tianocore

UEFI & EDK II TRAINING

EDK II Debugging w/ Intel® System Studio

tianocore.org

tianocore

LESSON OBJECTIVE

- Intel® System Studio Debugger Overview
  - Identify the Intel® System Studio Debugger host and target
- Getting started with Intel® System Studio Debugger
  - Using IPCCLI
  - TCF GUI
  - XDB GUI
- Configure when target has UEFI debug agent
- Using the INT1 ICEBP instruction as a debug break point

SYSTEM STUDIO DEBUGGER

Intel® System Studio Debugger Overview

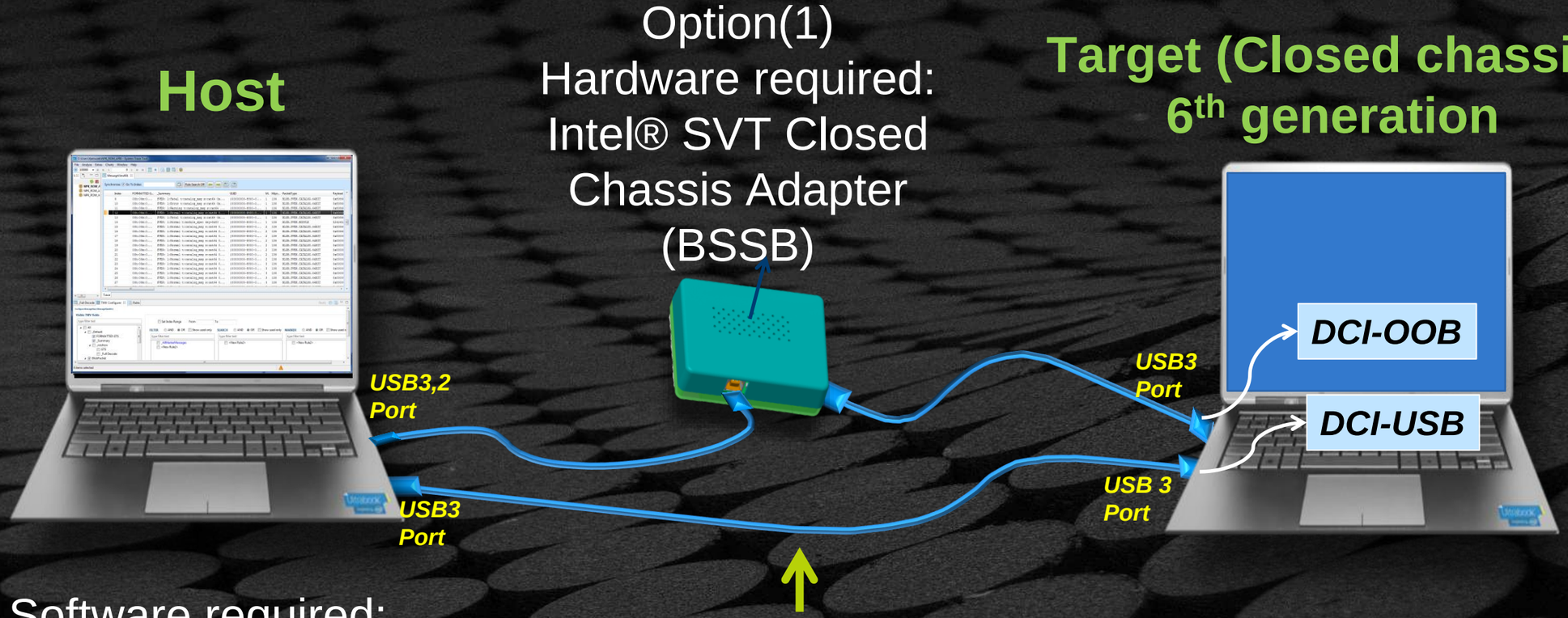
Debugging Closed Chassis Product

No debug connector

Target: closed chassis

Debug equipment

Debug Interface - DCI



Software required:
Intel® System Debugger

Option(2)
USB3 Debug cable type A-to-A cable

DCI tool-transport technology that allows connecting to a closed-chassis target system using USB 3.0.

System-wide Closed Chassis Debugging

JTAG*-based system debug and system trace over low-cost USB* connections

- Eliminate hours of developer time working on actual production hardware instead of expensive engineering samples
- Save hundreds of dollars on JTAG equipment & probes - debug over a standard USB* connection instead of expensive JTAG probes
- Developers no longer need to wait for a turn at a shared JTAG debugging station
- Design flexibility alleviates the requirement for an accessible hardware JTAG port

USB CABLE

Debug & Trace OS boot

INTEL® SVT CLOSED CHASSIS ADAPTER[1]

Debug & Trace from CPU reset

(1) SVT = Silicon View Technology – more details: https://designintools.intel.com/product_p/itpxdpsvt.htm

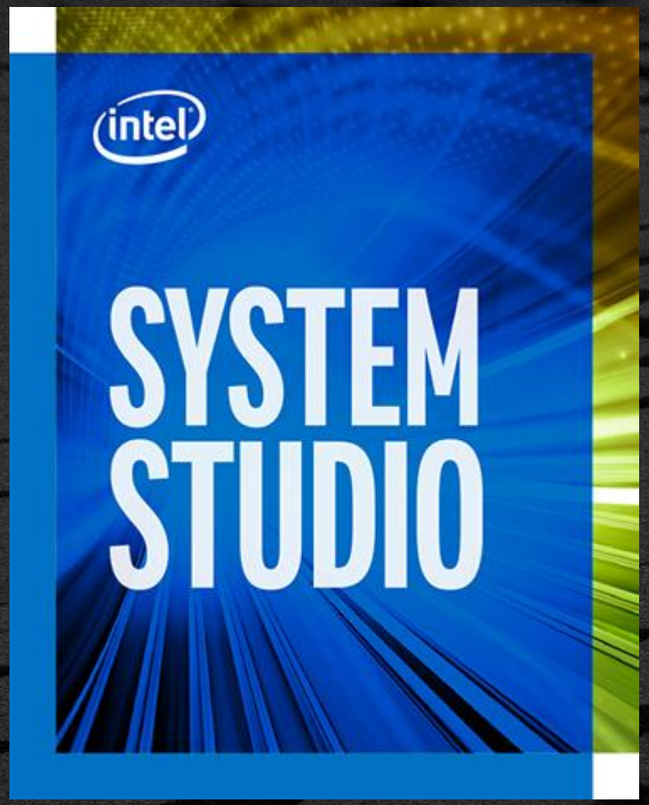
Intel® SVT CCA And USB3.0 Debug Cable

Intel® SVT Closed Chassis Adapter (CCA) can be purchased through Intel® Design-In Tools Store at https://designintools.intel.com/product_p/itpxdpsvt.htm

