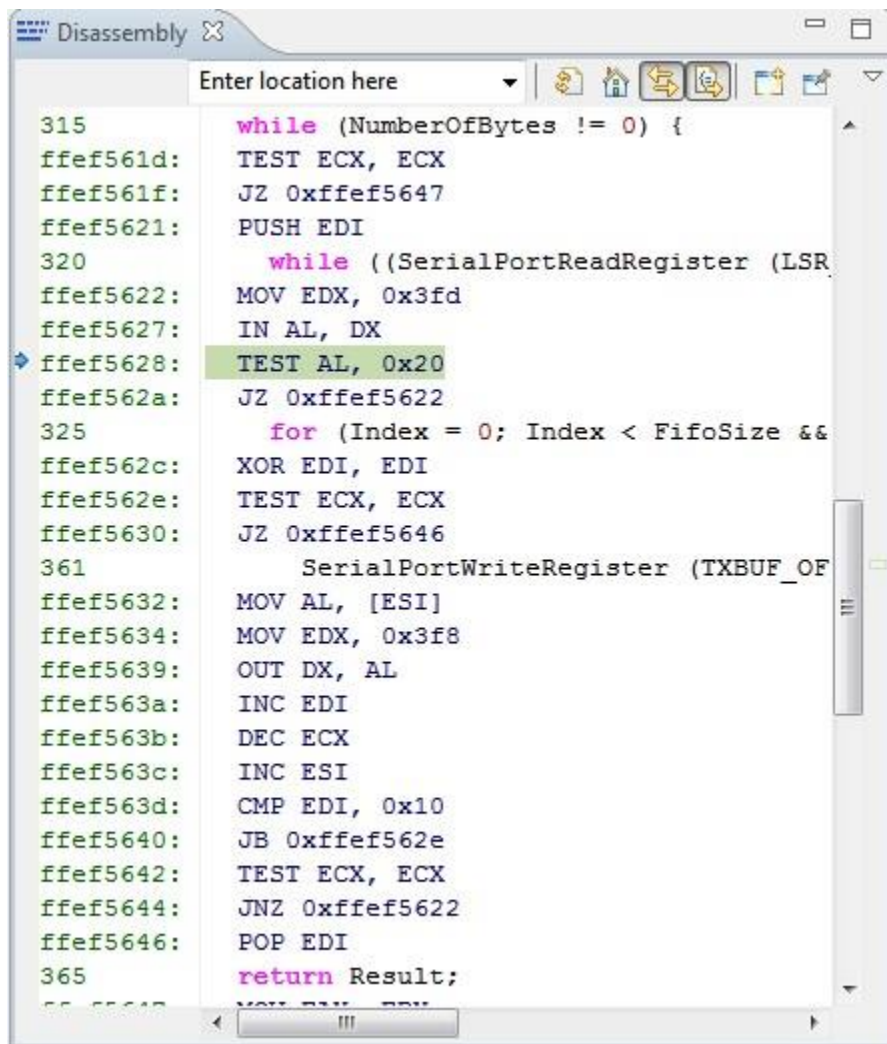
The Registers view will only be updated when the Target is in a halted state. The Registers window contains two columns. Name - Register Name & Value - the current value of the Register in Hexadecimal.



Working with Disassembly

Users can Open\Close the Disassembly View using:

- VeB Menu: **Trace Hub** → **Disassembly**
- Or using Debug Toolbar Icon []



```
315      while (NumberOfBytes != 0) {
ffe561d:    TEST ECX, ECX
ffe561f:    JZ 0xffef5647
ffe5621:    PUSH EDI
320      while ((SerialPortReadRegister (LSR
ffe5622:    MOV EDX, 0x3fd
ffe5627:    IN AL, DX
ffe5628:    TEST AL, 0x20
ffe562a:    JZ 0xffef5622
325      for (Index = 0; Index < FifoSize &&
ffe562c:    XOR EDI, EDI
ffe562e:    TEST ECX, ECX
ffe5630:    JZ 0xffef5646
361      SerialPortWriteRegister (TXBUF_OF
ffe5632:    MOV AL, [ESI]
ffe5634:    MOV EDX, 0x3f8
ffe5639:    OUT DX, AL
ffe563a:    INC EDI
ffe563b:    DEC ECX
ffe563c:    INC ESI
ffe563d:    CMP EDI, 0x10
ffe5640:    JB 0xffef562e
ffe5642:    TEST ECX, ECX
ffe5644:    JNZ 0xffef5622
ffe5646:    POP EDI
365      return Result;
ffe5647:    MOV EAX, EBX
```

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

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


Working with Memory View

Users can access the Memory View by -

Select: **Trace Hub ->Windows->Memory**



Or using Hotkey: CTRL+SHIFT+M

Or using the Debug Toolbar Icon []

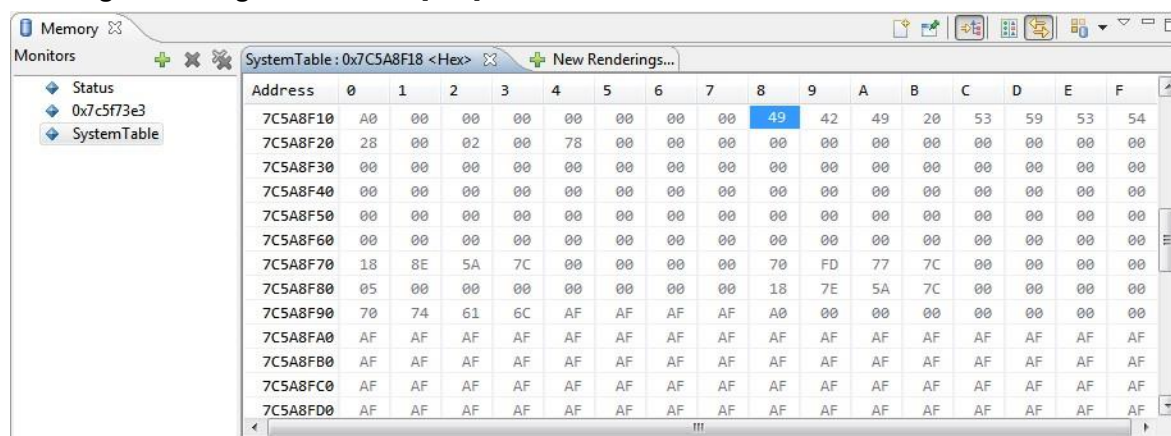


Figure: Memory View.

With the Memory view, you can look at the contents of memory at a specific address To add a new memory monitor from the Memory view.

Working of memory view is similar to software debugger.

NOTE: Arithmetic expressions are not supported in Memory view. When the user provides an arithmetic expression, VeB will display a warning to the user indicating the same.

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

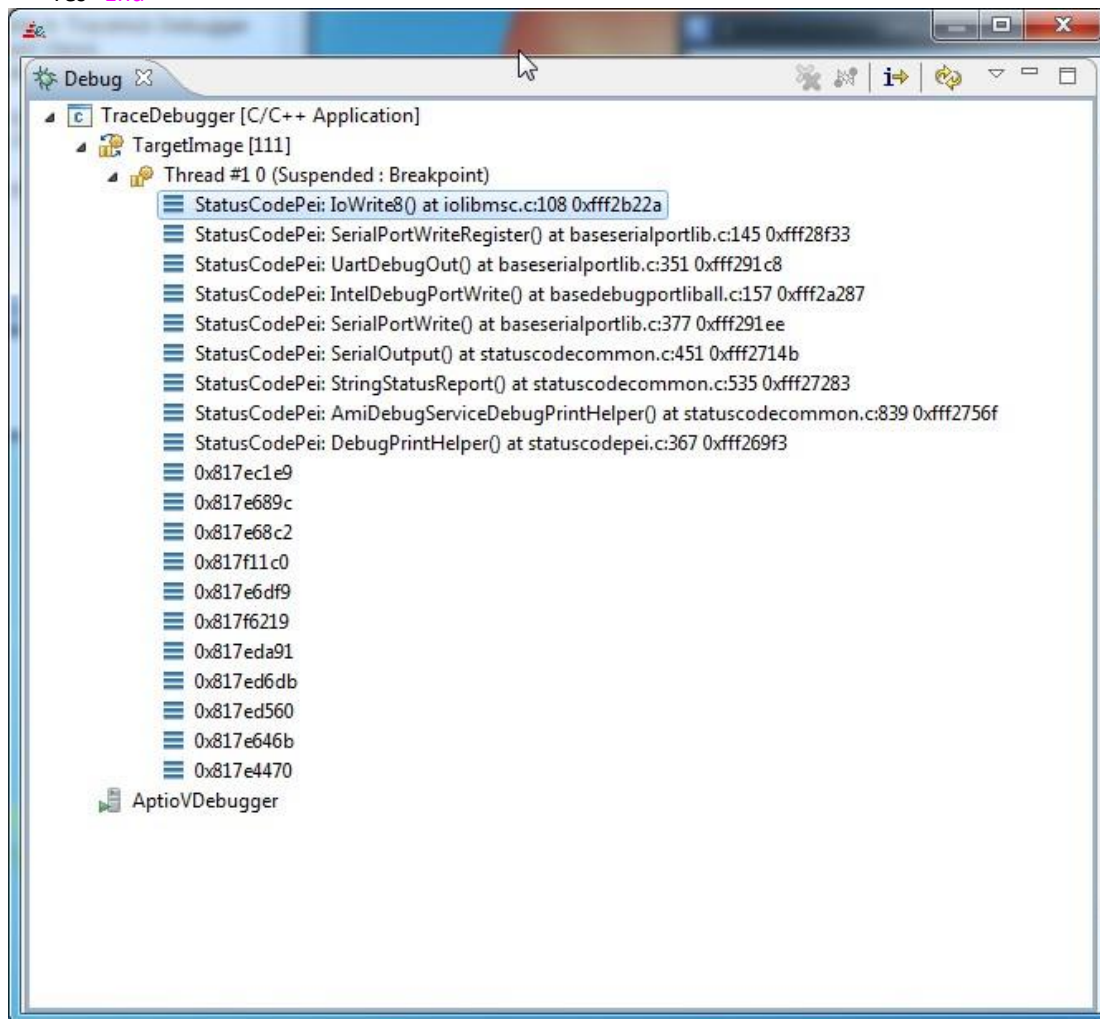
Name = "export EXTERNAL_GENFW_FLAGS"

Value = "--keepexceptiontable"

TokenType = Expression

Target AK

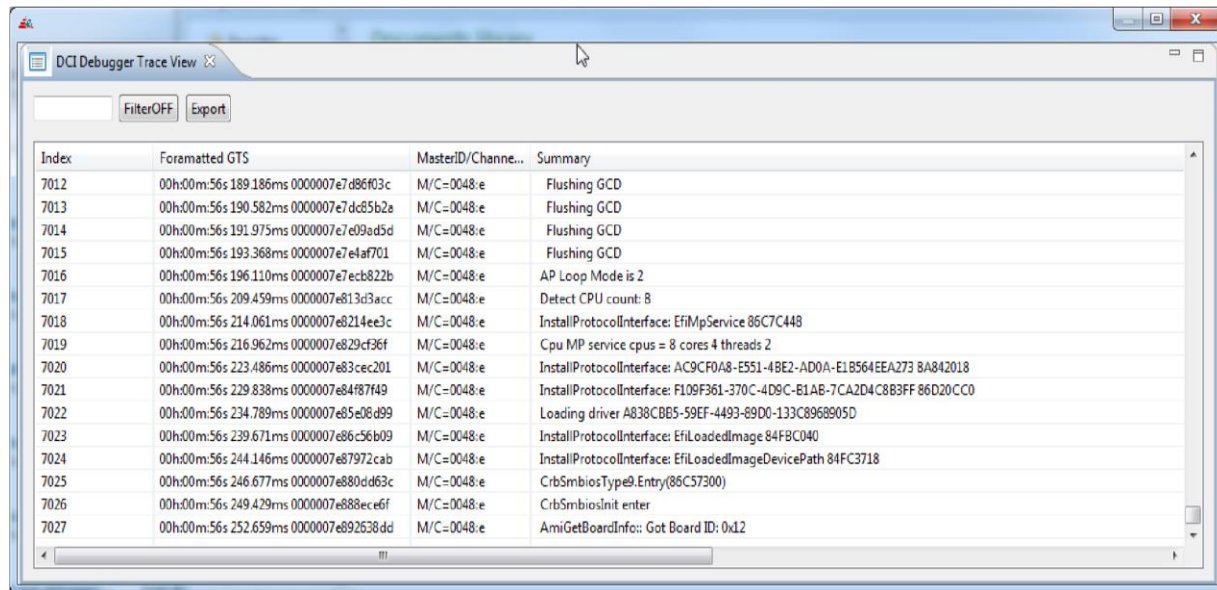
= Yes End



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




The screenshot shows the 'DCI Debugger Trace View' window. It has a 'FilterOFF' button and an 'Export' button. Below these is a table with four columns: Index, Formatted GTS, MasterID/Channel, and Summary. The table contains 16 rows of trace data.

Index	Formatted GTS	MasterID/Channel	Summary
7012	00h:00m:56s 189.186ms 0000007e7d86f03c	M/C=0048:e	Flushing GCD
7013	00h:00m:56s 190.582ms 0000007e7dc65b2a	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 A838CB85-59EF-4493-89D0-133C8968905D
7023	00h:00m:56s 239.671ms 0000007e86c56b09	M/C=0048:e	InstallProtocolInterface: EfiLoadedImage 84F8C040
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

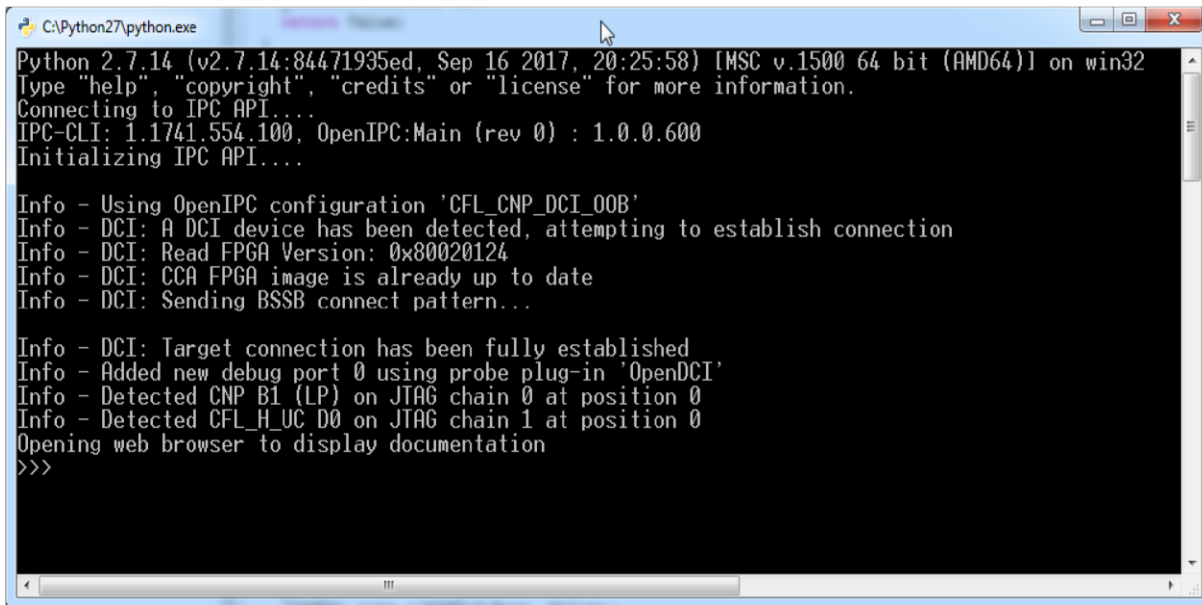
Trace view by default is enabled. But user can disable it, in the configuration dialog, by unchecking the checkbox "Enable Trace Messages".

Working with Python Console

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

- VeB Menu: **Trace Hub**→**Windows**→**Script**
- Or using Debug Toolbar Icon []

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


A screenshot of a Windows command prompt window titled 'C:\Python27\python.exe'. The window displays the output of a Python script. The text is as follows:

```
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_00B'
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
>>>
```

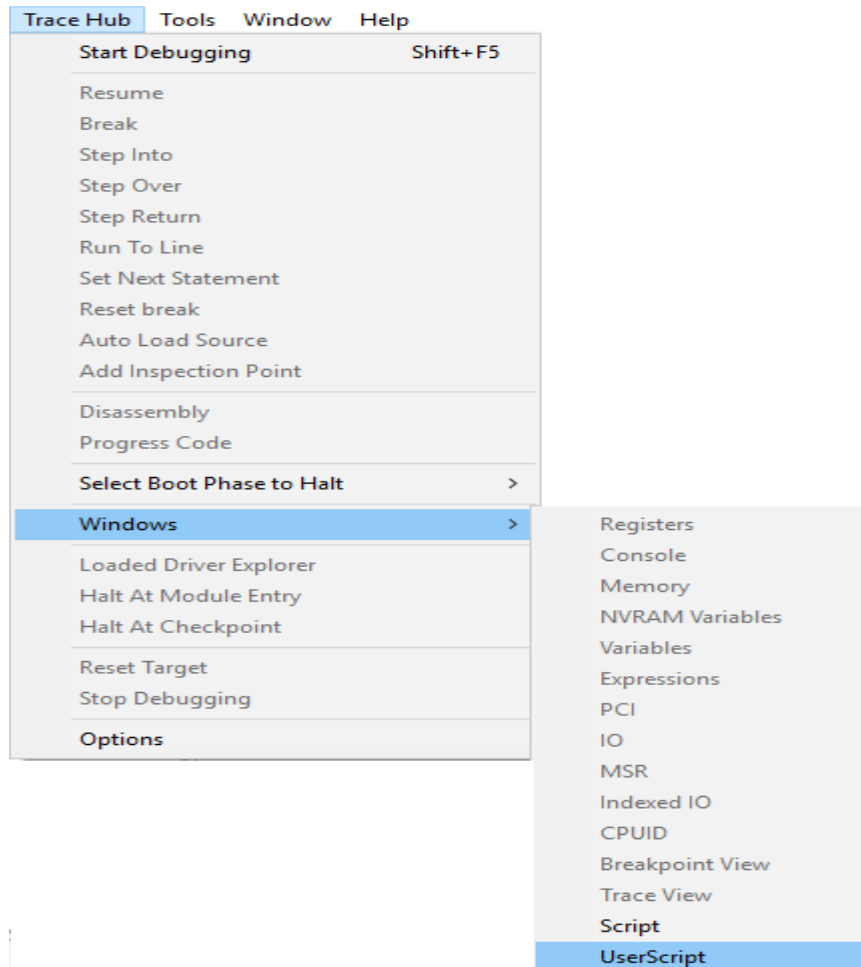
Debugger also supports to execute Scripts provided by silicon vendor.