Intel® SVT DCI DbC2/3 A-to-A Debug Cable : https://designintools.intel.com/SVT_DCI_DbC2_3_A_to_A_Debug_Cable_1_Meter_p/itpdciamam1m.htm

Download Intel® System Studio

GET STARTED NOW



Free 90-Day Renewable Community License

Use Intel® System Studio with a free community license backed by community forum support. This license allows you to use the software for 90 days. You can refresh the license an unlimited number of times, allowing you to use the latest version. Convert to a paid license at any time, which provides Priority Support for one year.

Supported host operating systems:

Windows*

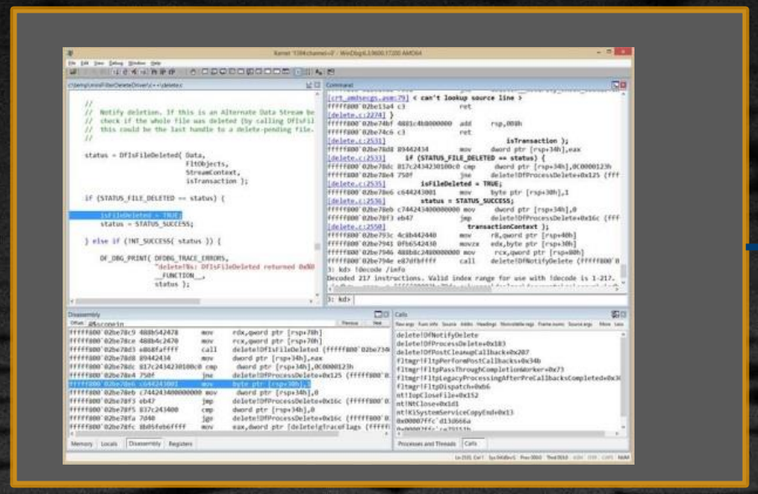
Linux*

macOS*

Register & Download

https://software.intel.com/en-us/system-studio/choose-download

Intel® System Debugger Overview



[1] System Trace is only Supporting 6th generation core

Key Features

- Linux* and Windows* host
- JTAG debug for Intel® Atom, Core™, Xeon® & Quark SoC-based platforms
- EFI/UEFI Firmware, bootloader debug, Linux* OS awareness and Kernel module debug
- Full CPU register description and bitfield editor
- Access to page translation and descriptor tables
- LBR, IPT On-Chip instruction trace support, SMP run control support
- JTAG debug & instruction trace to Microsoft* WinDbg* kernel debugger *NEW*
- System Trace: System-wide hardware and software event trace *NEW*

tianocore

Intel® System Debugger

Debug & Trace Intel® Architecture-based System Software & IoT applications

C/C++ High-Level Language Debug

Script Language

Flash Support

Peripheral Register Views

Execution Trace Support (Intel® Processor Trace)

UEFI* Awareness

OS Awareness
- Linux*
- Android*
- Yocto Project*
- Wind River Linux*

Extensions for MicrosoftWinDbg*

JTAG* (Intel® ITP-XDP3)

USB* Direct Connect Interface (DCI) (USB Closed Chassis Adapter, USB DbC)

UEFI* Agent EDKII for UEFI

Intel® Architecture

- Speed up debug and validation with deep hardware and software insight for faster time-to-market
- Ideal for platform bring-up and debug of firmware, UEFI/BIOS, operating systems, and device drivers
- Debug Windows* and Linux* kernel sources, and dynamically loaded drivers & kernel modules

tianocore

UEFI BIOS Support

General UEFI Features:

- Source-level debug in any phase of UEFI, from reset to OS boot
- Load symbols for all or selected modules
- Simultaneous debug of MSVC and GCC-built modules

Passive mode (interrogation-based):

- Inspect target memory to locate modules, load symbols
- Can be used on production BIOS
- Requires JTAG or DCI cable

Active Mode (agent-based):

- Receive notifications from agent as modules are loaded/unloaded
- Break at init of a named module, regardless of load position
- Requires debug agent “SourceLevelDebugPkg” in EDKII
- Only use on Debug version of FW

GETTING STARTED

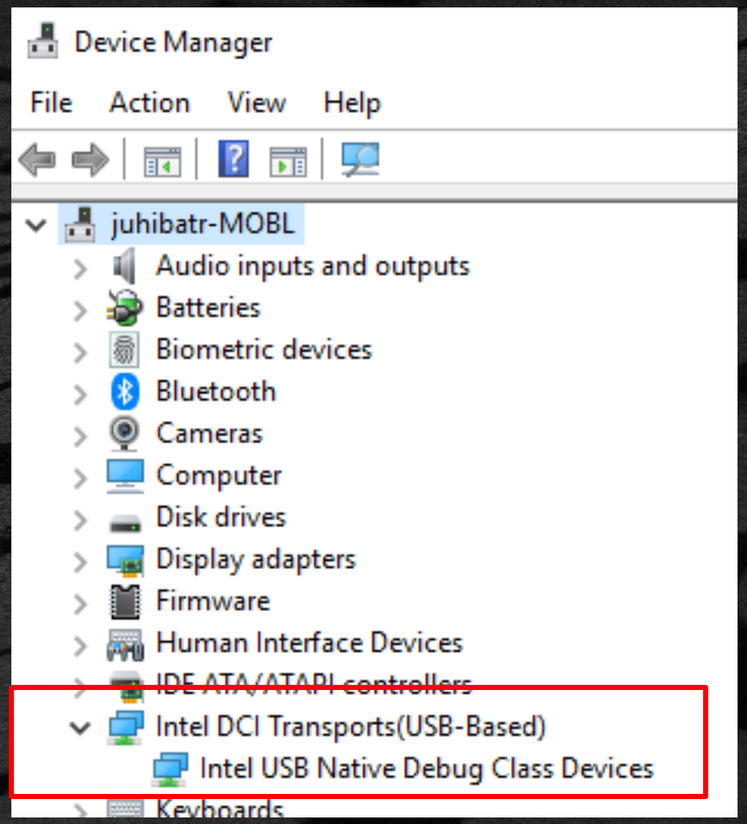
Getting started with Intel® System Studio Debugger

Power up the Target

Power Up the target, make sure the Intel® Debug Cable - DCI probe is connected the target and host

Check for the connection by opening the Device Manager and looking for Intel DCI Transports (USB based) – Intel USB Native Debug Class Devices as shown in the picture.

Windows Device Manager



Intel® System Studio Debugger- IPCCLI

Ipccli - Tool that provides a Python command line interface to use OpenIPC

First edit the config file OpenIpcConfig.xml
Go to folder `C:\IntelSWTools\system_studio_2019\tools\OpenIPC_1.1915.3701.200\Config` and open file `OpenIpcConfig.xml`
Edit it to reflect `<DefaultIpcConfig Name="BXTP_DCI_OpenRC" />` for UP Squared board.

Open command line prompt as administrator and setup the environment using the iss_env.bat command

```
$> cd C:\IntelSWTools\system_studio_2019
$> iss_env.bat
```

Execute ipccli (make sure python and python/scripts are in the path)

```
$> ipccli
```

Intel® System Studio Debugger- IPCCLI

- Halt the target using `itp.halt()`
- Set reset break using following command `itp.cv.resetbreak`
- Set `itp.cv.resetbreak=1` if it is set to 0
- Then use this command to for reset `itp.threads[0].port(0xcf9, 0x6)`
- Check the status of the target threads and halt them if not already halted using `itp.status()` and `itp.halt()` resp.
- Enable `INT1(ICEBP)` handler over ISD: `itp.breaks.int1exception = 1`
- Use `itp.go()` to resume processors to running
- Use `itp.help()` for help manual

Setting IO breakpoint from IPCCLI

Open Ipccli Command window and use instruction `itp.brget()` for any breakpoints

Use command `itp.brnew("0x80", "io")` to create a IO breakpoint on Port 80 Hex

Go back to the ISD GUI and resume the target

The target will halt on hitting the IO breakpoint

Once the Breakpoint is hit , click on Loadthis in the GUI to load the source files which will be stopped at Postcode function call.

Use the command `itp.brdisable( #)` where “#” is the breakpoint to disable

tianocore

Intel® System Studio Debugger - 2 GUI Versions

TCF based debugger

- Target Communication Framework(TCF) works with the Eclipse IDE
- TCF provides a complete modern debugger for C/C++

XDB Based debugger

- Intel ITP based debugger supporting JTAG
- Moving to Legacy debugger by Intel® System Studio

The underlying debugging in similar but the user interface is different

tianocore

GETTING STARTED

1. Download latest Intel® System Studio :

   http://intel.ly/system-studio

2. Run installer until installer completes installation

TCF - Launch script for Intel® System Studio


: C:\Program Files (x86)\IntelSWTools\system_studio_2019\iss_ide_eclipse-launcher.bat


: opt/intel/system_studio_2019/IntelSWTools/system_studio_2019/iss_ide_eclipse-launcher.sh

XDB - Launch script for Intel® System Studio Debugger – Legacy


: C:\Program Files (x86)\IntelSWTools\system_studio_2019\System Debugger 2019\system_debug\xdb.bat


: opt/intel/system_studio_2019/system_debugger_2019/system_debug/xdb.sh

tianocore

Intel® System Studio Debugger - 2 GUI versions

TCF based debugger

- Target Communication Framework(TCF) works with the Eclipse IDE
- TCF provides a complete modern debugger for C/C++

XDB Based debugger

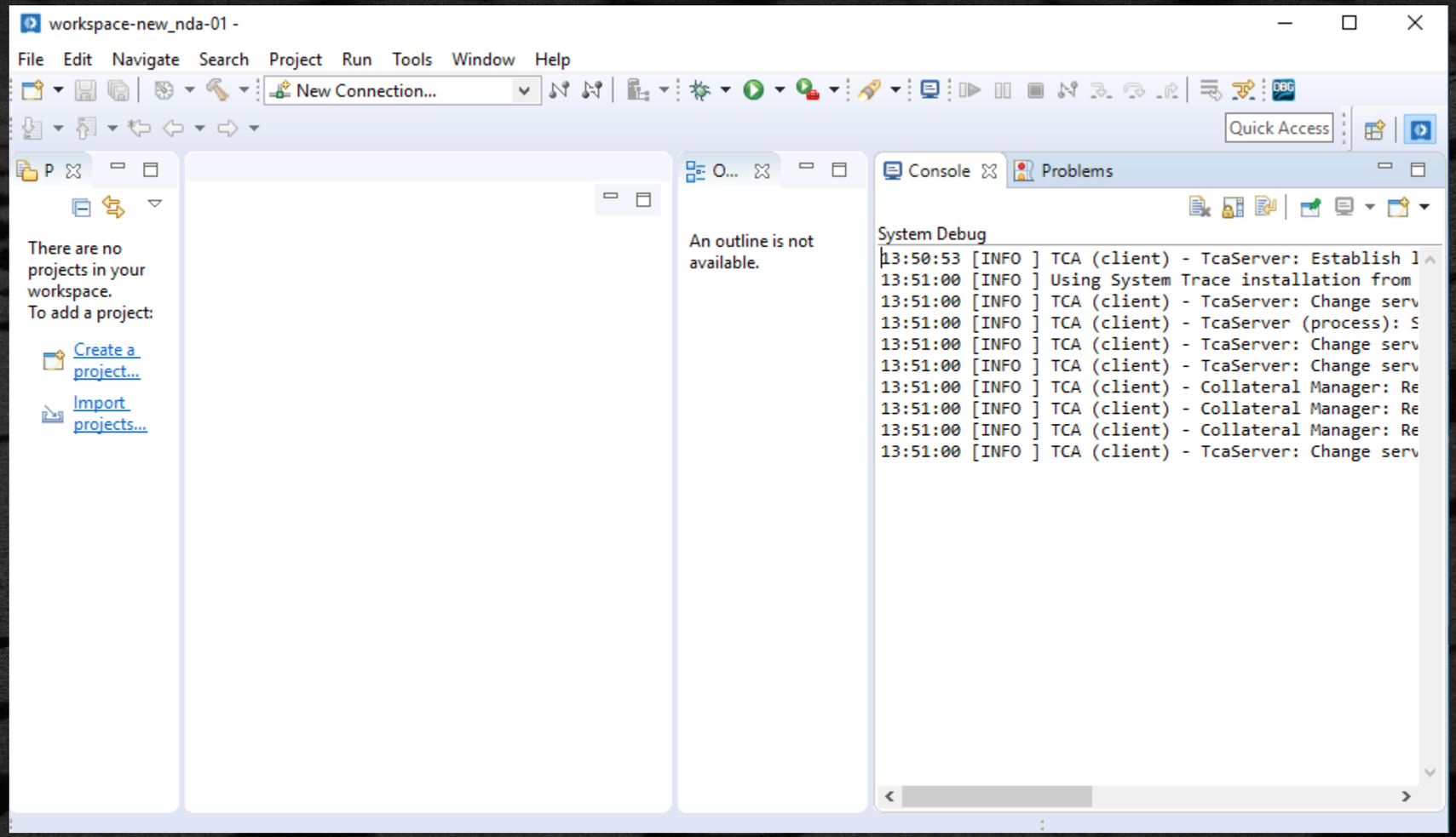
- Intel ITP based debugger supporting JTAG
- Moving to Legacy debugger by Intel® System Studio