User can access this using menu: **Trace Hub**→**Windows**→**User Script** or using Toolbar icon [] Script automatically sets up connection with target and executes commands.

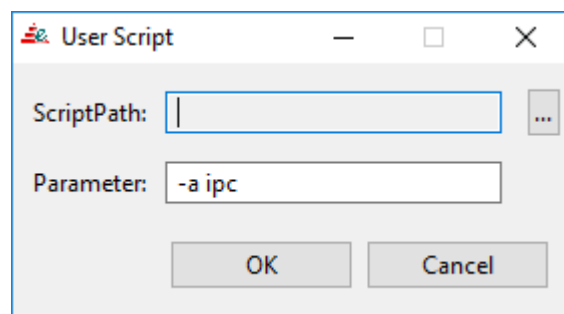
Working with CScripting

Ami Trace Hub debugger allow users to execute Intel CScripts. Users can execute CScripts using:

- VeB Menu: **Trace Hub**→**Windows**→**User Script**



Selecting the option opens a UserScript dialog as below



Users can give script path and parameter use to point to the scripts that they want to execute. After clicking OK button VeB will automatically launch a python shell and start executing the script.

Note: Executing Intel Cscripts requires user not connected with Debugger and CScripts to perform the connection on their own.

Working with IO View

- Users can access the IO by –
- Select: **Trace Hub → Windows → IO**

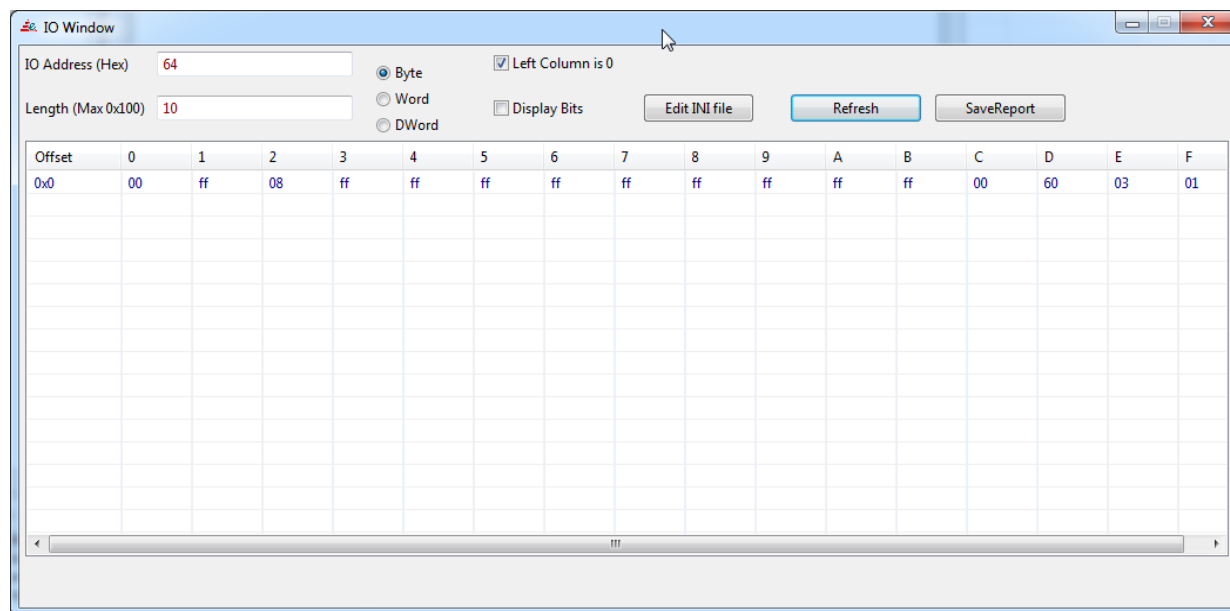


Figure: IO View

Only one static IO Window is possible at a time. Enter the IO Port address of the IO Space and length want to observe in the corresponding Textboxes.

Options

Edit - User can select the desired offset from the grid and edit it, select something outside the grid or press Enter to write the edited value to Target.

Display - The IO Space Display format can be changed by selecting Radio buttons Byte, Word, Dword. Displays in Big Endian Format.

Display Bits - displays the values in Binary format.

NOTE: Only one IO Window is possible at a time, it is not Auto updatable while stepping, user need to Refresh button.

Save Report - Dumps the data to a Text file.

- Users can access the IIO by –
- Select: **Trace Hub** → *Windows* → *IIO*

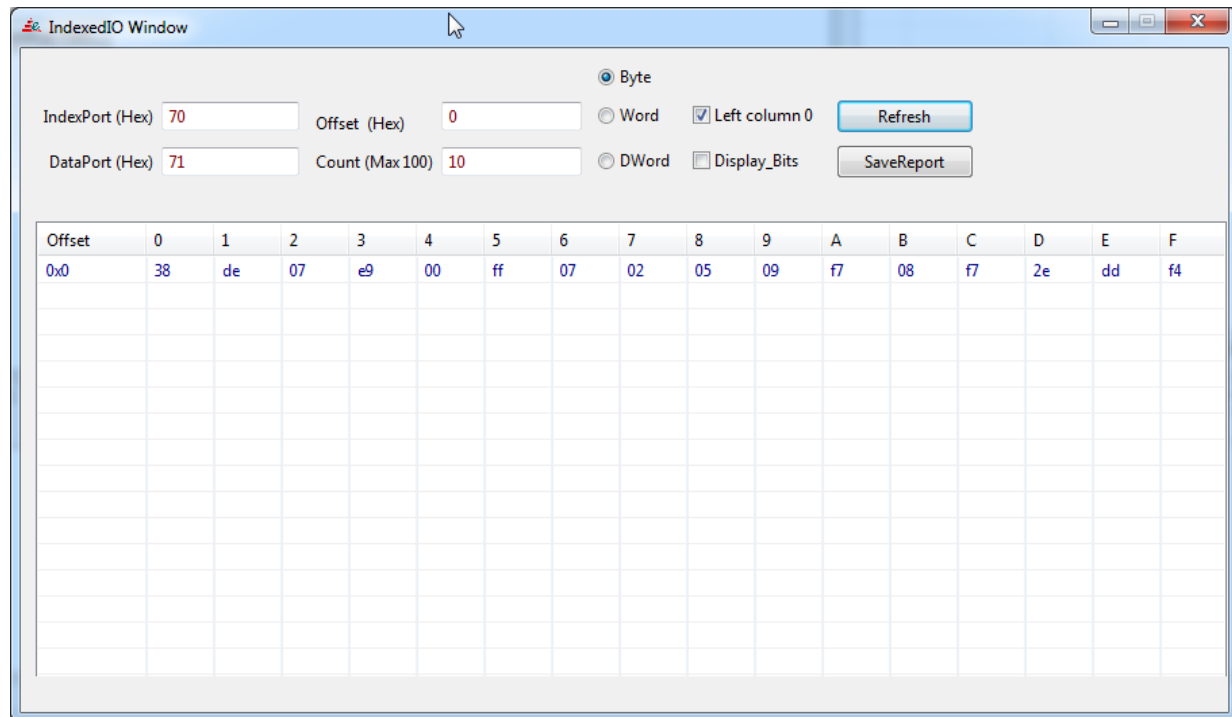


Figure: IIO View

Enter "Index Port" and "Data Port" of the IIO Space and Offset and Size want to observe in the corresponding Textboxes. Data can be modified by mouse click on respective location. The same modification will reflect on target side also. To change the format of display use Radio Buttons Byte, Word and Dword

Working with MSR View

- Users can access the MSR by –
- Select: **Trace Hub → Windows → MSR**

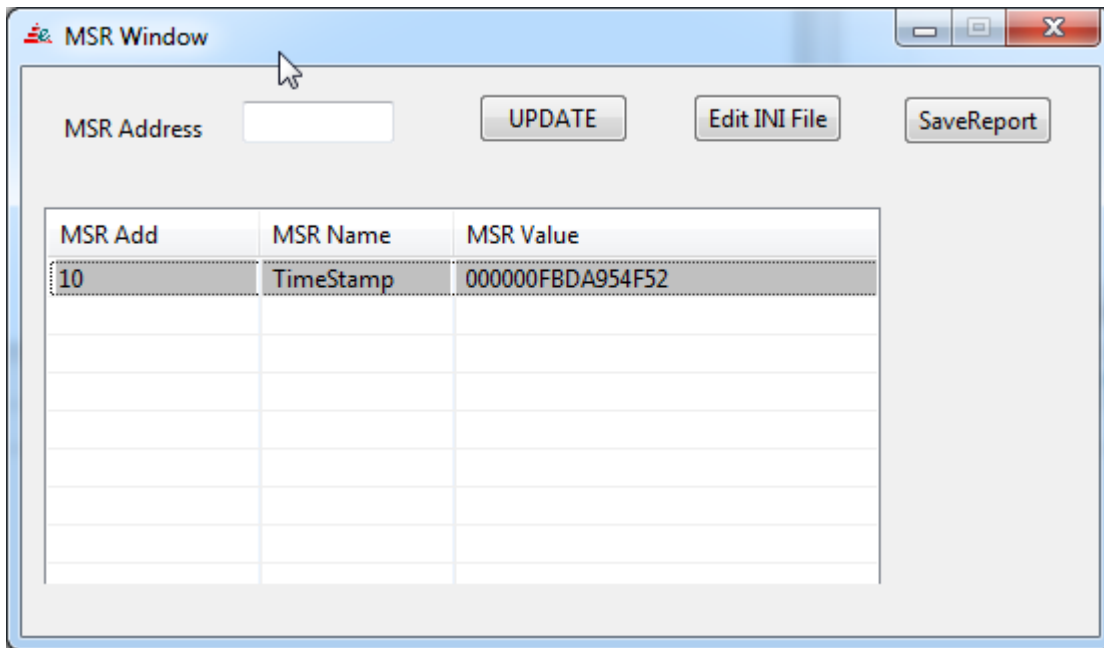


Figure: MSR View

Only one static MSR Window is possible at a time.


On opening, the MSR window, the details of the MSR already present in the MSR.ini file will be displayed.

Users can add new entry of MSR in MSR window by entering address in MSR address in text field and click update button. If the entered MSR is valid then new entry will be updated in MSR.ini file. 'Edit NI file' button helps users to Edit the MSR.ini in the VeB editor, INI file has to be modified according to the set of rules mentioned in the INI file. User can edit the values of a Writable MSR in MSR window.

Working with PCI

- Users can access the PCI by –
- Select: **Trace Hub → Windows → PCI**



Or using the Trace Hub Toolbar Icon []

Users need to first Scan PCI to populate the available PCI List. Clicking on 'Scan PCI' button will list the available PCI devices BDF, Vendor ID and Dev ID.

In From and To text boxes user can specify the bus numbers range used to Scan PCI. The PCI List will display the corresponding available PCI devices information.

Selecting a BDF entry in the PCI List will display the PCI device's Configuration Space.

This is 256 bytes that are addressable by knowing the 8-bit PCI bus, 5-bit device, and 3-bit function numbers for the device (commonly referred to as the *BDF* or *B/D/F*, as abbreviated from *bus/device/function*).

NOTE: Working of this PCI is like software debugger except that memory mapped option will be disabled as it is currently not supported.

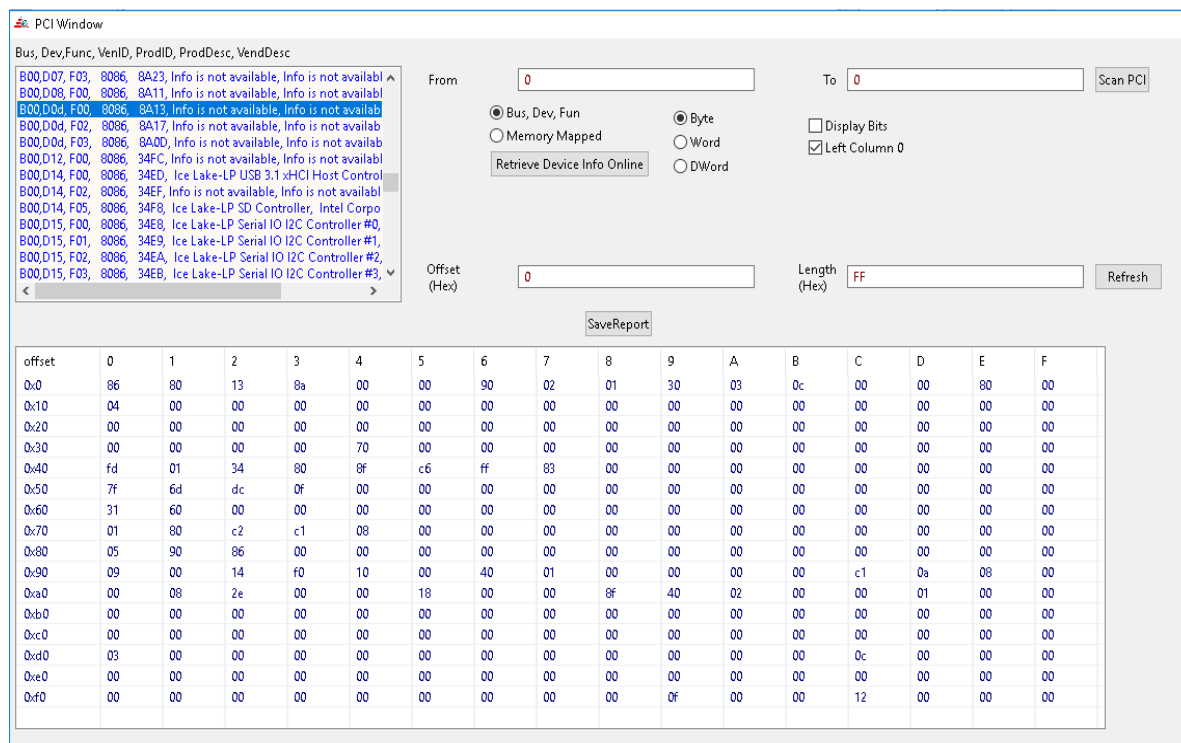


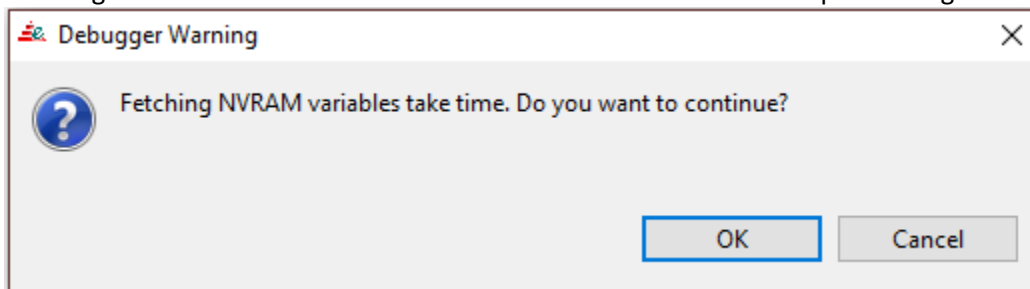
Figure: PCI View

Getting NVRAM Variables Information

Select: **Trace Hub** → **Windows** → **NVRAM** variables Or using the Trace Debug Toolbar Icon [NV]

Only one static NVRAM Window is possible at a time.

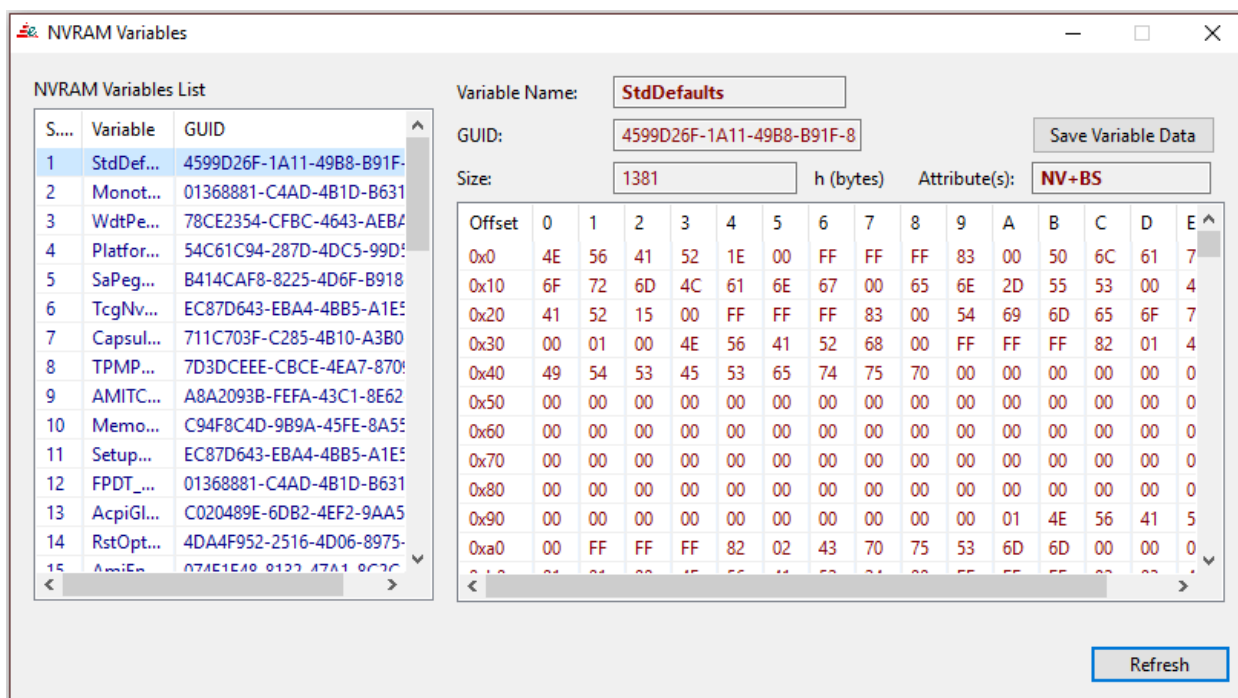
Fetching all NVRAM variables take time. User will be notified before proceeding further as shown below.



If user clicks ok then debugger will open the NVRAM window. In NVRAM window there is NVRAM variable panel present where user can see the list of NVRAM variable with its GUID information.

If user want to see the content of specific NVRAM variable then user has to double click on that variable name.

After double click, right side of fields in the NVRAM window will automatically update with info such as Variable name, Variable GUID, size of variable in bytes, attribute and content of variable. Click on “Refresh” button to get updated value of NVRAM variable.

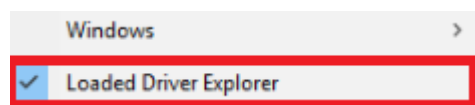



User can save the NVRAM variable data into the file by clicking on “Save Variable Data” button

Working with Loaded driver view

Users can toggle the Loaded Driver Explorer by -

- ☐ Select: **Trace Hub** → **Loaded Driver Explorer**



Or using Trace Debug toolbar icon[]

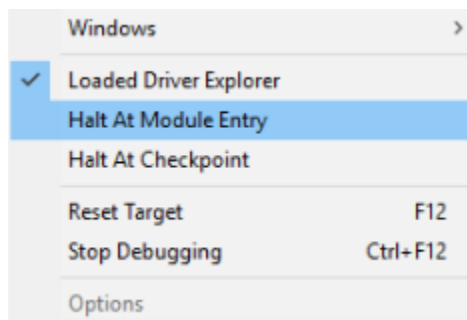
The Loaded Driver Explorer Lists the Drivers in the loaded order they are loaded on the target side. Target module is required to get the loaded driver notification to host and DEBUG_CODE token must be enabled as it is dependent on traces. If traces are not updating

loaded driver view will not update. Listing loaded drivers will be slow as it is dependent currently on trace messages.

Working with Halt at Module Entry

Users can access the Halt at Module Entry window by -

- Select : **Trace Hub→Halt At Module Entry**



- Or using the Trace Debug Toolbar Icon []

In the module selection window, provide the module name and select “ok”. Users can select the type of Halt driver option they want (like All, Next) or select ‘Driver Name’ and type in the driver name to halt on.

Options

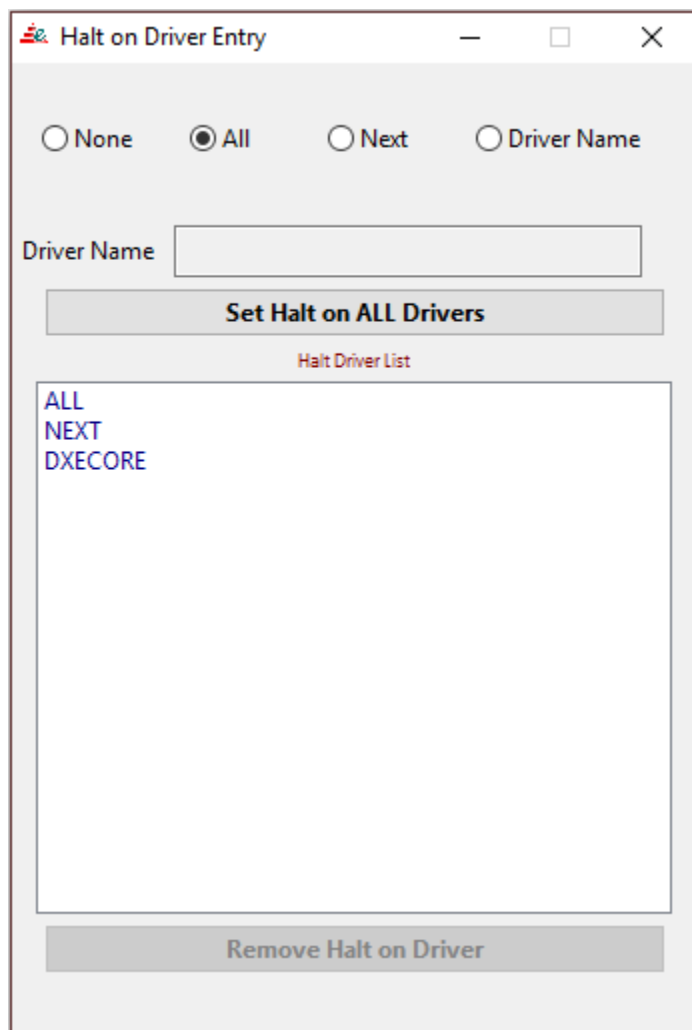
None – Does not halt on any Module Entry, previous halts are all cleared, acts like Halt Driver OFF.

All – Halt at all loaded drivers entry points.

Next - Halts only at next loaded driver’s entry point.

Driver Name - Halts at user specified driver’s Entry Point only, displayed in the Halt Driver List.

For instance, user provides Halt all and in the subsequent entry point user Halt All removed and is modified as Halt Next. Halt module will no longer halt at all modules but will only halt at the next loaded halt module. The behavior is same as Halt Driver->Next.

A screenshot of a software dialog box titled "Halt on Driver Entry". At the top, there are four radio buttons: "None", "All" (which is selected), "Next", and "Driver Name". Below these is a text input field labeled "Driver Name". A large grey button labeled "Set Halt on ALL Drivers" is positioned below the input field. Underneath this button is a section titled "Halt Driver List" in red text. This section contains a list box with three items: "ALL", "NEXT", and "DXECORE", all in blue text. At the bottom of the dialog is another large grey button labeled "Remove Halt on Driver".

NOTE: Halt module names/state are persisted across Target reboots, therefore the user need to disable the Halt if not required during target reboots.

The driver name is the final efi image name such as "CpuPei" (without the quote).

User needs to halt or enable reset break once after the target reset.

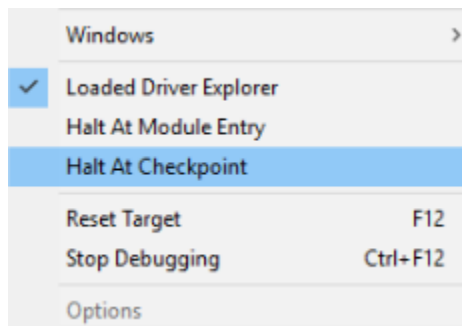
Working with Halt at Checkpoint

The user can configure the Debugger to break on the occurrence of a Checkpoint. The user can set the Debugger to halt on the desired checkpoint using the following options:

- 'ALL' - Debugger will halt on all the Checkpoints
- 'NEXT' - Debugger will halt only on the next loaded Checkpoint.
- 'CHECKPOINT' - Debugger will halt on the Checkpoint provided by the user. Multiple Checkpoints can be setup to halt.

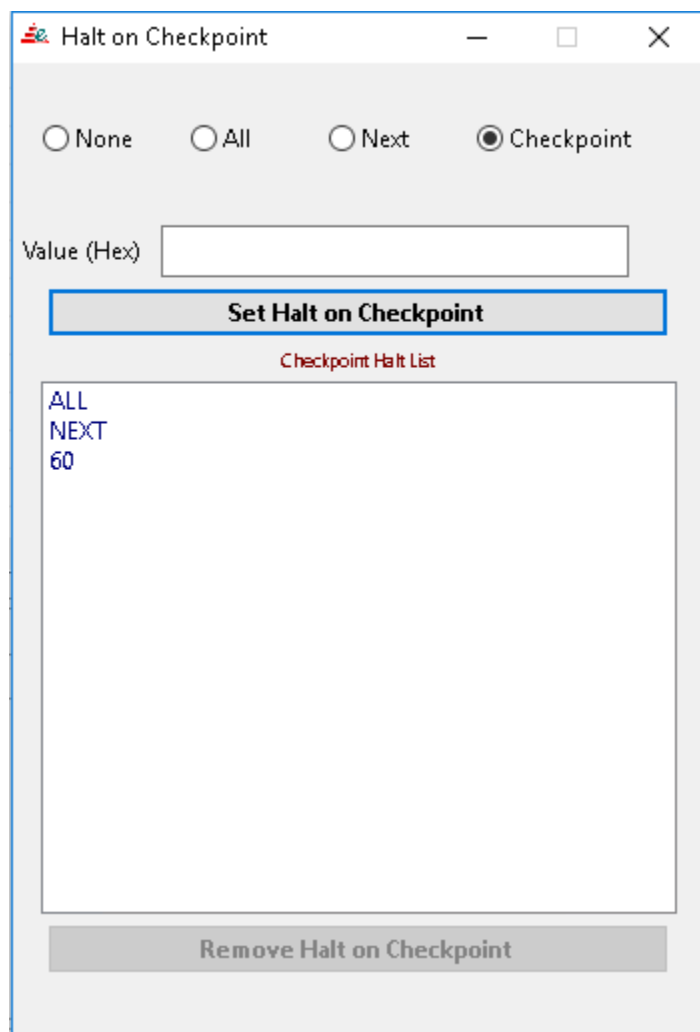
Users can access the Halt At Checkpoint by -

- Select: **Trace Hub** → **Halt At Checkpoint**



- Or using the Trace Debug Toolbar Icon []

In the Halt Checkpoint View, user can specify to halt the target on the occurrence of a Port 80 checkpoint update.



Options

None - Does not halt on any Checkpoint, or Clears existing checkpoint halts.

All - Halts on the occurrence of all checkpoints

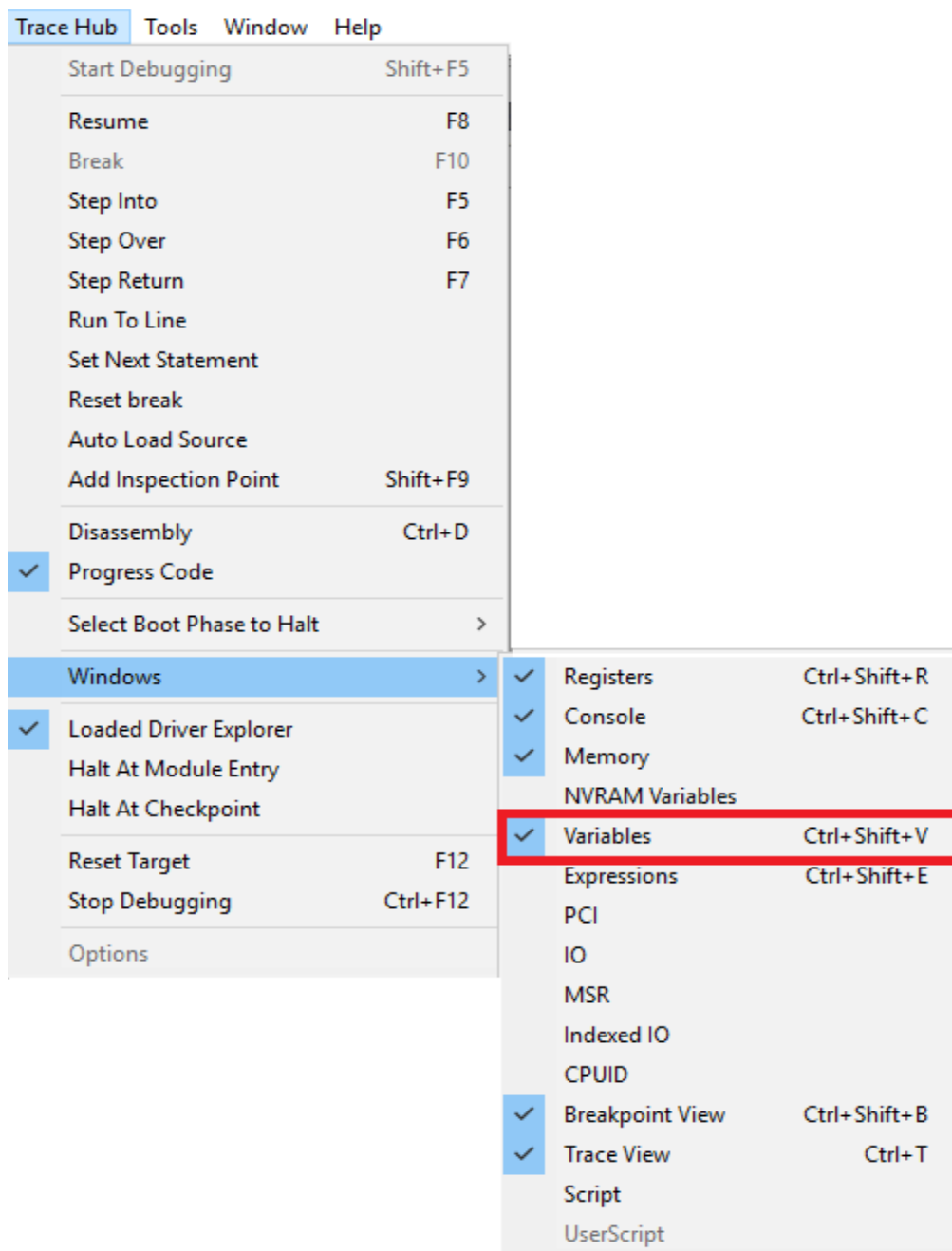
Next - Halts at next Checkpoint only, then Checkpoint Halt 'Next' is cleared.

Checkpoint - Halts at user specified Checkpoints only.

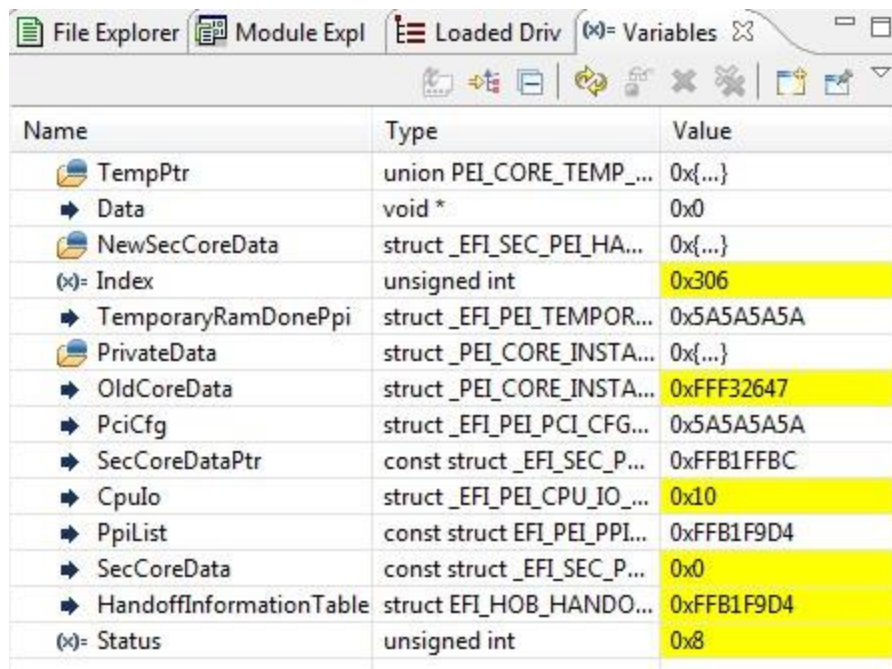
NOTE: Halt Checkpoint names/state are persisted across Target reboots, therefore the user needs to disable the checkpoint halt if not required during Target reboots.

Working with Variables View

Users can access the Variables View by - Select : **Trace Hub**→**Windows**→**Variables**



- Or using Hotkey: **CTRL+SHIFT+V**



Name	Type	Value
TempPtr	union PEI_CORE_TEMP_...	0x{...}
Data	void *	0x0
NewSecCoreData	struct _EFI_SEC_PEI_HA...	0x{...}
Index	unsigned int	0x306
TemporaryRamDonePpi	struct _EFI_PEI_TEMPOR...	0x5A5A5A5A
PrivateData	struct _PEI_CORE_INSTA...	0x{...}
OldCoreData	struct _PEI_CORE_INSTA...	0xFFFF32647
PciCfg	struct _EFI_PEI_PCI_CFG...	0x5A5A5A5A
SecCoreDataPtr	const struct _EFI_SEC_P...	0xFFB1FFBC
CpuIo	struct _EFI_PEI_CPU_IO_...	0x10
PpiList	const struct EFI_PEI_PPI...	0xFFB1F9D4
SecCoreData	const struct _EFI_SEC_P...	0x0
HandoffInformationTable	struct EFI_HOB_HANDO...	0xFFB1F9D4
Status	unsigned int	0x8

The variables window displays only the local variables in the current scope. The variable window contains three columns. The Name, Type and Value columns.

Name column displays the name of each local variable. If a variable has children (like with structures etc), the variable window will provide that variable in a tree view that can be expanded\collapsed.

Value column shows value of that variable in Hexadecimal format.

Type column shows the type of each variable.

Each variable is displayed in the format that is proper for its own data type. Data structures have their type names in the Type column.

NOTE: Variable View will auto-update on every step or halt.

If the value of a variable is changed since its last update, the value will be highlighted.

Editing a Variable

Users can edit the variable's value by selecting the value under Value Column and edit it.

NOTE: Only Hexadecimal inputs are allowed.

Additional Options

Users can see the Memory content and Edit it using the Memory window, to do that users need to right-click on the variable and select "view memory" option as in the picture

below. This will use the 'Value' of the variable as the start address for memory display. For more information refer to Memory View.

Watch Variables

If a variable (Parent) is selected for Watch, that variable will be displayed in the Expressions View.

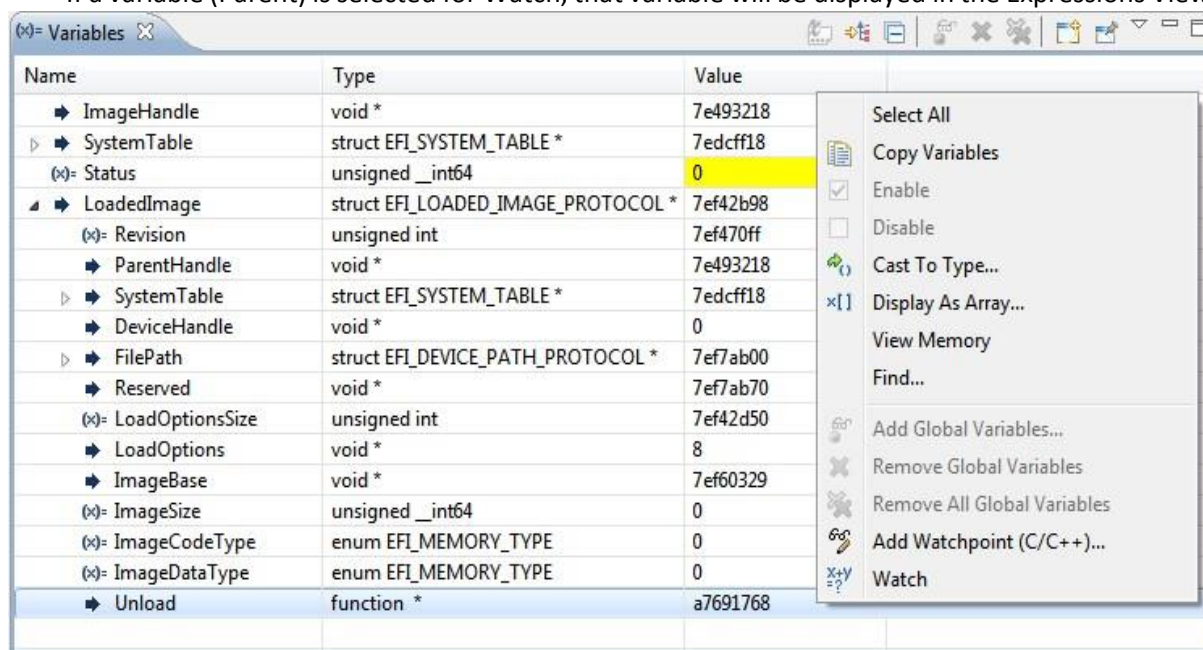


Figure: Variables View.