The underlying debugging in similar but the user interface is different

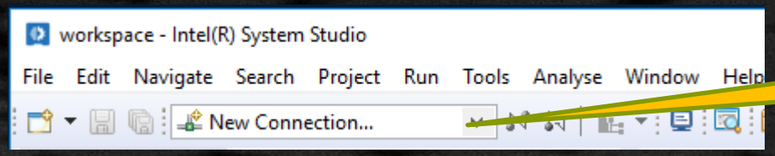
tianocore

Intel® System Studio

Launch Eclipse IDE



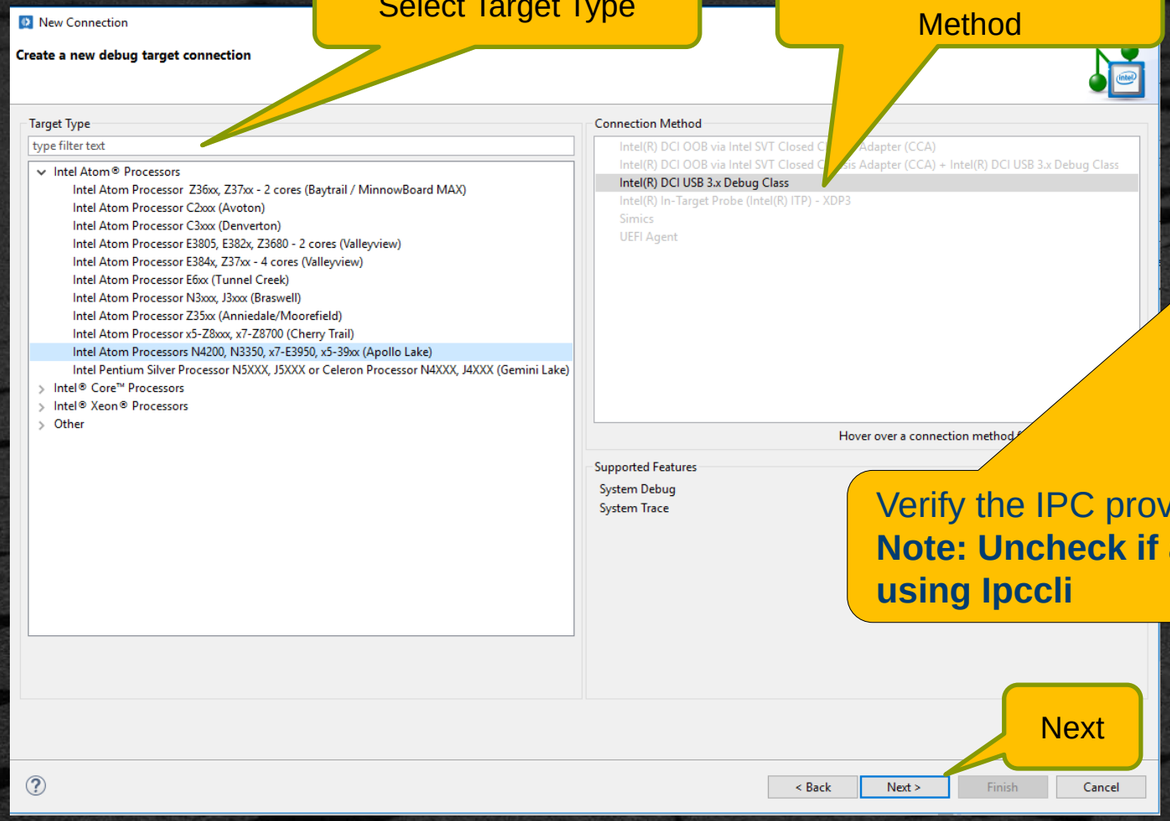
Create a New Debug Target Connection

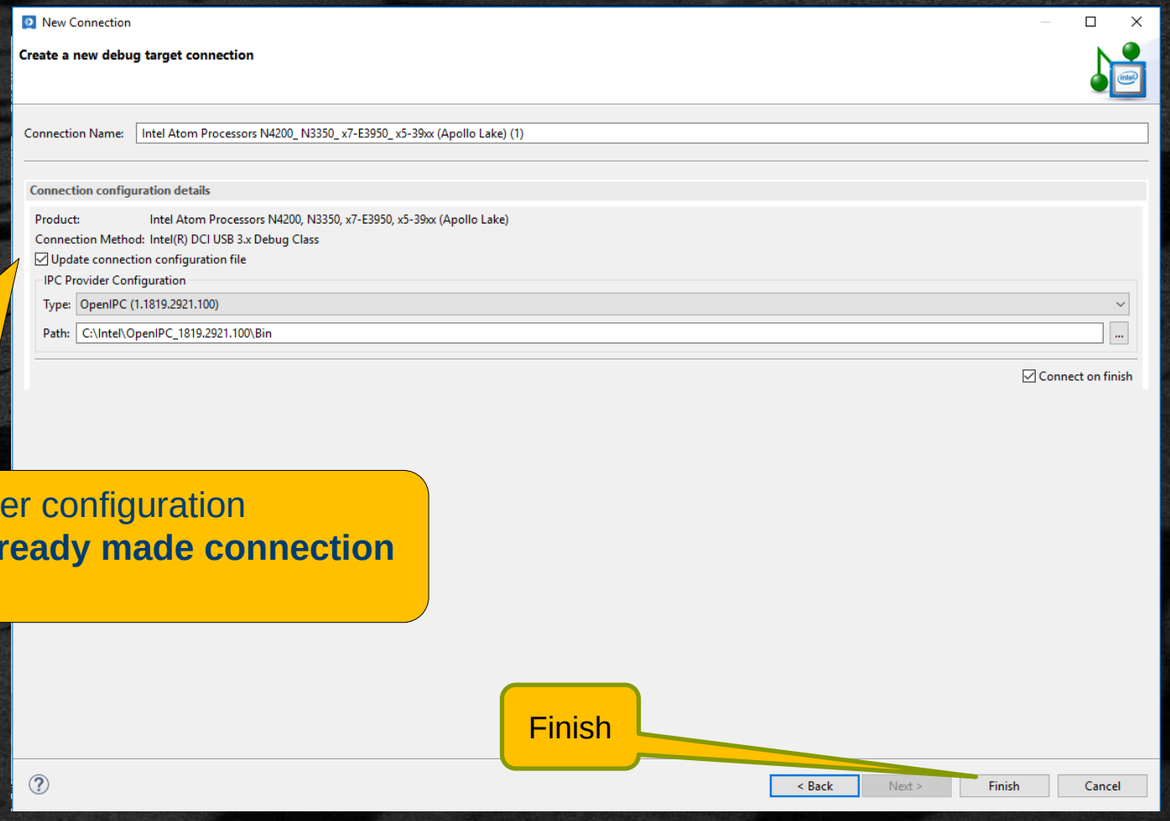


Select New Connection

Select Target Type

Select Connection Method





Verify the IPC provider configuration
Note: Uncheck if already made connection using Ipccli

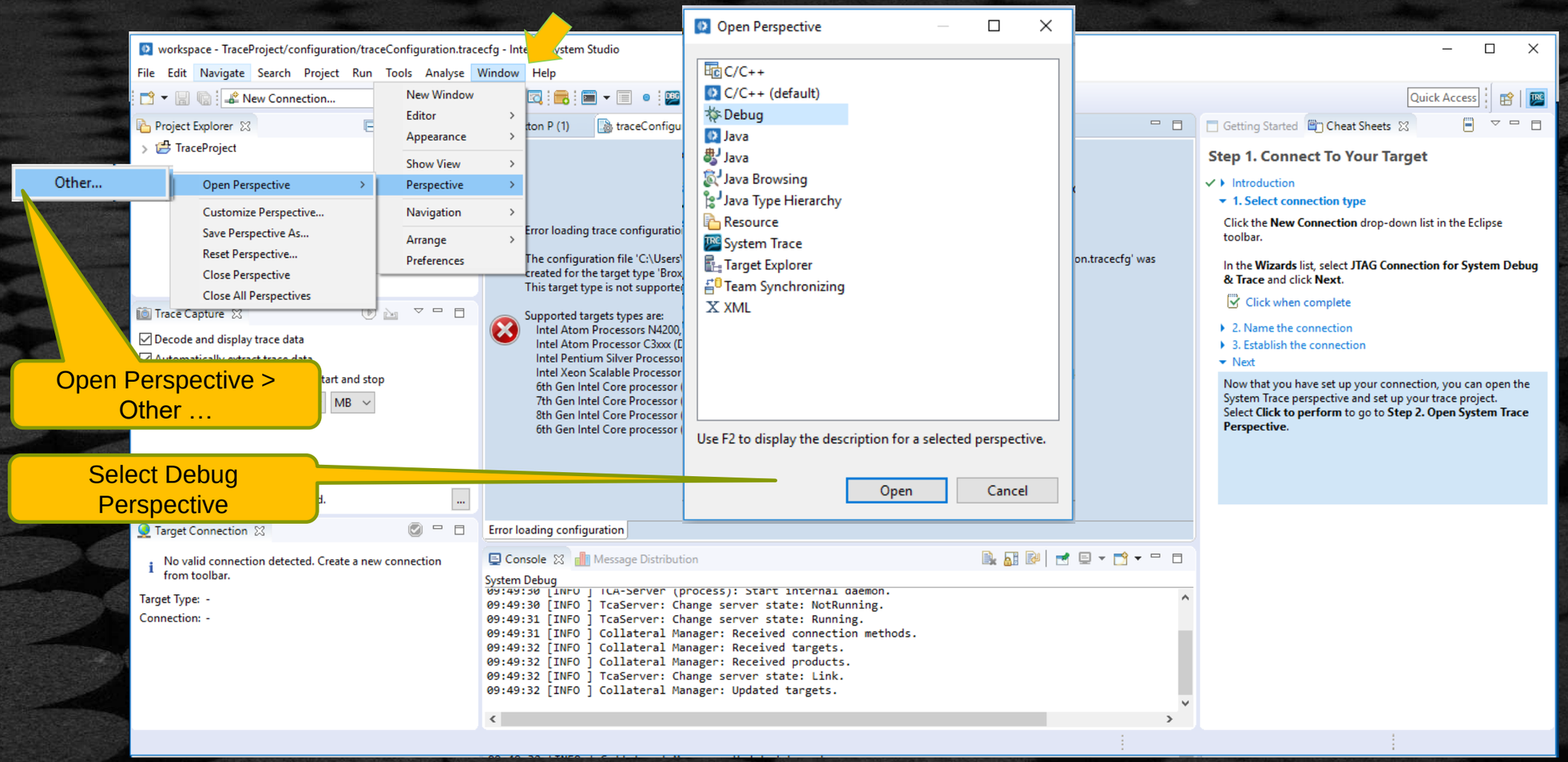
Next

Finish

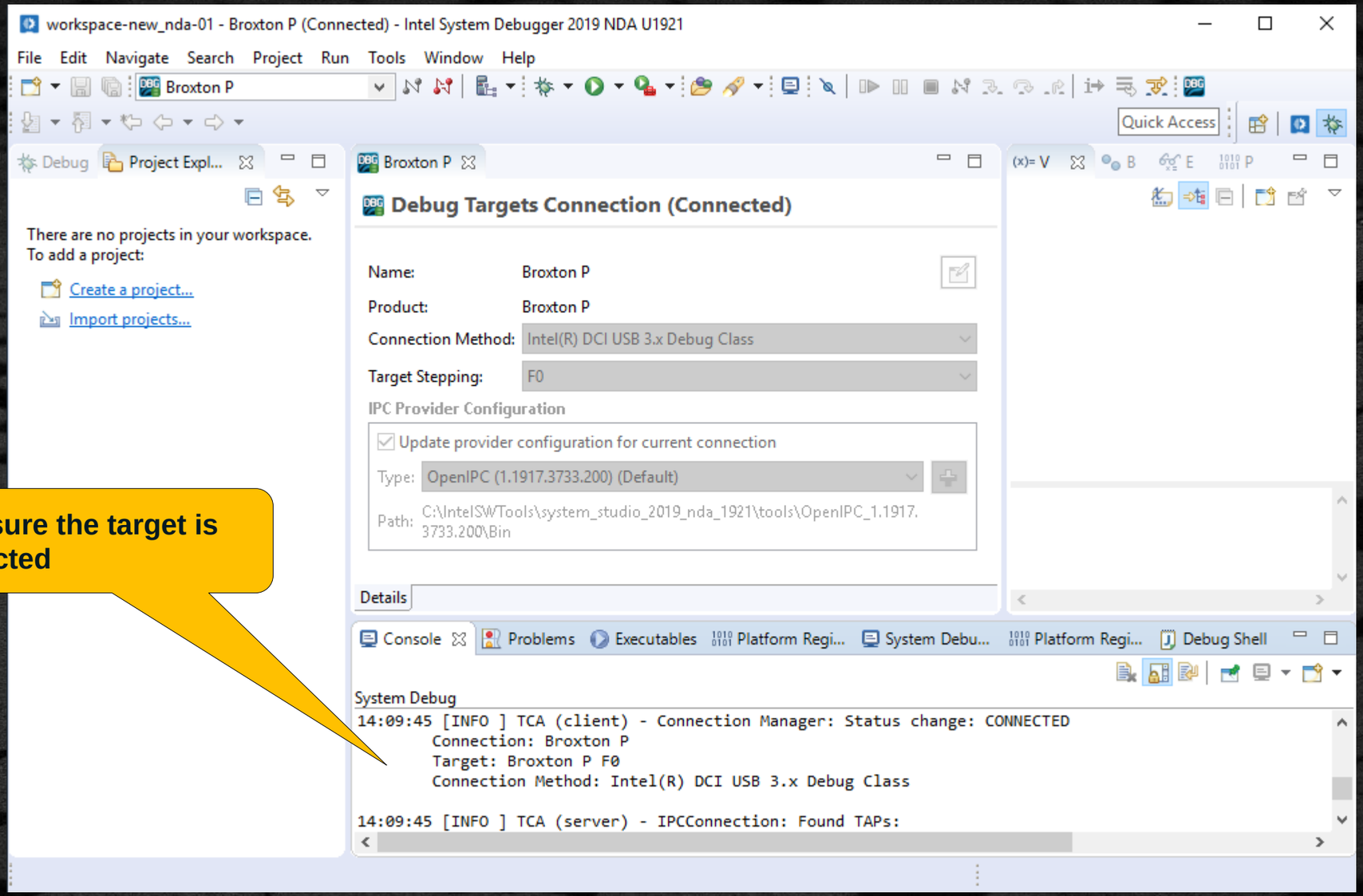
tianocore

Intel® System Studio - TCF

Open Perspective Debug

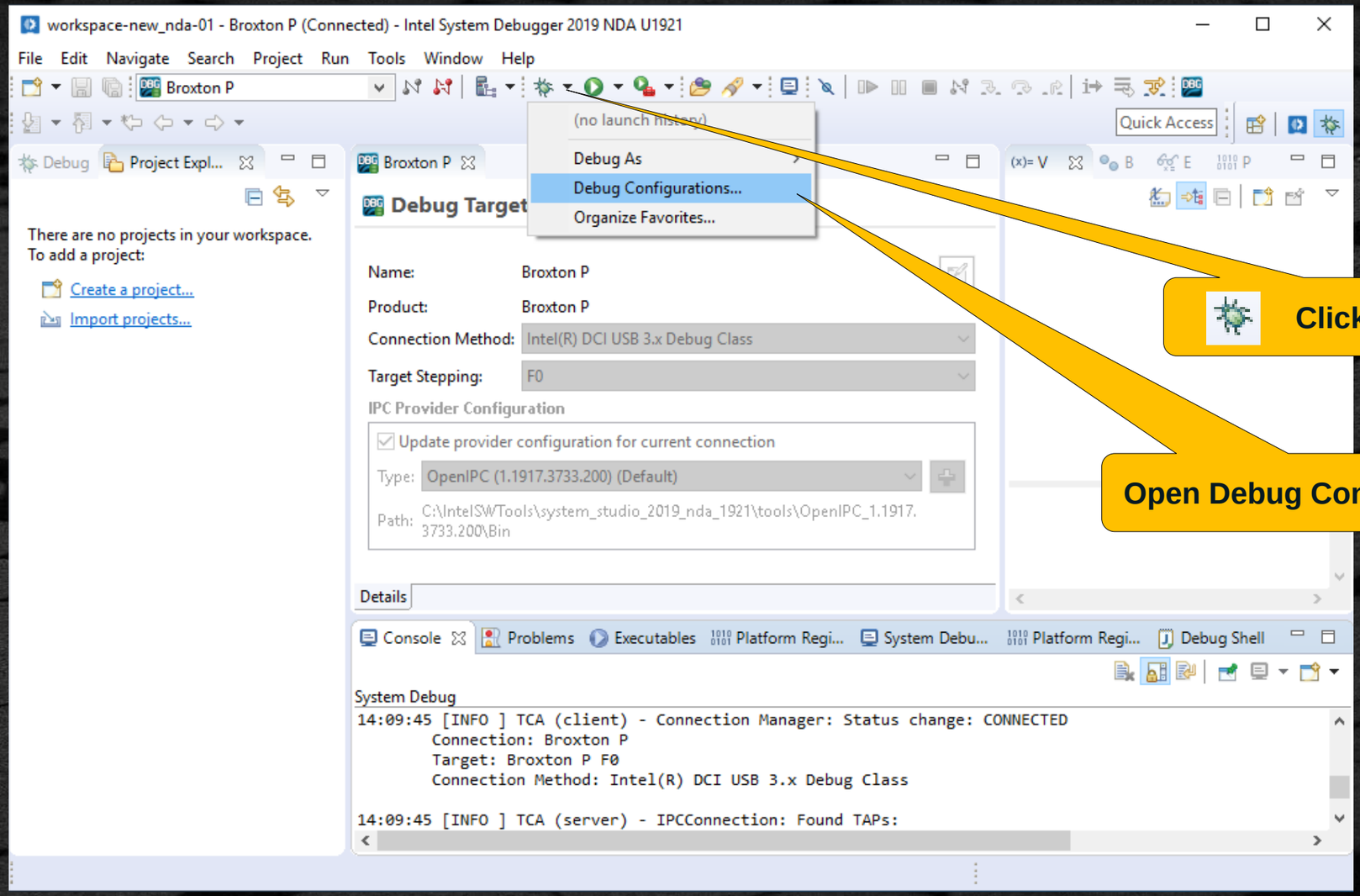


Debug Perspective