[NOTE] For the View Memory option, the value of the variable will be taken as the input address to show memory content.

NOTE: To view the Address of the variable, type command 'var' in AptioVDebugger Console. User need to follow these steps to view the address of the variable.

NOTE: Debugger support change any type of variables to an array

User should select the variable which has to change to array.

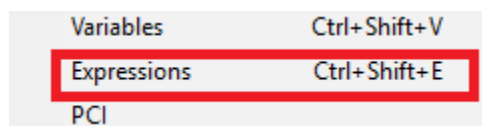
Right click, in pop up menu select "Display As Array". In pop up dialog enter start index and length.


NOTE: If variable window is not listing all variable, then make sure that OPTIMIZATION is off.


Working with Expressions (Local and Global Variables)

Users can access the Expressions View by -

- Select: **Trace Hub ->Windows->Expressions**



- Or using Hotkey: CTRL+SHIFT+E
- Or using the Debug Toolbar Icon []

This will open the Expression view Window. Global Variables, Specific Variable Data can be inspected in the Expressions View. It acts as a watch window to observe global variables. Expressions view is similar to the Variables view, except in Expressions the Variable needs to be entered by selecting []. User can add new expression of global variable or local variable to the expression view.

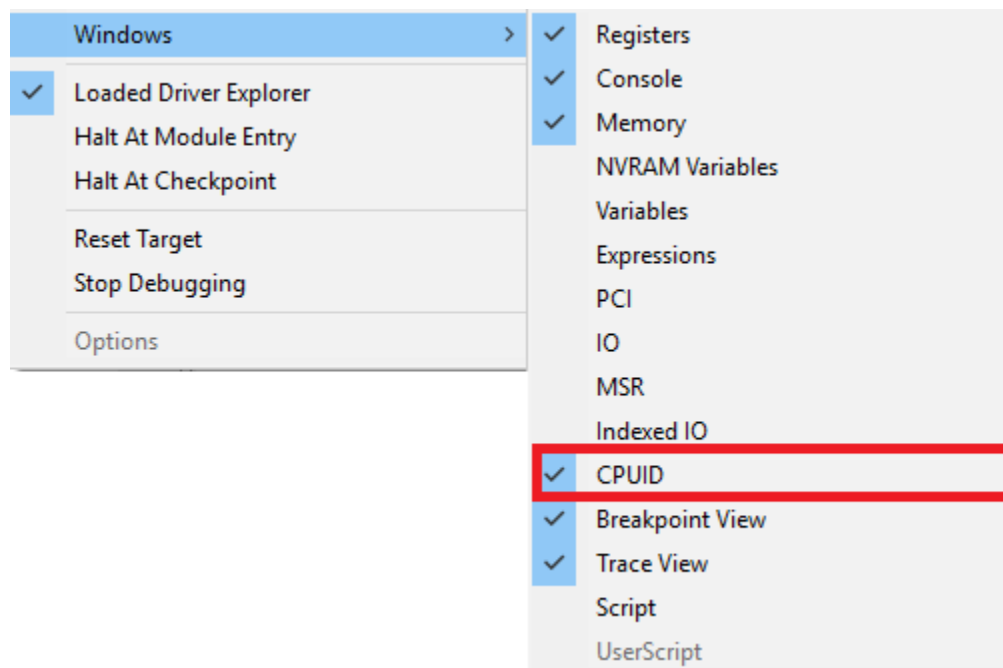
Expression	Type	Value
gDxeCoreRT	struct EFI_RUNTIME_SERVICES *	7edcfe18
gDxeCoreST	struct EFI_SYSTEM_TABLE *	7edcff18
Hdr	struct EFI_TABLE_HEADER	{...}
FirmwareVendor	unsigned short *	0
FirmwareRevision	unsigned int	0
ConsoleInHandle	void *	0
ConIn	struct _EFI_SIMPLE_TEXT_INPUT_PROTOC...	0
ConsoleOutHandle	void *	0
ConOut	struct _EFI_SIMPLE_TEXT_OUTPUT_PROTO...	0
StandardErrorHandle	void *	0
StdErr	struct _EFI_SIMPLE_TEXT_OUTPUT_PROTO...	0
RuntimeServices	struct EFI_RUNTIME_SERVICES *	0
BootServices	struct EFI_BOOT_SERVICES *	7ef7c4f0
NumberOfTableEntries	unsigned __int64	0
ConfigurationTable	struct EFI_CONFIGURATION_TABLE *	0
GRP(General Registers).REG(RBX)	Unsigned / Readable,Writeable	7EF432E0
+ Add new expression		


Any Watch's selected from Variable, register window will be displayed in the Expressions View.

NOTE: The scope of the Global Variable is the Module's range, but Local variable scope is the function only.

Displaying CPU Information

Select: **Trace Hub->Windows->CPUID**



or select the Trace Debug Toolbar Icon ]

Only one static CPUID Window is possible at a time. This window displays the details of the Target CPU.

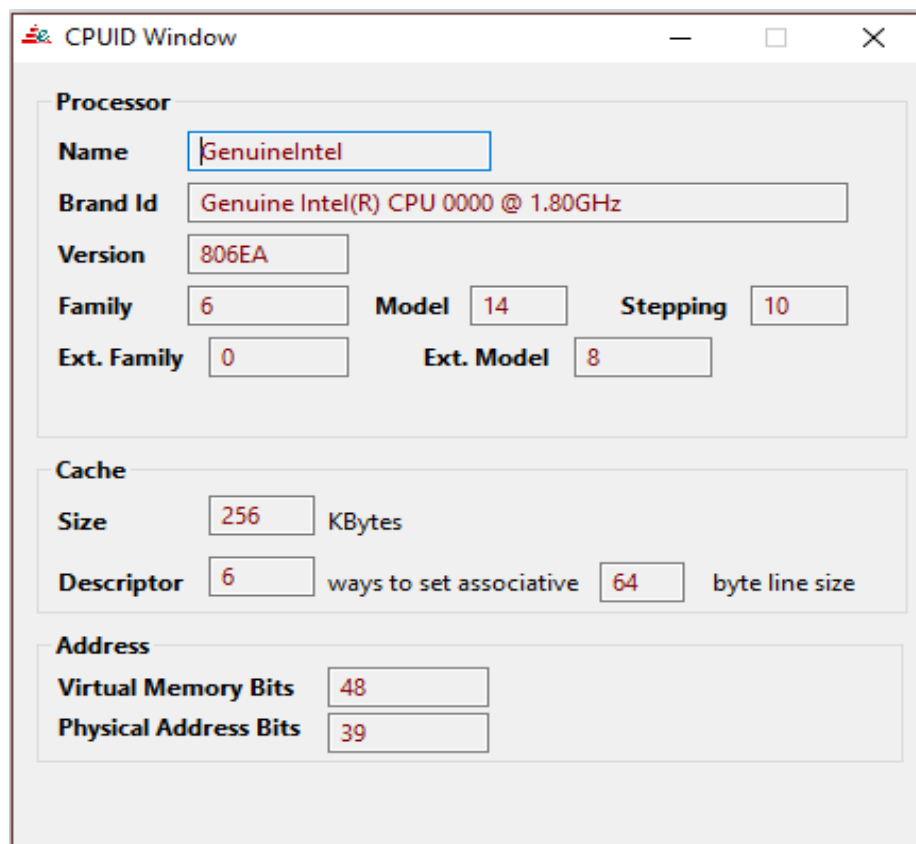
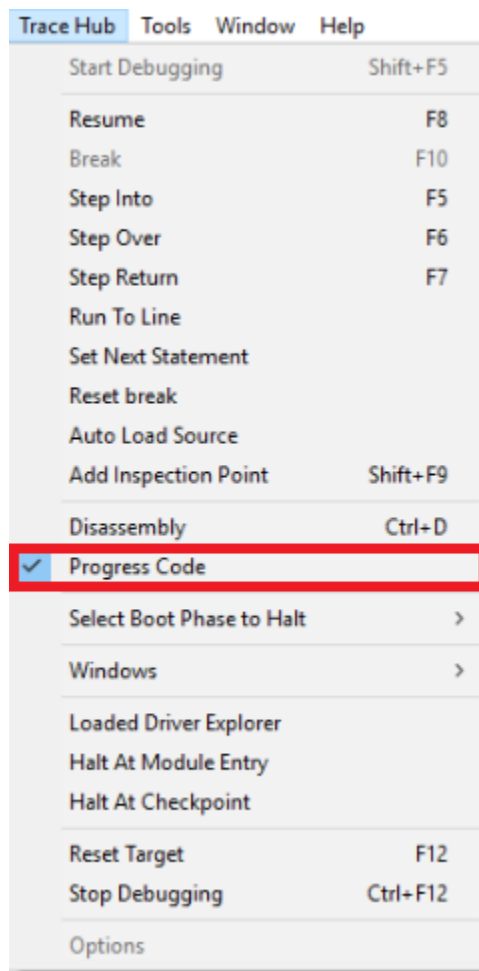


Figure: CPUID View

Working with Progress Code View

Users can toggle the Progress Code View by -

- Select: **Trace Hub -> Progress Code**

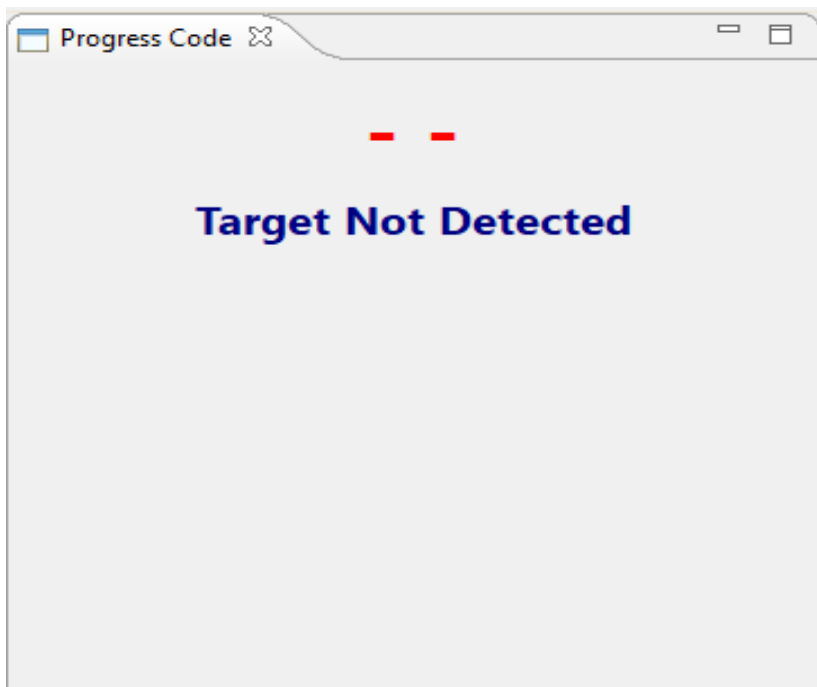


Or using the Trace Debug Toolbar Icon []

The Progress code view will display the Progress code Checkpoints while Target is in Running\Booting state and will display the Port 80 Checkpoint and UEFI progress code list When Target is in Halted State. Checkpoints when halted will be displayed in RED and while running will be displayed in GREEN and UEFI progress code list will be in black color.

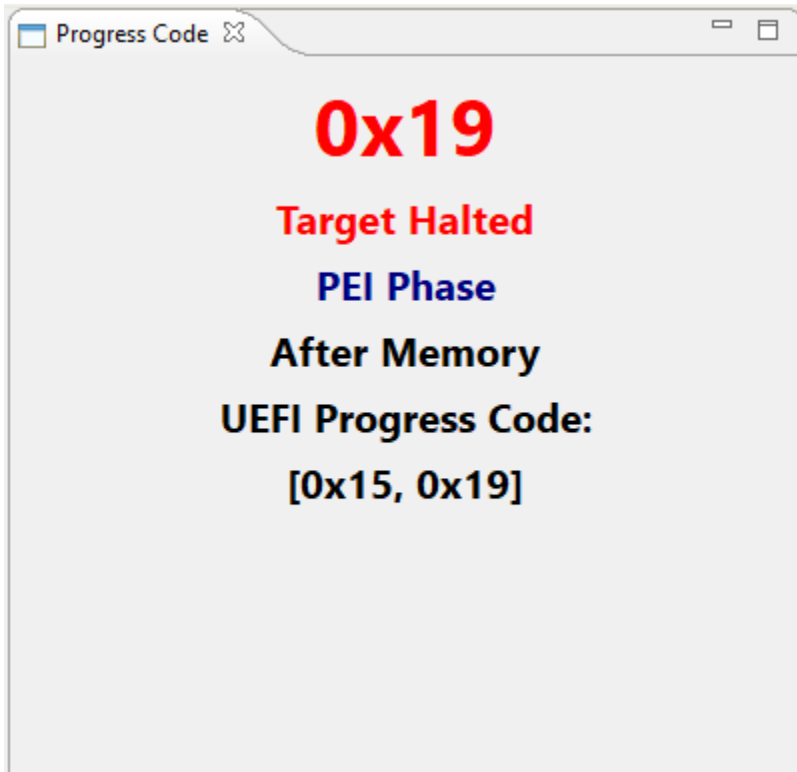


When Target is not connected checkpoint window will display with message as "Target Not Detected". Below screenshot showing when target not detected.



Additional Target Info

The progress code view will display the state of target, phase of Bios (such as PEI phase, DXE phase or SMM phase) and display the relevant info in each phase, such as in PEI phase - current execution is before memory or after memory



Note:


Checkpoint, Phase Info and UEFI progress code will get displayed only when user enable the reset break or halt once after reset the target.

For last some drivers, user need to do halt to get the updated progress code in progress code view.

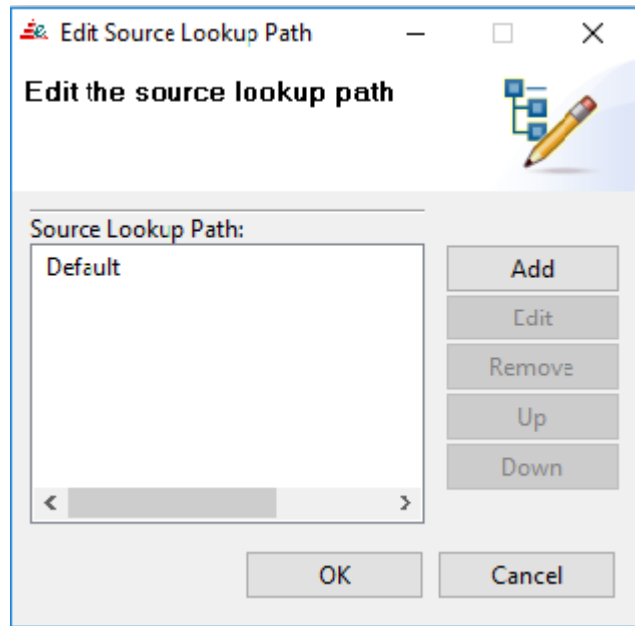
Working with Source Lookup path Editor

Source Lookup path Editor is used to change the source lookup path, to find the source file, when the bios build path is different from the current project path. User need to provide the build folder full path in the project.

Users can access the Source Editor View by -

- Select: **Trace Hub -> Windows -> Source Editor**
- or select the Trace Debug Toolbar Icon 

When user select the Source Editor Menu then “Edit Source lookup path” dialog will get open



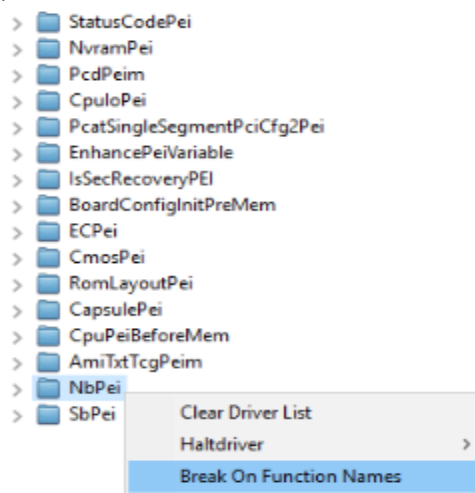
Note:

User need to add the project path up to build folder and after adding the path, source will get displayed only after the next module gets loaded or during reset.

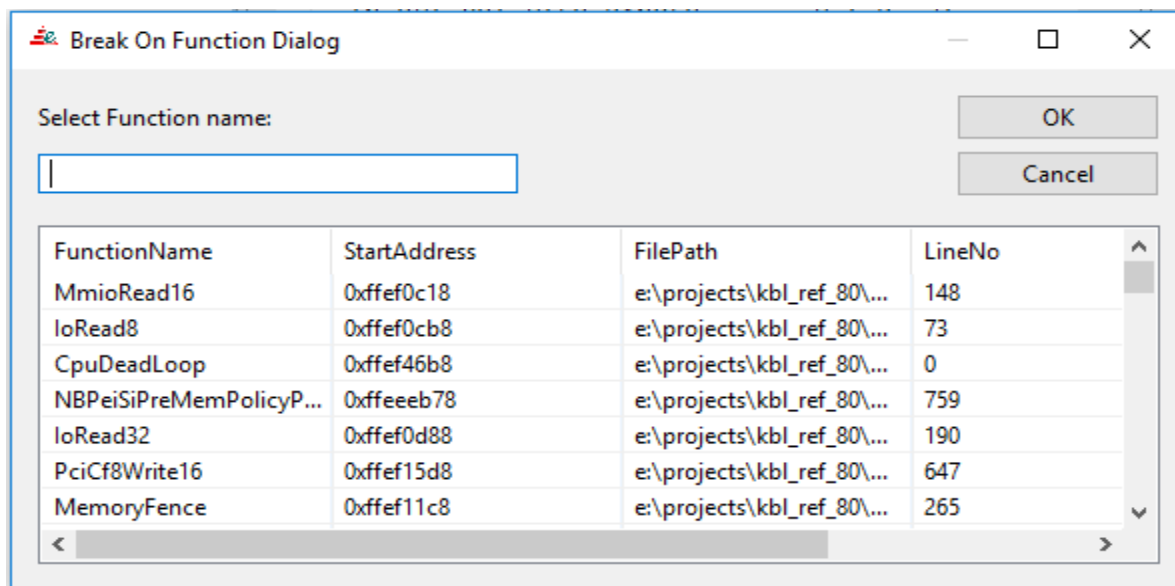
Auto source load icon may need to be enabled, if loaded module notification is not received.

Working with Break on Function Names

Break on Function Names is used to add breakpoint on the function entry when only the pdb path is available.