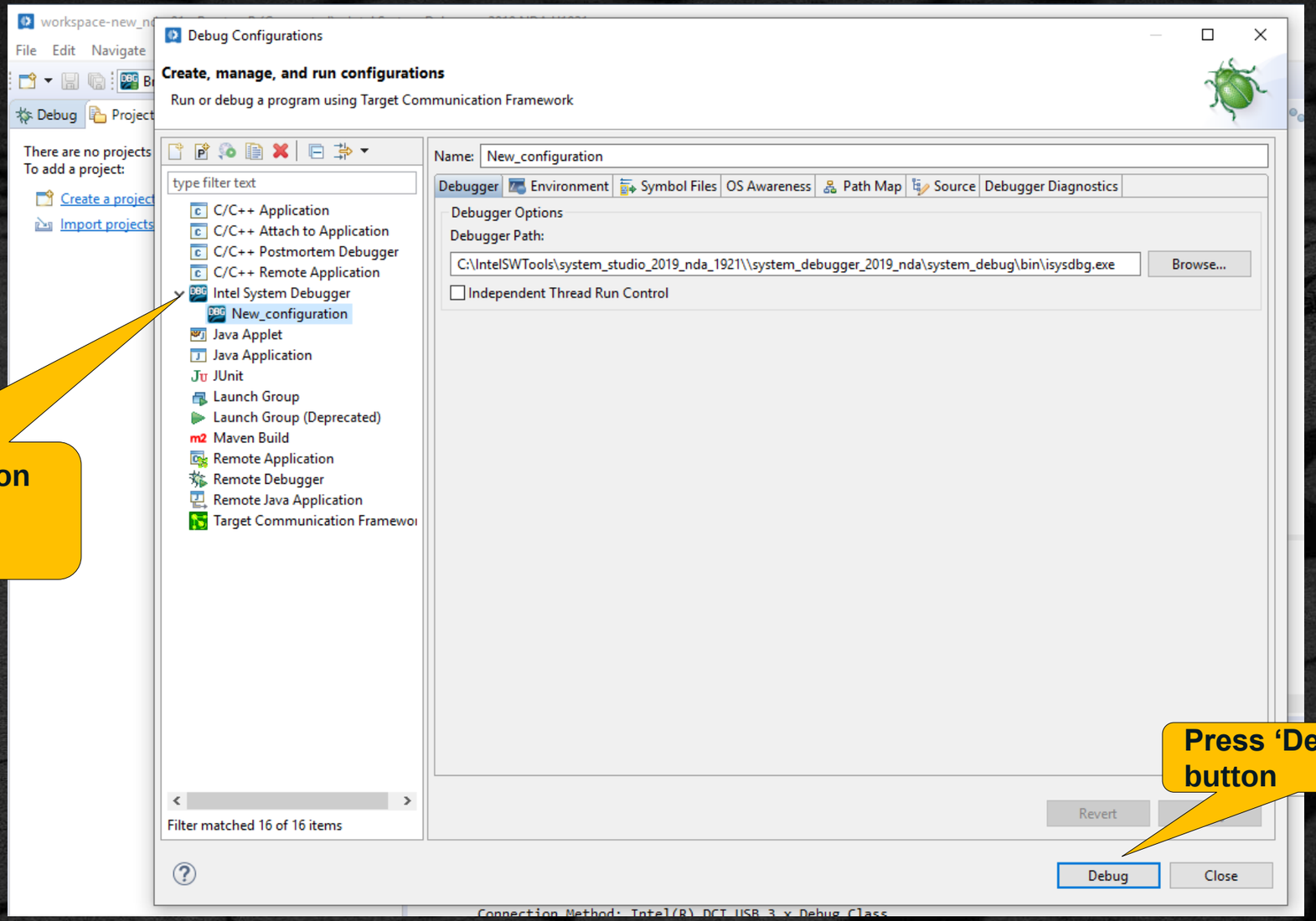


Launch System Debug - TCF

Select Debug Configurations

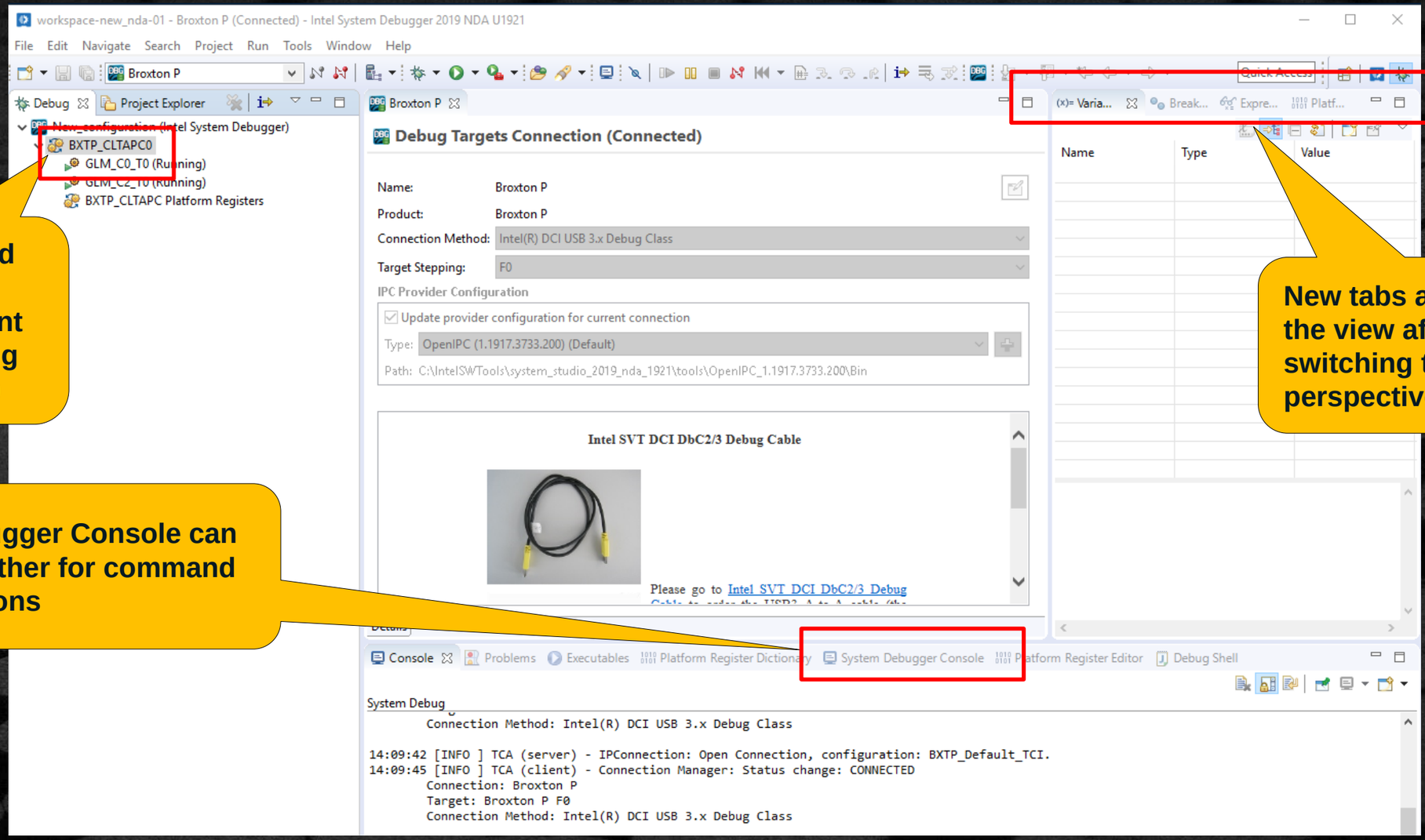


Launch System Debug - TCF



Debug Perspective - TCF