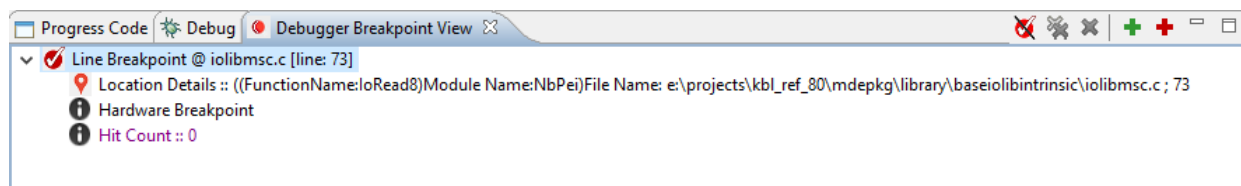


Users can access the Break on Function Names dialog by right clicking the modules that are listed in loaded driver view.




User can sort Function Name, Start Address and File Path by clicking the headings.

After Click ok, User can see the added breakpoint with function name, Module name, file name and line number in Debugger Breakpoint View as below




Execution Control

Resume


If process is halt at breakpoint, you can resume the execution by selecting Resume on Trace Hub 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 Trace Hub 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 Trace Hub menu or select Trace Hub tool bar icon  or the hotkey **F12** to reset the target


Stepping Controls

Step Into


Click Step Into on the Trace Hub menu or select Trace Hub 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 Trace Hub menu or select Trace Hub 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.

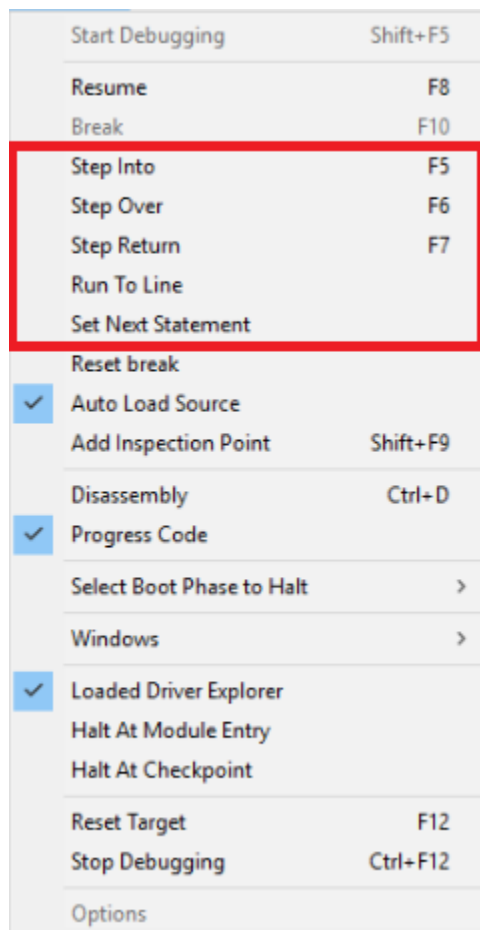
Step Return

Click Step Return on the Trace Hub menu or select Trace Hub 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 Trace Hub 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 run 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**).



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 Trace Hub Menu or select Trace Hub 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.

Understanding Breakpoint vs Inspection Point

Breakpoint - is a point at which the Debugger will halt the execution on the Target.

- * If the Breakpoint has Conditions associated with it (See Conditional breakpoints), the Breakpoint will halt the execution only if all the Conditions are satisfied.

- * If the Breakpoint has Inspections associated with it (See Inspection Points), the Inspection Options will be displayed on the AptioVDebugger Console window and the execution will be halted.

Inspection Point - is a point at which the Inspections options are displayed on the AptioVDebugger Console window, but the execution of the Target will NOT be halted.

Breakpoints - Setting & Usage

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

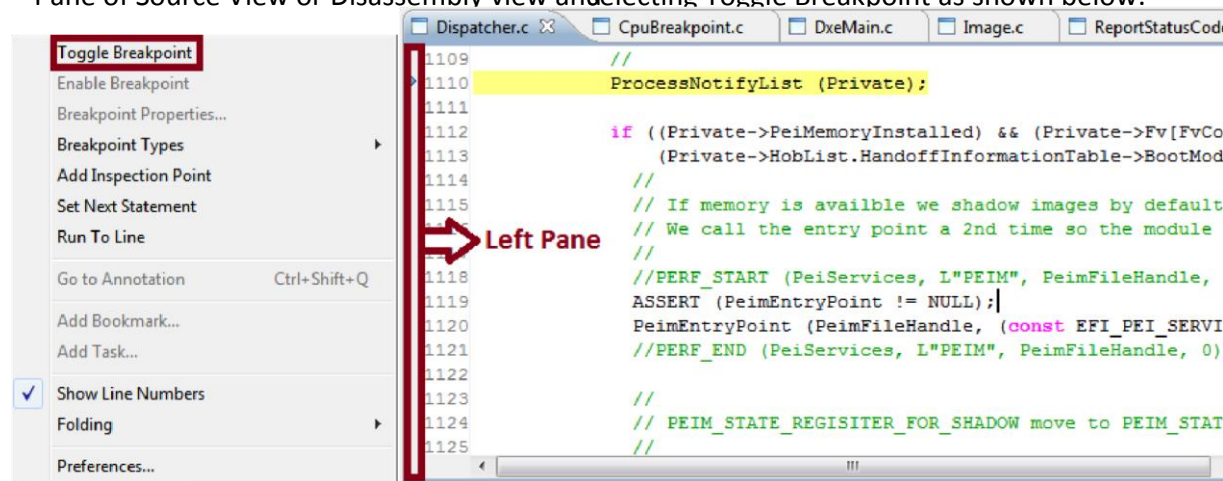
Breakpoint @ Compile Time (In Code)

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.



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.



Software Breakpoint will not be applied unless the target platform is halted at the module entry.

The user will be displayed the following message will be whenever user try to place software break point.


"Please note the Software Breakpoint created will not be applied after a platform reset unless the target platform is halted at the module entry"

While resuming the target if there are any inactive break points, then following message will be displayed in AptioV Debugger console with the list of inactive breakpoints.

"The following software breakpoints will not halt the target platform since the modules associated with them are yet to be loaded.

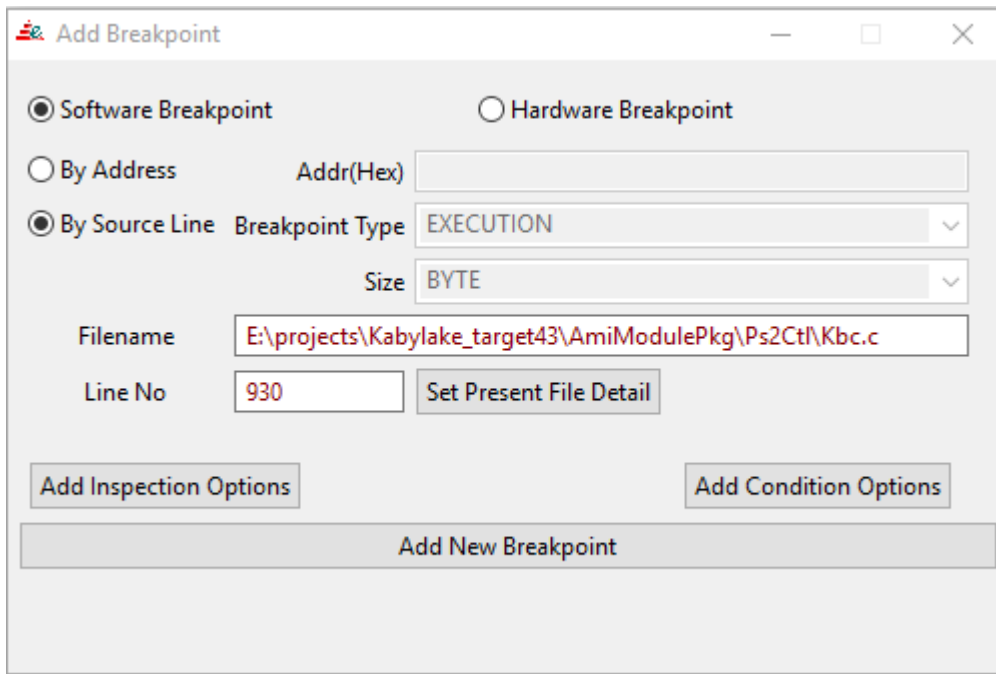
Setting a Custom Breakpoint

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: **Trace Hub** → **Windows** → **Breakpoint View**
- Toolbar: using icon []

This will open the "Add Breakpoint" window which contain options to Set Hardware/Software breakpoint

- With Conditions (Optional)
- With Inspection Options (Optional)



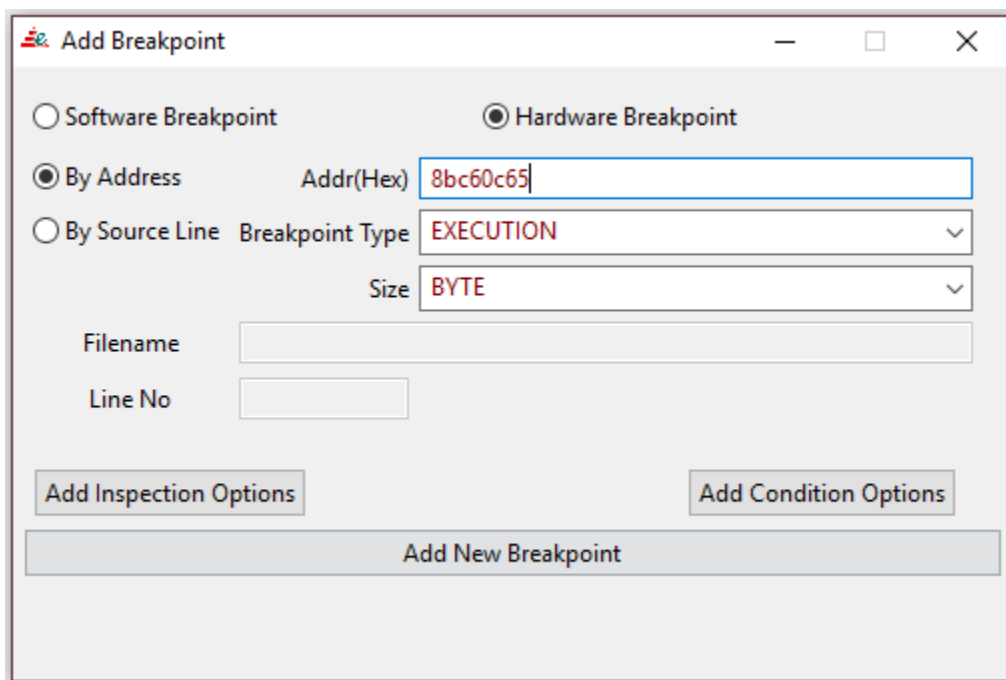
The 'Add Breakpoint' dialog box is shown. It has two main sections: 'Software Breakpoint' (selected) and 'Hardware Breakpoint'. Under 'Software Breakpoint', there are two options: 'By Address' and 'By Source Line' (selected). The 'By Source Line' section includes a 'Breakpoint Type' dropdown set to 'EXECUTION' and a 'Size' dropdown set to 'BYTE'. The 'Filename' field contains 'E:\projects\Kabylake_target43\AmiModulePkg\Ps2Ctl\Kbc.c' and the 'Line No' field contains '930'. There is a 'Set Present File Detail' button. At the bottom, there are two buttons: 'Add Inspection Options' and 'Add Condition Options', and a large 'Add New Breakpoint' button.

Adding a Custom HW Breakpoint

Custom HW break points can be set by selecting the hardware breakpoint option in the Custom breakpoint view. Hardware Breakpoint can be set By Address or By Source Line.

HW Breakpoint: @ Address

When setting Hardware Breakpoint by Address, the 'By Address' radio button needs to be selected, The Address, breakpoint type and the size need be entered. Then click on "Add New breakpoint" window. These break points are triggers whenever the Address specified is being accessed based on the breakpoint type specified.



The supported breakpoint types are

EXECUTION – This type of breakpoint will trigger on execution of specified memory address, that is when the specified memory address is the current Instruction Pointer (IP).

DATA WRITE – This type of breakpoint will trigger only when writing data at specified memory address or changing the content present in specified memory address.

IO READ ONLY – This type of breakpoint will trigger when reading from specified IO port address happen.

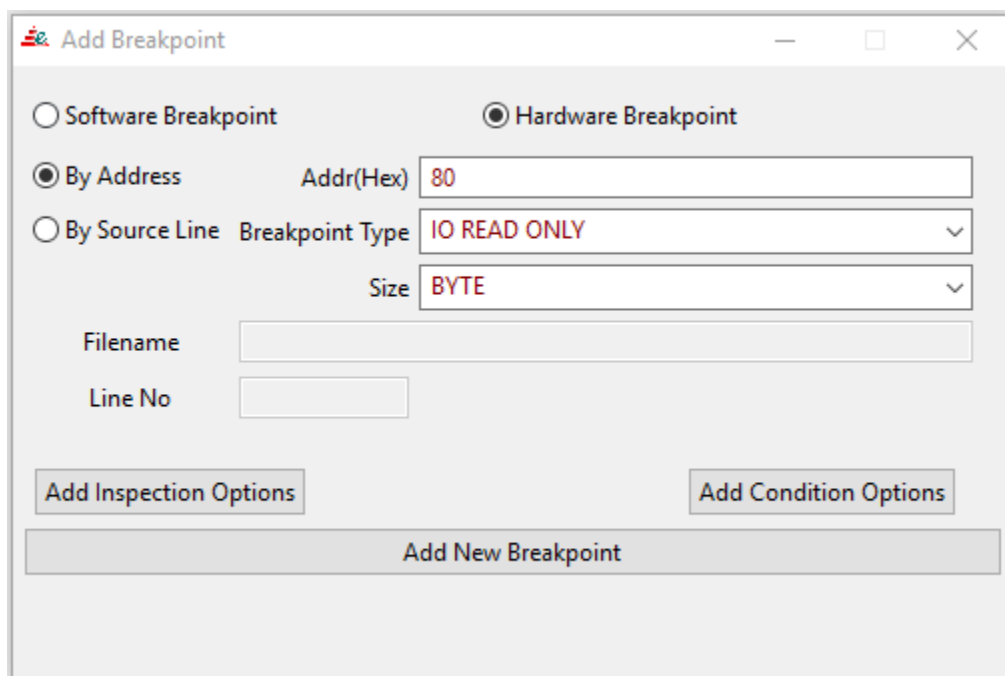
IO WRITE ONLY – This type of breakpoint will trigger only when writing data to the specified IO Port address.

DATA READ AND WRITE NO EXEC – This type of breakpoint will trigger whenever reading or writing of data at specified Memory address happens, But not when the Address is the Current Instruction. User can see the breakpoints set in breakpoint view window.

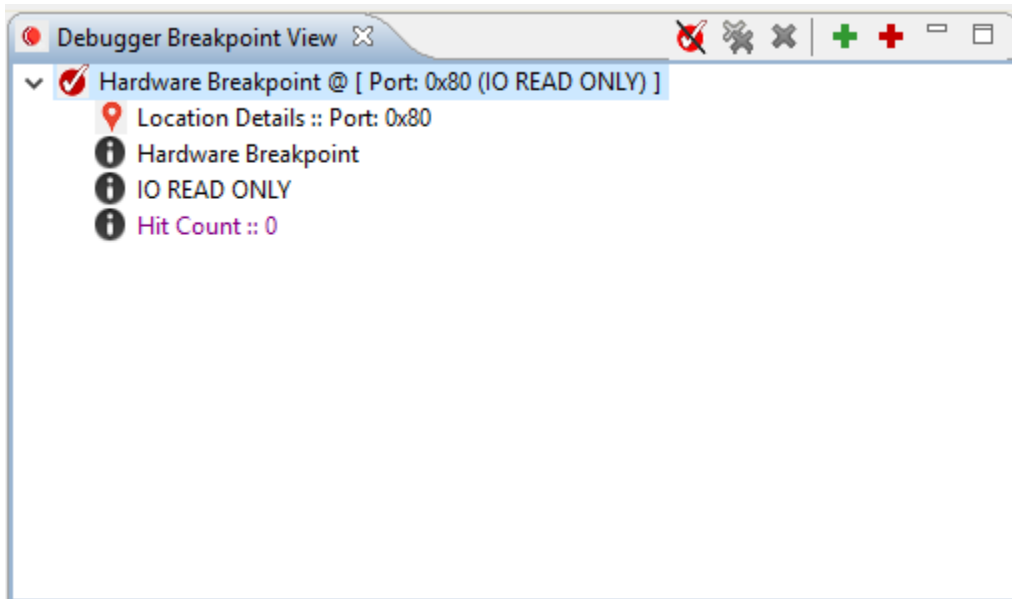
Users can also add the breakpoint from Breakpoint view window by clicking on Red plus symbol Refer Breakpoint View.

IO HW Breakpoint Example:

To set an IO READ ONLY Hardware breakpoint at IO Port 0x80: Select hardware breakpoint, address as 0x80, Breakpoint type as IO READ ONLY and size as BYTE.

A screenshot of the "Add Breakpoint" dialog box in the Trace Hub Debugger. The dialog has a title bar with a red bug icon and the text "Add Breakpoint". Inside, there are two radio buttons: "Software Breakpoint" (unselected) and "Hardware Breakpoint" (selected). Under "Hardware Breakpoint", there are two options: "By Address" (selected) and "By Source Line" (unselected). The "By Address" option has a text field labeled "Addr(Hex)" containing the value "80". Below this, there are two dropdown menus: "Breakpoint Type" set to "IO READ ONLY" and "Size" set to "BYTE". At the bottom, there are two empty text fields labeled "Filename" and "Line No". Below these fields are two buttons: "Add Inspection Options" and "Add Condition Options". At the very bottom is a large button labeled "Add New Breakpoint".

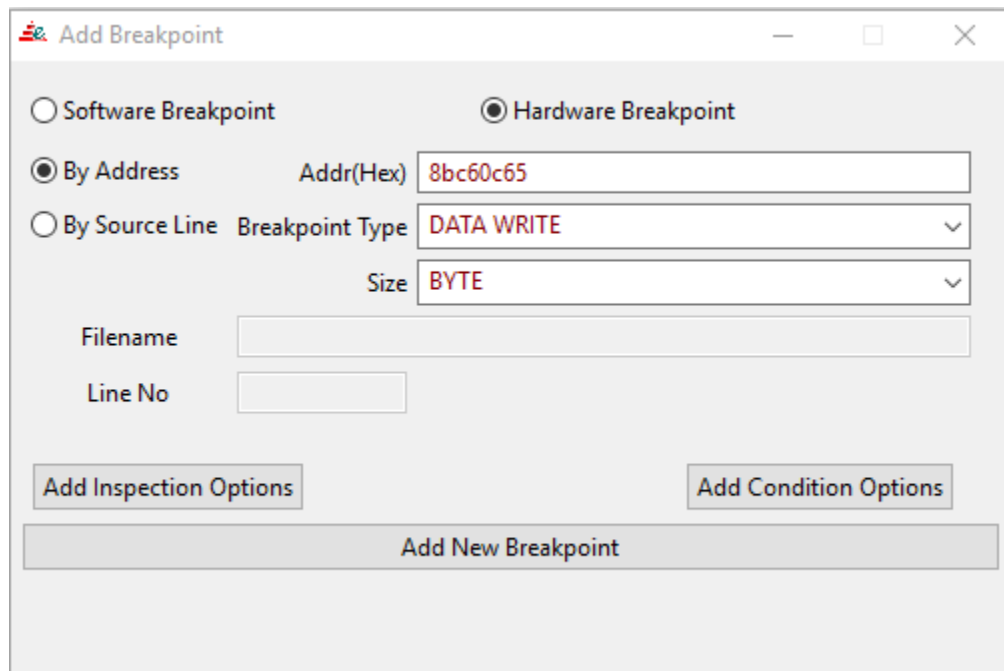
Click on "Add New Breakpoint" button. You can see the breakpoint is set in breakpoint view window.



DATA WRITE HW Breakpoint Example:

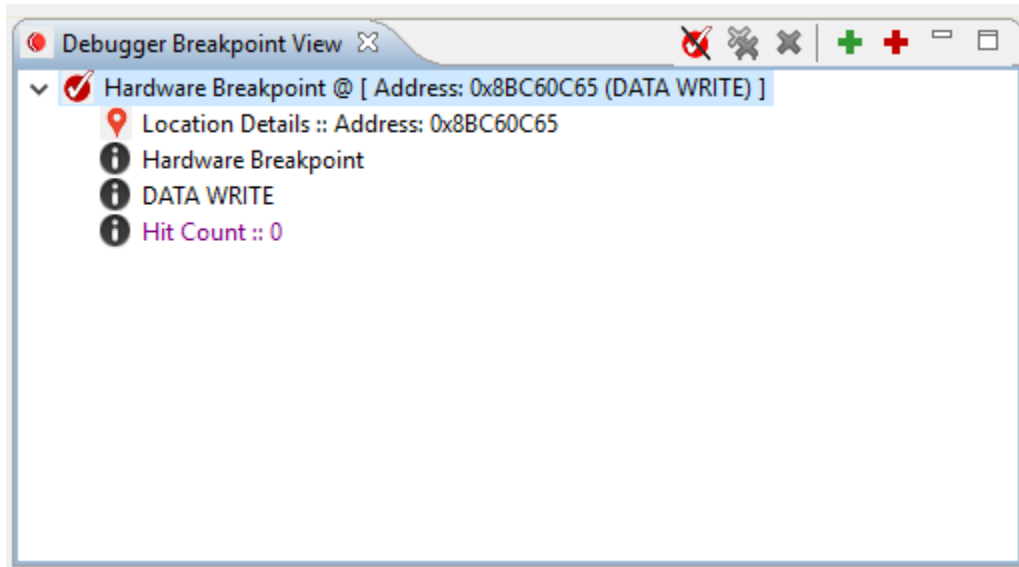
If you want to set the breakpoint, which must trigger whenever someone changing the first two-byte content at memory address 0x00FFB180 then follow the below steps.

Select hardware breakpoint, address as 0x00FFB180, Breakpoint Type as DATA WRITE and size as WORD.



Click on “Add New Breakpoint” button.

You can see the breakpoint is set or not in breakpoint view window.



HW Breakpoint: @ SourceLine

When setting Hardware Breakpoint by Address, the ‘By Address’ radio button needs to be selected, The Address, breakpoint type and the size need be entered.

User can click on the source line required in source view and use the ‘Set Present File Detail’ to populate the Filename and Line No fields.

Adding a Custom SW Breakpoint

Custom SW break points can be set by selecting the software breakpoint option in the Custom breakpoint view. Software Breakpoint can be set By Address or By SourceLine.

SW Breakpoint: @ Address

A Valid Memory Address is the required input for setting a Software breakpoint at an Address

Click on “Add New Breakpoint” button to set the Breakpoint.

SW Breakpoint: @ SourceLine

The filename and line number may be input as parameters if the breakpoint needs to be created using By Source Line option.

User can click on the source line required in source view and use the ‘Set Present File Detail’ to populate the Filename and Line No fields.

Note: Adding Software breakpoint @Address or @Source line in PEI Phase before Memory is initialized, will automatically be set as Hardware Breakpoints, and it should show as hardware breakpoint.

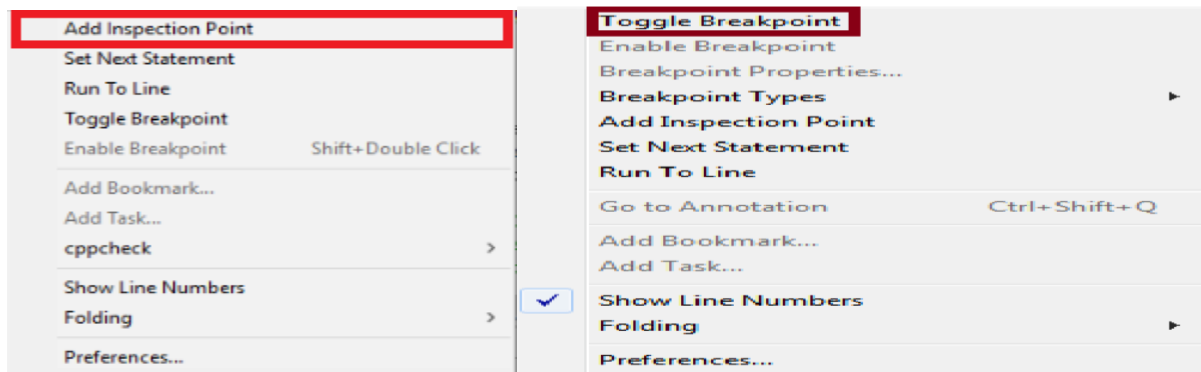
Inspection Points - Setting & Usage

Setting an Inspection point at Boot time can be done using any of the methods mentioned below.

Inspection Point @ Boot Time

Using Editor Pane (Source View)

One way to set an Inspection point on any source line, is by right clicking on the Left Pane of Source View and selecting 'Add Inspection Point', as shown below.



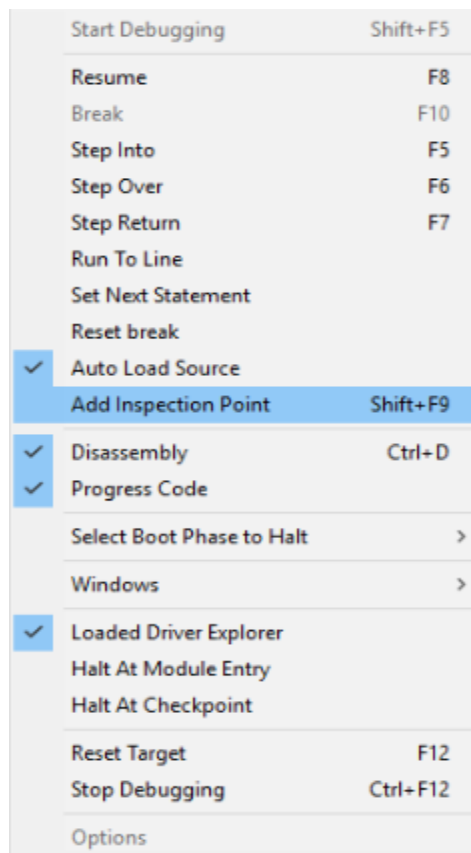
Like breakpoints, using this method, the Inspection points set in PEI phase before Memory initialization will be automatically set as Hardware Inspection points. Software Inspection points will be set only after memory is initialized.


NOTE: When setting Inspection points in PEI phase before Memory is initialized, Debugger will automatically set a Hardware Inspection point using the Debug registers, it is recommend that users do not set more than 3 combined breakpoints + Inspectionpoints since debugger uses 1 Debug register for internal stepping purposes.

Add Inspection Point View

Users can add a new Inspection Point by –

- Select: **Trace Hub → Add Inspection Point**



- Or using Hotkey: SHIFT+F9
- Or using the Trace Debug Toolbar Icon []

Inspection Point is like breakpoint. Using Inspection Point View, user can see and dump the requested information such as PCI data, IO data, Variable data, context Register data and memory data to console window also the same could be log into a file at Inspection Point.

In Inspection Point Window, user can select the inspection point either by selecting “By Source line” or “By Address” option.

By Source line

Enter the file name and line number into corresponding text boxes to set inspection point using “By Source Line” option.

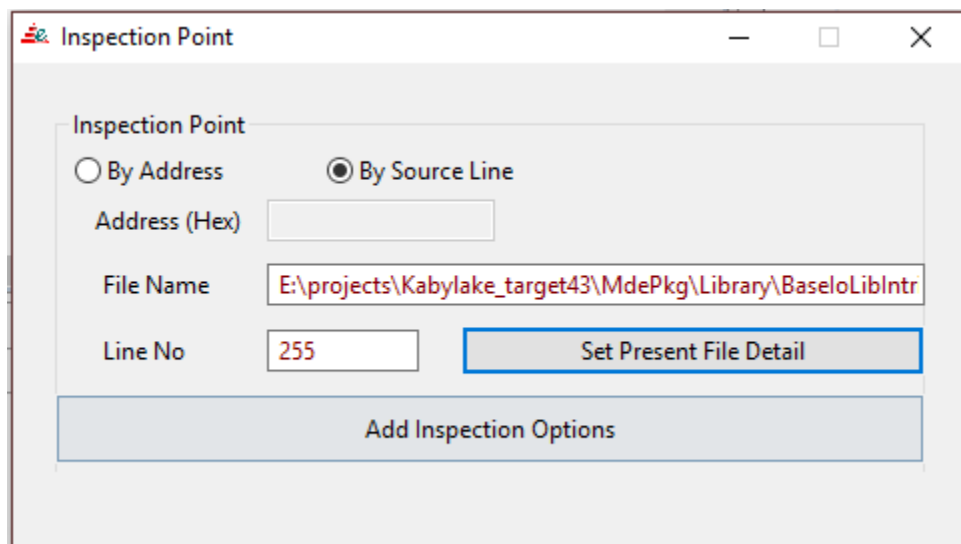


Figure: Inspection point by source line

By Address

Enter the address into “Address” text box at which user wants to inspect data

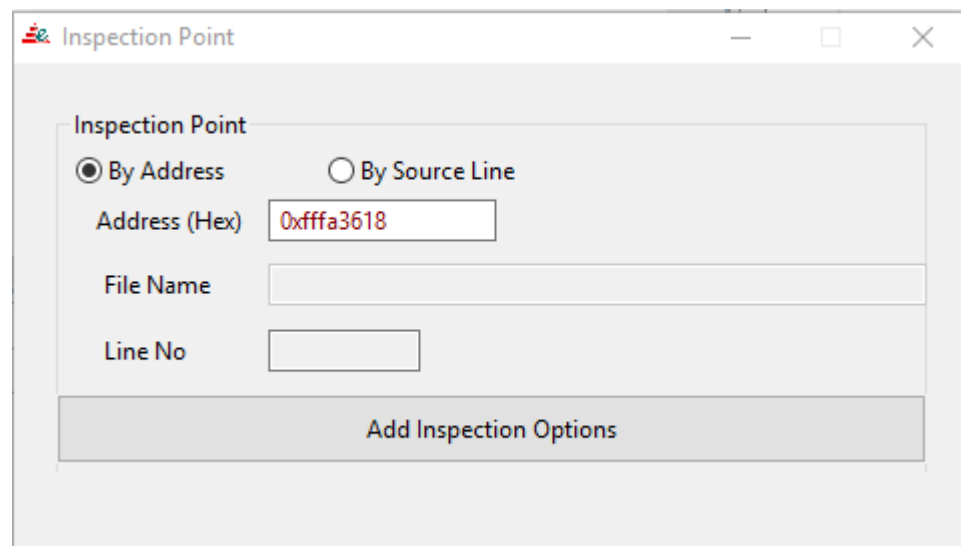


Figure: Inspection point by Address

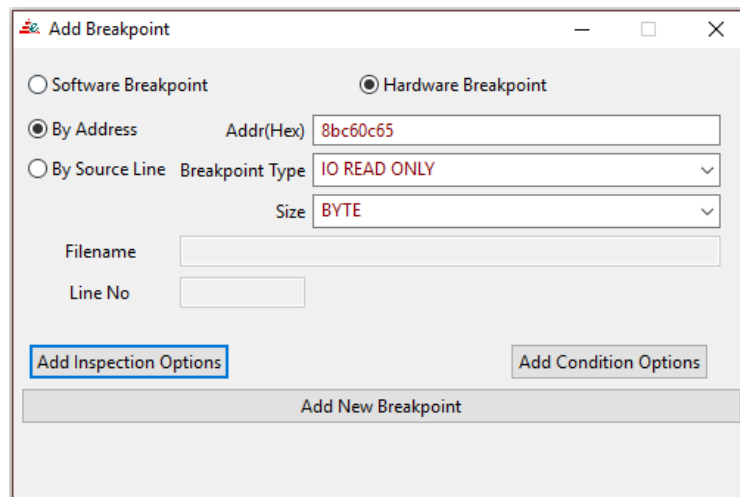
Note: At Inspection Point, target will not halt the execution state unlike breakpoint.

Adding Inspection Options to a Breakpoint

Inspection Options can be set on any set Breakpoint (Hardware or Software). When the breakpoint is 'HIT', the Inspection Options will be displayed on the AptioVDebugger Console and

Target's execution will be halted. Users can set Inspection options to Breakpoints while setting the Breakpoint itself by using the Add Breakpoint View or add Inspection Options to existing Breakpoints using the Breakpoints View. Inspection Options can be set to HW and SW breakpoints.

In order to set the Inspection Options to a Breakpoint use "Add Inspection Options" button from "Add Breakpoint" window.

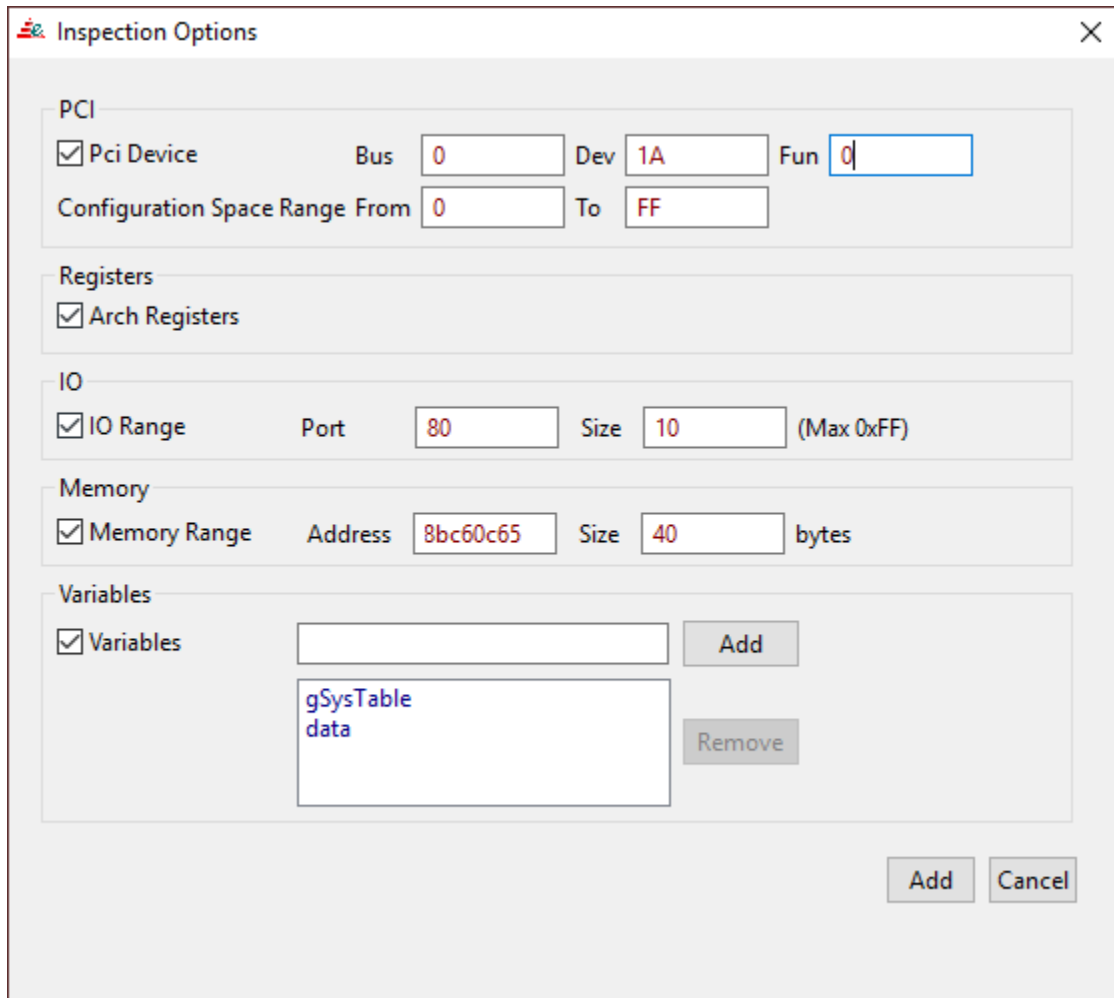


Clicking on "Add Inspection Options" button will open the Inspection Options Window. The "Inspection Options" window contains the Inspection Options available to be set such as PCI device option for dumping PCI configuration data at inspection point. Select Option, which you want to dump the information and provide corresponding data. Click on "Add" button.

User can see the dumped information into AptioV console window. Also this information automatically saved into inspection.txt file

Inspection Option Window

This window consist of different options to dump different data at inspection point. User can select one or more option. Following information user can dump at inspection point.



The 'Inspection Options' dialog box contains several sections for configuring the debugger's inspection capabilities. The 'PCI' section includes a checked 'Pci Device' checkbox, with input fields for 'Bus' (0), 'Dev' (1A), and 'Fun' (0), and a 'Configuration Space Range' from 0 to FF. The 'Registers' section has a checked 'Arch Registers' checkbox. The 'IO' section includes a checked 'IO Range' checkbox, with input fields for 'Port' (80) and 'Size' (10), and a note '(Max 0xFF)'. The 'Memory' section includes a checked 'Memory Range' checkbox, with input fields for 'Address' (8bc60c65) and 'Size' (40), and a label 'bytes'. The 'Variables' section includes a checked 'Variables' checkbox, a list box containing 'gSysTable' and 'data', and 'Add' and 'Remove' buttons. At the bottom right are 'Add' and 'Cancel' buttons.