Now Debugging Target



Cores are listed in Debug tab showing current status (Running or Suspended)

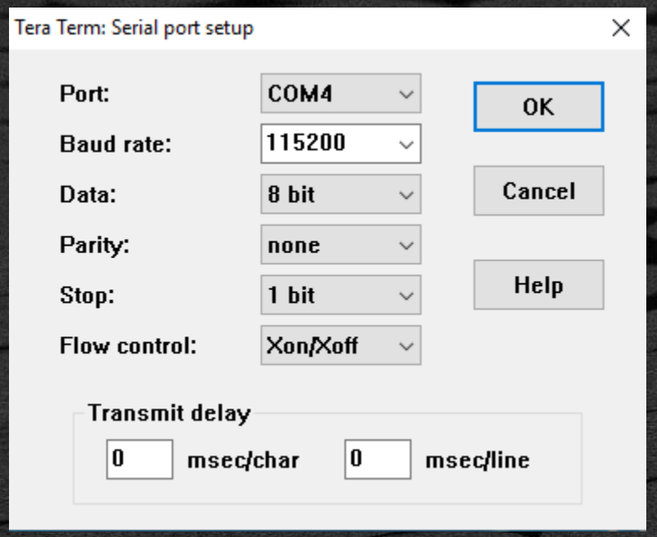
New tabs are added in the view after switching to debug perspective