Figure. Inspection Option window

PCI

By selecting this option, the Debugger will dump the PCI configuration of the specified BDF and Offset. The output will be displayed on the AptioVDebugger Console Screen.

```

Console X
TraceDebugger16b9820d851 [C/C++ Application] AptioVDebugger

PCI Data for Bus: [0x0] Device: [0x1A] Function: [0x0] Range: (0x0 - 0xff)
|-----|
| PCI          | 0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  |
|-----|
| 0x00000000   | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |
| 0x00000010   | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |
| 0x00000020   | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |
| 0x00000030   | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |
| 0x00000040   | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |
| 0x00000050   | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |
| 0x00000060   | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |
| 0x00000070   | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |
| 0x00000080   | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |
| 0x00000090   | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |
| 0x000000A0   | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |
| 0x000000B0   | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |
| 0x000000C0   | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |
| 0x000000D0   | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |
| 0x000000E0   | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |
| 0x000000F0   | FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF |
|-----|

IO Data for Port: [0x80] Length: [0x10] bytes
|-----|
| IO          | 0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  |
|-----|
| 0x00000080   | 00 00 FF FF FF FF FF FF FF FF FF FF FF FF FF |
|-----|

```

Sample PCI Output

Registers

By selecting this option, Debugger will dump the entire register set on the AptioVDebugger Console screen.

IO

User can dump the IO space starting from a specified Port to a specified offset.

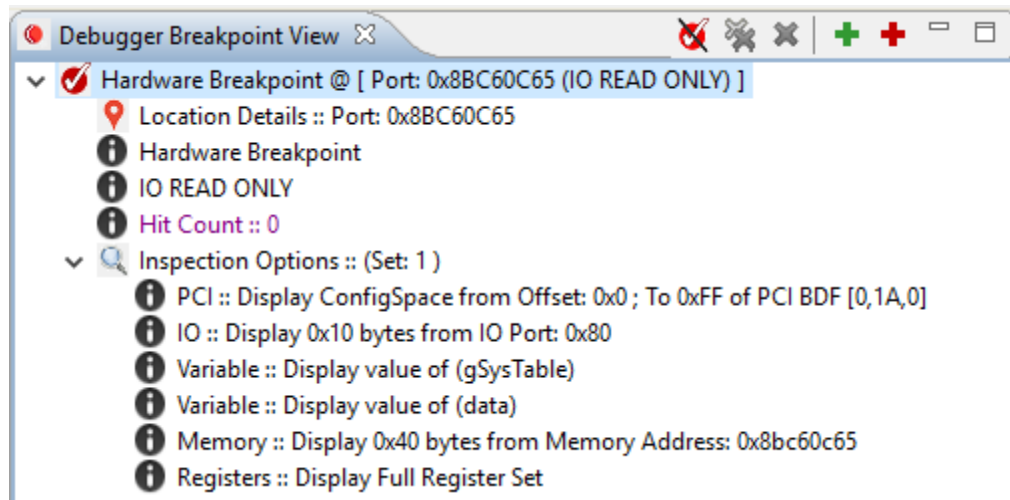
Memory

User can dump the Memory space starting from a specified Address to a specified offset.

Variables

User can enter the name of a Local or Global variable and its value will be dumped to the AptioVDebugger Console Screen.

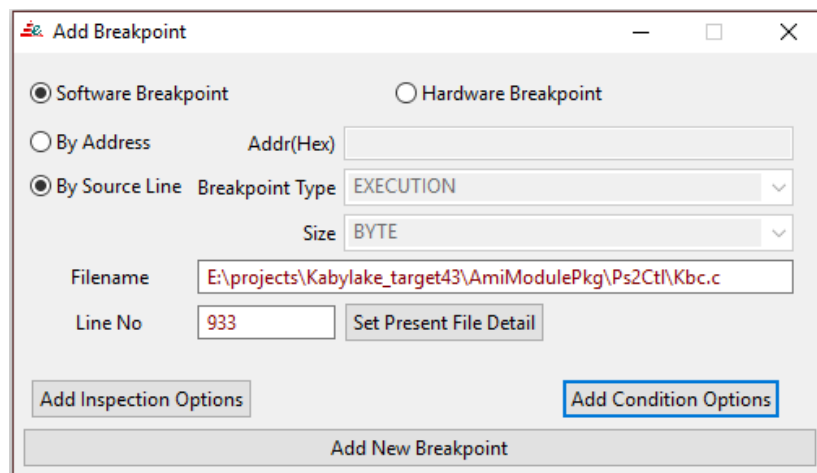
User can see inspection point with its detail information in breakpoint view window.



Adding Conditions to a Breakpoint

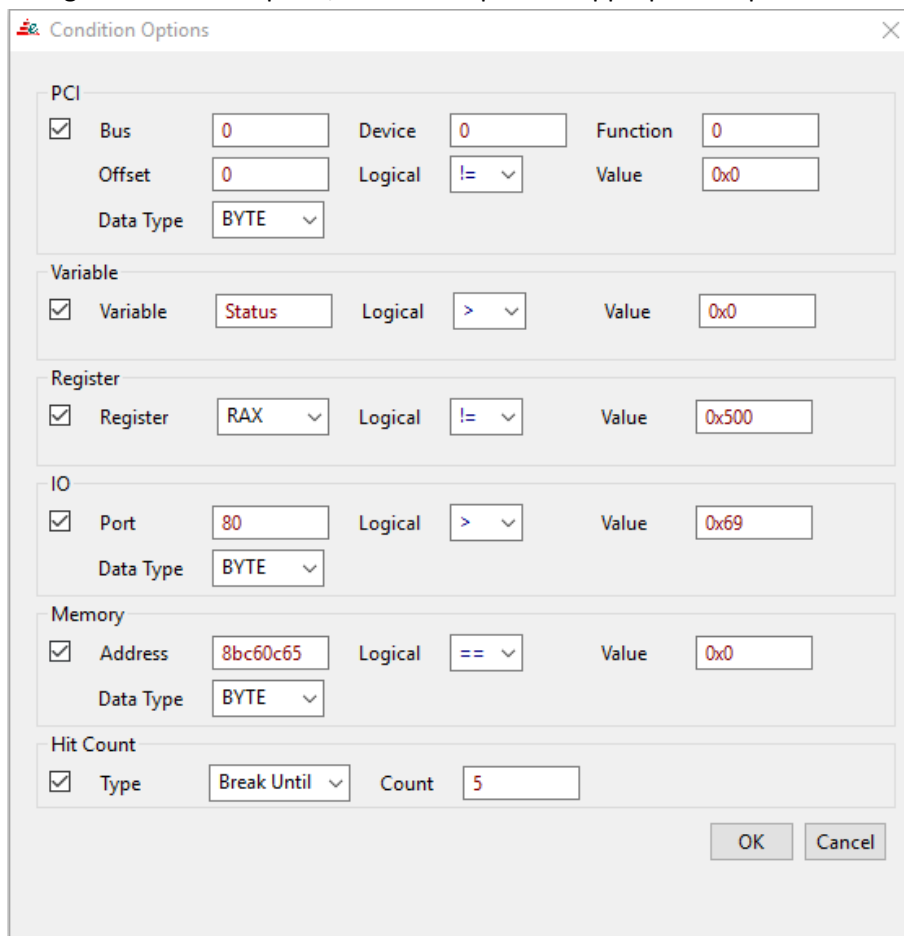
Conditions can be set on any Breakpoint. When the breakpoint is 'HIT', the target will either Halt execution or continue execution based on the conditions. Users can set Conditions to Breakpoints while setting the Breakpoint itself by using the Add Breakpoint View (or) can add Conditions to existing Breakpoints using the Breakpoints View. Conditions can be set to HW and SW breakpoints.

In order to set Conditions to a Breakpoint, use "Add Condition Options" button from "Add Breakpoint" window. Clicking on "Add Condition Options" button will open the Condition Options Window.



The 'Add Breakpoint' dialog box is shown. It has two main sections: 'Software Breakpoint' (selected) and 'Hardware Breakpoint'. Under 'Software Breakpoint', there are two options: 'By Address' and 'By Source Line' (selected). The 'By Source Line' section includes a 'Breakpoint Type' dropdown set to 'EXECUTION' and a 'Size' dropdown set to 'BYTE'. The 'Filename' field contains 'E:\projects\Kabylake_target43\AmiModulePkg\Ps2Ctf\Kbc.c' and the 'Line No' field contains '933'. There is a 'Set Present File Detail' button. At the bottom, there are three buttons: 'Add Inspection Options', 'Add Condition Options' (highlighted with a blue border), and 'Add New Breakpoint'.

User can select different condition (one or more) to halt at specified breakpoint if those conditions satisfy. After selecting the condition option, user has to provide appropriate input as below explained.



The 'Condition Options' dialog box is shown. It contains several sections for setting conditions:

- PCI**: A checkbox is checked. Fields include 'Bus' (0), 'Device' (0), 'Function' (0), 'Offset' (0), 'Logical' (!=), 'Value' (0x0), and 'Data Type' (BYTE).
- Variable**: A checkbox is checked. Fields include 'Variable' (Status), 'Logical' (>), 'Value' (0x0).
- Register**: A checkbox is checked. Fields include 'Register' (RAX), 'Logical' (!=), 'Value' (0x500).
- IO**: A checkbox is checked. Fields include 'Port' (80), 'Logical' (>), 'Value' (0x69), and 'Data Type' (BYTE).
- Memory**: A checkbox is checked. Fields include 'Address' (8bc60c65), 'Logical' (==), 'Value' (0x0), and 'Data Type' (BYTE).
- Hit Count**: A checkbox is checked. Fields include 'Type' (Break Until) and 'Count' (5).

At the bottom right, there are 'OK' and 'Cancel' buttons.

PCI

By selecting this option, user can set the PCI configuration condition for breakpoint.

In order to set PCI configuration condition user has to provide values for the mentioned fields, such as bus, device, function, offset, logical condition, value and Data Type.

Variable

By selecting this option, user can set the particular variable, choose the logical operation to be performed and the value with which it has to be compared. Register

By selecting this option, user can set the particular register, choose the logical operation to be performed and the value with which it has to be compared.

IO

By selecting this option, user can set the particular IO, choose the logical operation to be performed and the value with which it has to be compared. Also need to choose data type. Memory

By selecting this option, user can set the particular memory address, choose the logical operation to be performed and the value with which it has to be compared. Also need to choose data type.

Hit Count

User can select any one of the below option as hit count type.

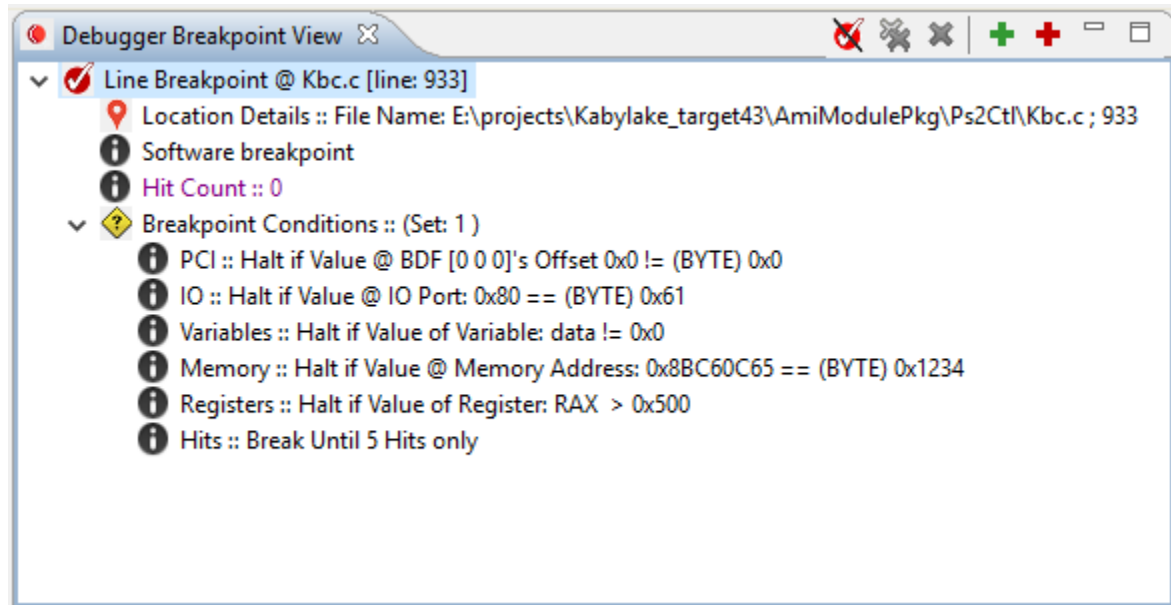
Break Until (n) – Breakpoint will halt on hit until the count value is $\leq n$, after which the breakpoint hit will not halt at this breakpoint.

Break After (n) – Breakpoint will not halt on breakpoint hit until count value is $> n$, after which it will halt at this breakpoint.

Click on Add button in the “Condition Option” window. After this user can see “Add Condition Option” button in the Add Breakpoint window as like below.

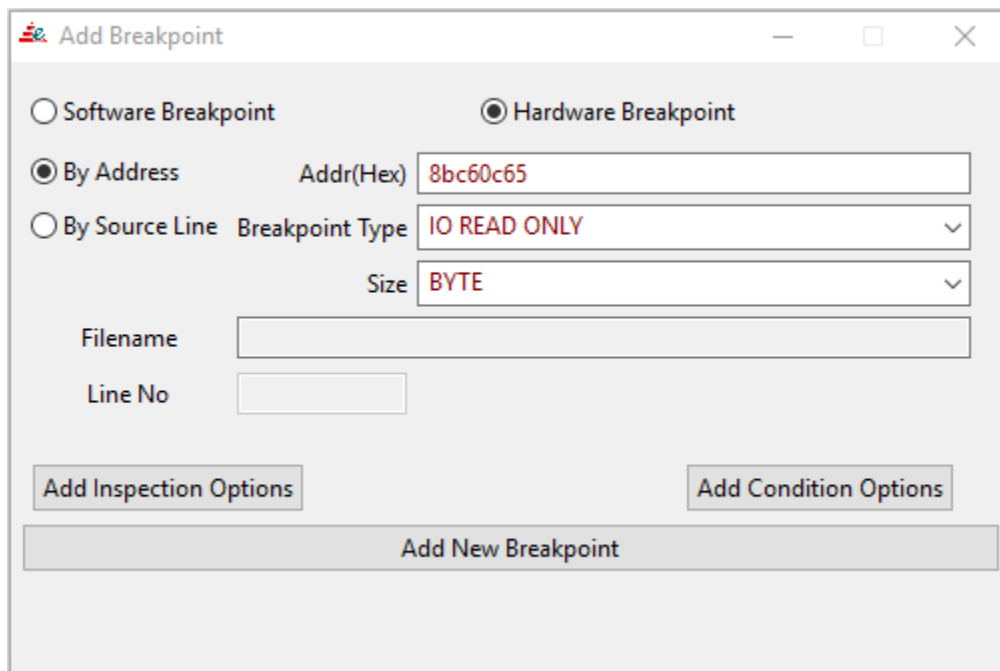
Click on “Add New Breakpoint” button in the “Add Breakpoint” window.

User can see breakpoint with its detailed information in breakpoint view window.



Example:

Below screenshot, we are setting a new breakpoint. Now if we want to make it as a conditional breakpoint, select 'Add Condition Options'.



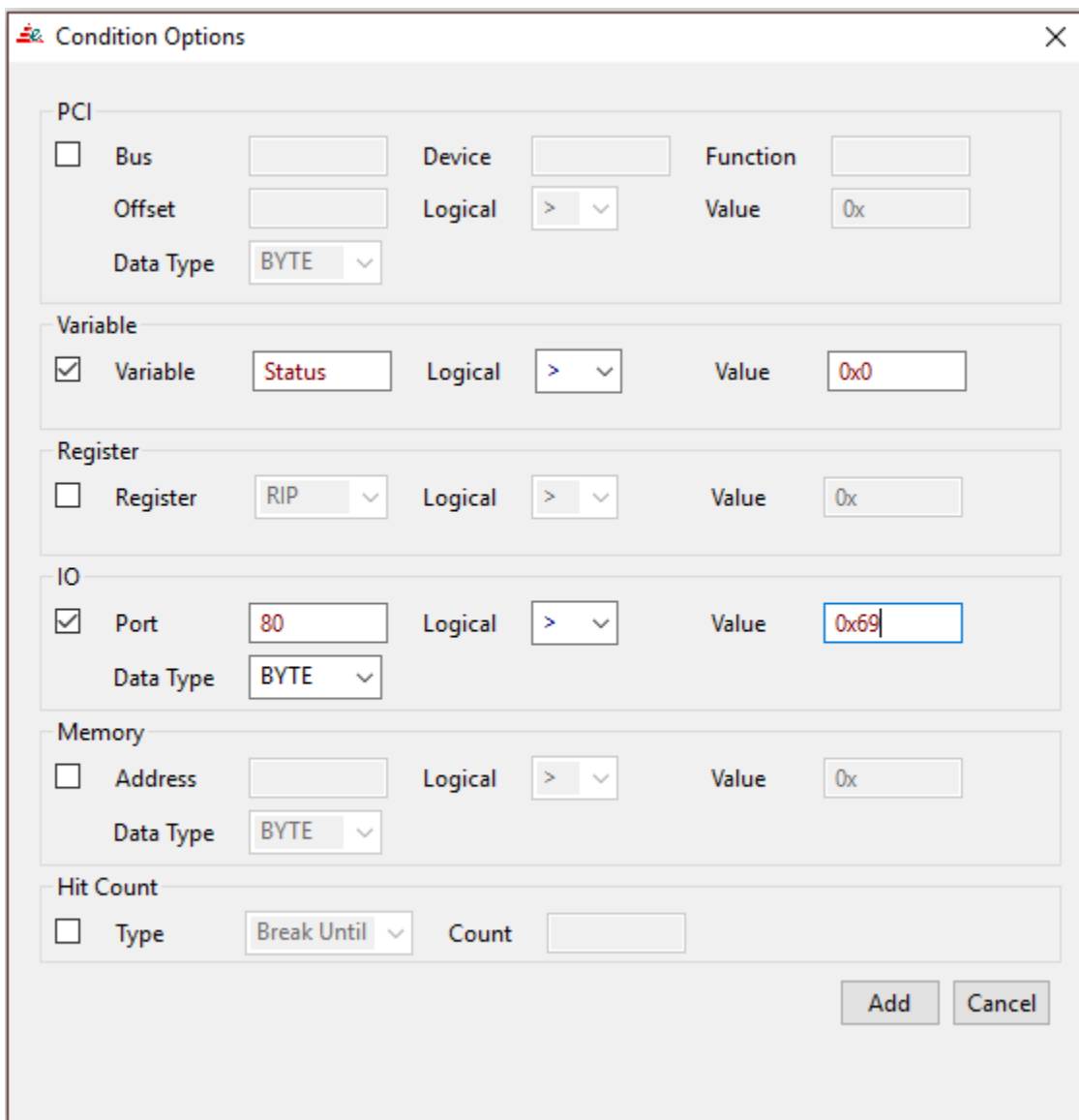
In below screenshot we were setting the condition by clicking on add condition. This will open the conditions window, now we set a sample condition,

Condition1 -> Local variable Status == 0

And

Condition 2 -> IO Port 80 value is equal to 0x15.

This means above breakpoint will happen only if both the conditions will be satisfied, otherwise it will not halt. Please Refer the below picture.

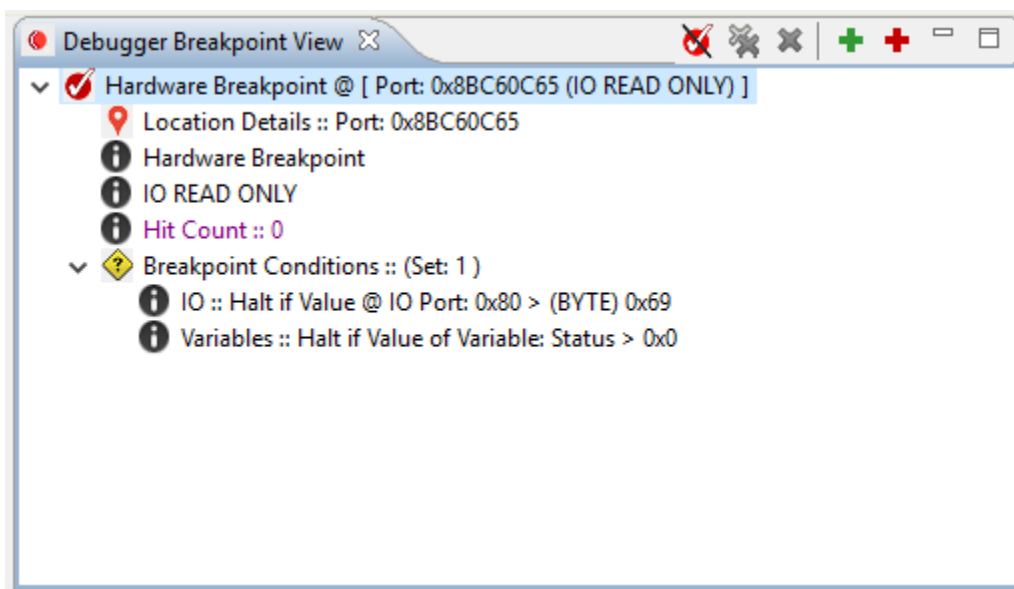


The screenshot shows the 'Condition Options' dialog box in the Trace Hub Debugger. The dialog is divided into several sections for setting different types of conditions:

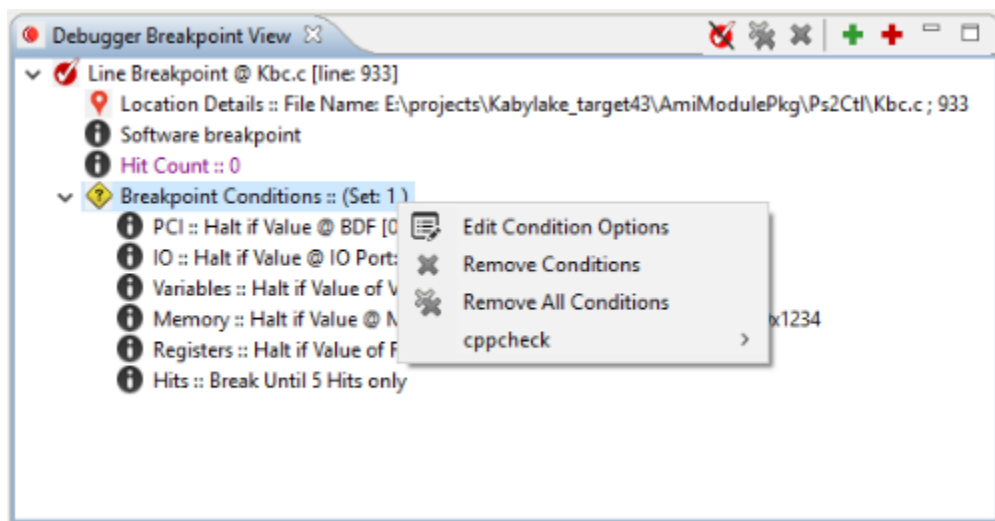
- PCI:** Includes fields for Bus, Device, Function, Offset, Logical (with a dropdown arrow), and Value (set to 0x).
- Variable:** The 'Variable' checkbox is checked. The variable is set to 'Status', Logical is set to '>' (with a dropdown arrow), and the Value is set to '0x0'.
- Register:** Includes fields for Register (set to 'RIP' with a dropdown arrow), Logical (with a dropdown arrow), and Value (set to 0x).
- IO:** The 'Port' checkbox is checked. The port is set to '80', Logical is set to '>' (with a dropdown arrow), and the Value is set to '0x69'.
- Memory:** Includes fields for Address, Logical (with a dropdown arrow), and Value (set to 0x).
- Hit Count:** Includes a 'Type' dropdown (set to 'Break Until') and a 'Count' field.

At the bottom right of the dialog are 'Add' and 'Cancel' buttons.

In addition, we can see and add condition or inspection point on existing breakpoint using breakpoint view. Simply by right click on breakpoint then click on Add Condition/Inspection option to add Condition/Inspection point. Below screenshot is showing how to add condition or inspection point to existing breakpoint.

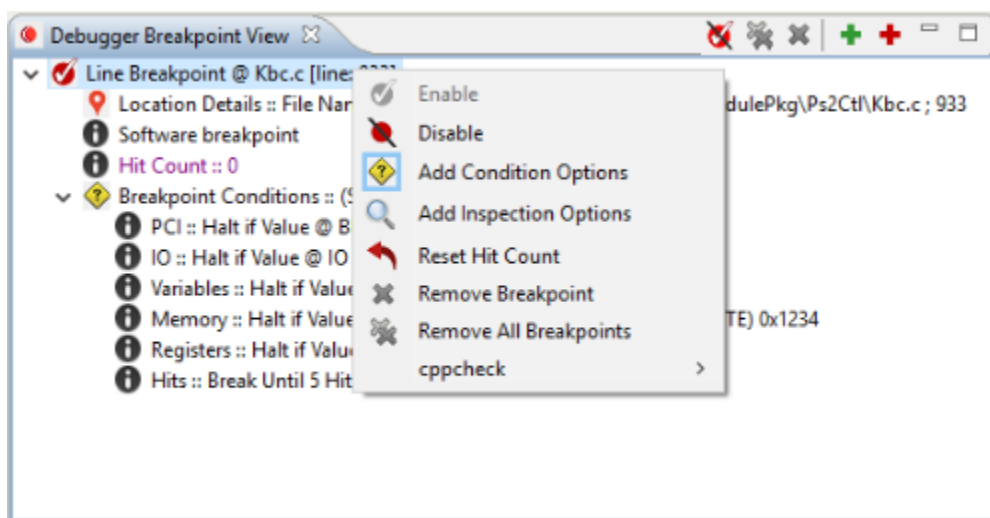


User can modify or edit the set Conditions from the Breakpoint View by Right clicking on the Conditions Set, like below



And selecting 'Edit Condition Options', this will again pop up the Condition Options window with the set conditions. User can edit the previously set conditions or add/remove conditions.

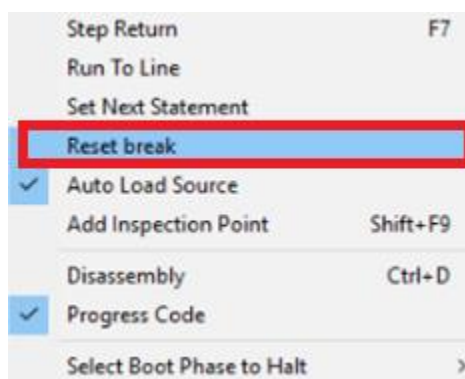
Users can also add a new set of conditions (allows multiple conditions of the same type) by right Clicking on the breakpoint itself and selecting 'Add Condition Options', like below



Halting at Reset Vector

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

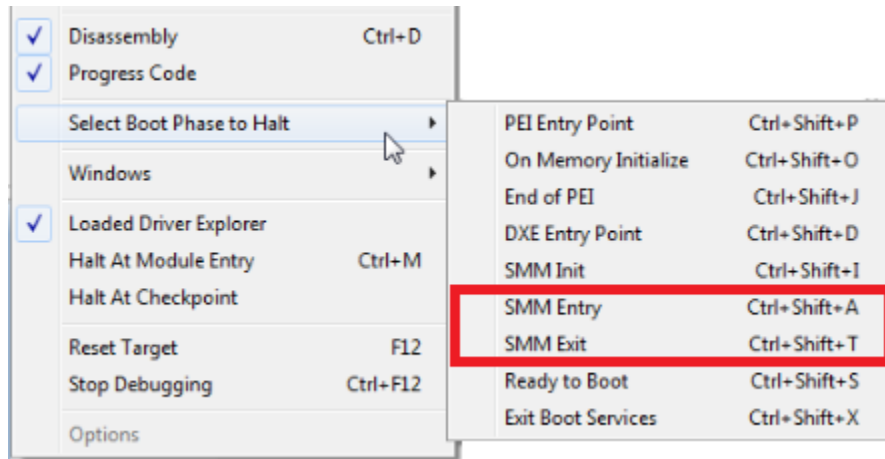
- VeB Menu: **Trace Hub**→**Reset Break**
- Toolbar: using icon []



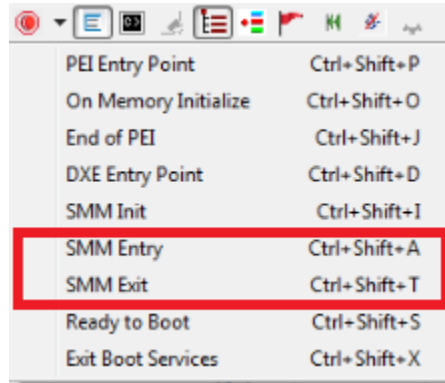
SMM Debugging

Halting on SMM Entry\Exit

Select: **Trace Hub** → **Select To Boot Phase** → **SMM Entry/Exit** from Menu item



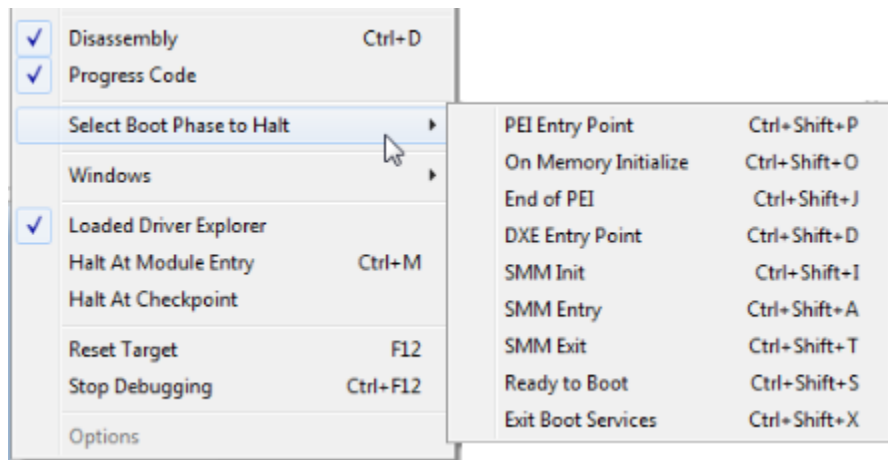
or Select SMM Entry/Exit from Toolbar



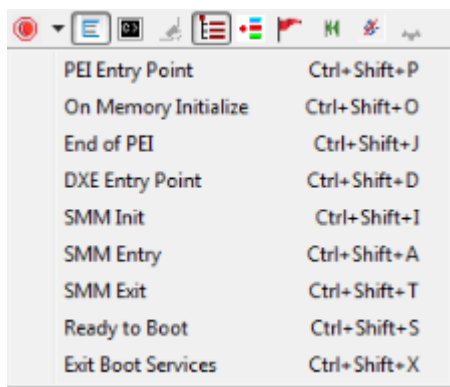
NOTE: Requires PCD "gUefiCpuPkgTokenSpaceGuid.PcdCpuSmmEnableBspElection" to be set to FALSE

Working with Select Boot Phase to Halt

The user can configure the Debugger to break on each phase. The user can set the Debugger to halt on each phase using the following options:



Or using Trace debug tool bar



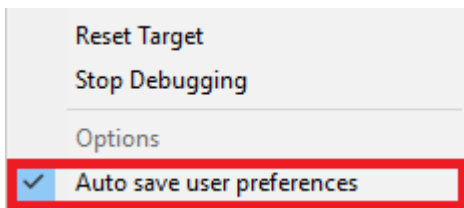
Note: User needs to halt or enable reset break to get the special breakpoints hit

Working with Auto save user preferences

Auto save user preferences is used to restore the previous debug session dialogs, views and special breakpoints and the support is enabled by default.

Users can access the Auto save user preferences by -

- Select: **Trace Hub -> Auto save user preferences**



- or select the Trace Debug Toolbar Icon []

Note:

Console views, NVRAM, MSR, Source Editor and option Dialog do not have this support.

In Halt at Module Entry and Halt at checkpoint dialogs, values also will get restore

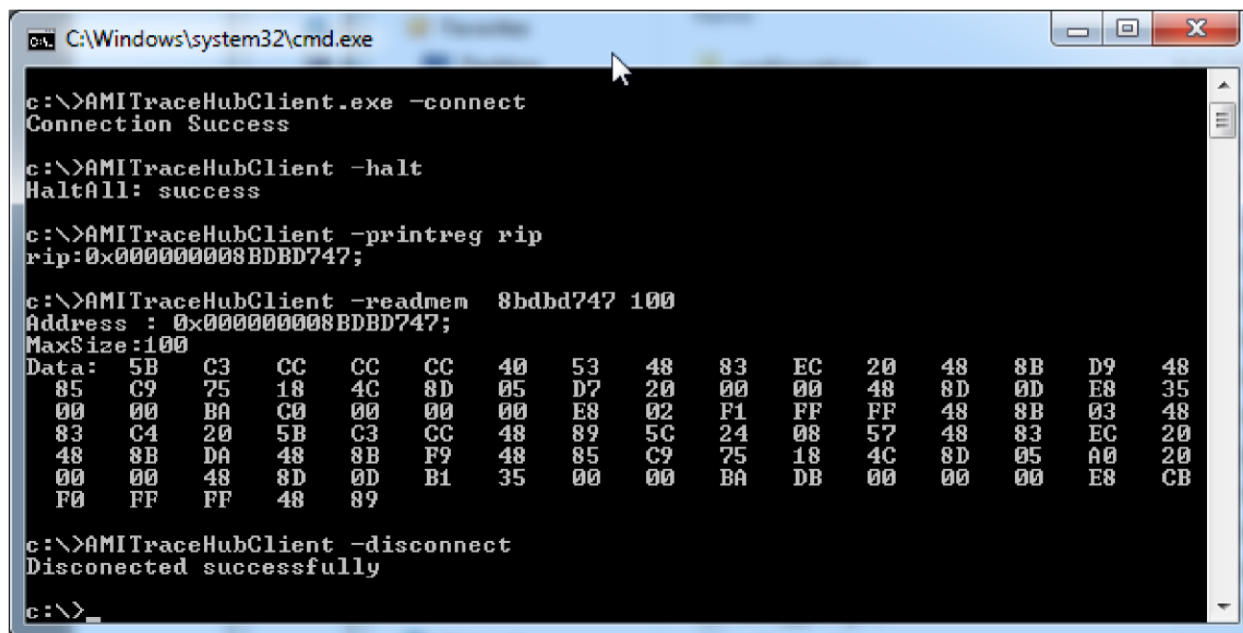
Perform Basic DCI Functional Test

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

A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window has a black background with white text. The text shows the execution of the AMITraceHubClient.exe program with various command-line arguments. The output includes connection status, halt status, register values, and a memory dump. The memory dump is presented as a grid of hexadecimal values.

```
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 00 E8 02 F1 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
Disconnected successfully

c:\>
```

Figure 3 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. For the selective debugging, if not able to see the source properly, please also add the "UefiDriverEntryPoint" module to the selective debugging.
3. Target module is required to get the loaded driver notification to host over trace messages.
4. To debug a specific module in Aptio V, add "__debugbreak()" code to break target before continuing with debug operation
5. Edit and Continue of Files during the debug session is not supported.
6. On Purley Refresh – Reset Break will halt at address other than reset vector
7. Step Out is not expected to work if the RSP is modified by code, e.g. SMM Exit.
8. Resetting CPU will clear all debug settings, except for reset break flag. This means for example that user-specified breakpoints will be invalid until the target is halted once after reset. Note: The halt can be due to either a reset-break, or due to an user-initiated halt. In either case the debugger will restore the necessary debug settings.
9. Step operations once the target has stopped at an HLT instruction is not possible. Set a breakpoint after the HLT and run to it instead.
10. If the platform enters the CAT Error state (identified by glowing red light),
 - a. Power cycle the platform after removing all connection between target and host
 - b. Disconnect and Reconnect the Host Debugging Session
11. 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.
12. Sometimes the following warning is reported to user – "WARNING: Could not open/create prefs root node Software\JavaSoft\Prefs at root 0x80000002. Windows RegCreateKeyEx(...) returned error code 5."
Fix: This is a known Java issue. Save the following content to .reg file and execute the file :
Windows Registry Editor Version 5.00
[HKEY_LOCAL_MACHINE\Software\JavaSoft\Prefs]
13. AMI Debug solution does not support debugging OS
14. Adding more breakpoint will slow down the run control process such as Step into, Step over, Step out, and run to line
15. On target reset to enable User breakpoints or SMM breaks, the user needs to halt and resume the target once after reset
16. In order to enable DCI in ME in server platforms, need to enable DCI in FitC to default xml



-
17. Commands like crb or crb64 are not supported and require additional python code which is not available by default
 18. While fast stepping (continuous push stepping), disassemble view may show error message, user need to push refresh button, to show all the disassemble code.
 19. Please disable the BIOSGuard, if encounter for some module breakpoint does not work issue.

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.



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.
4. **Issue:** Host not able to detect the DBC device, while using the PCI to USB3.0 card.
Solution: use the suggest PCI to USB3.0 card, [SHINESTAR USB 3.0 PCIe Expansion Card](#).

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 Exl 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.