System Debugger Console can be used together for command line instructions

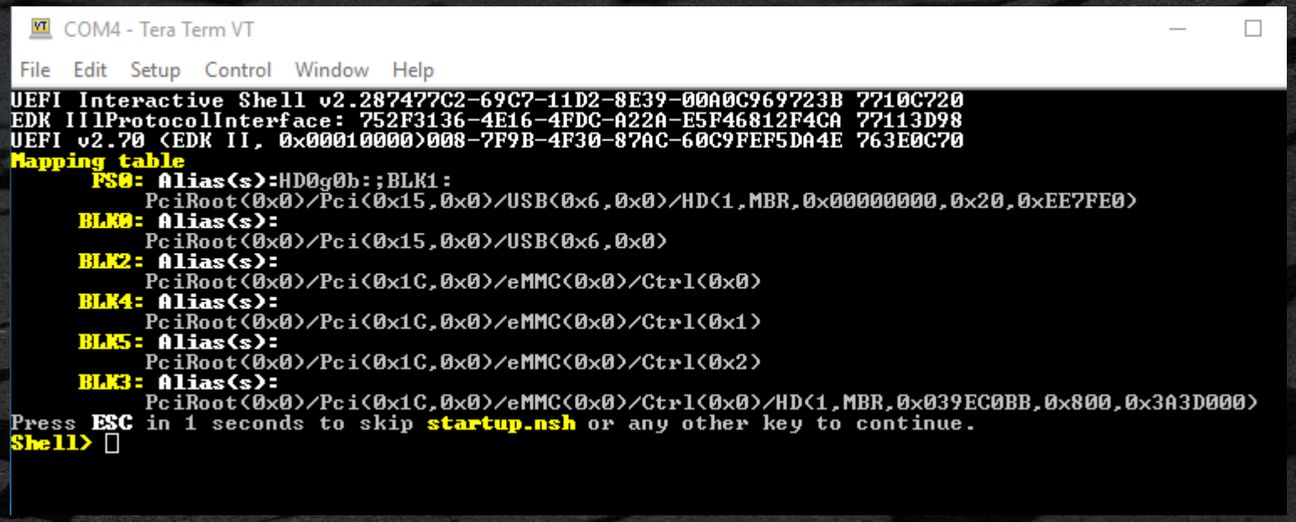
Serial Debug Output Example

Tera Term used as example:

Shows target currently at the UEFI Shell Prompt.

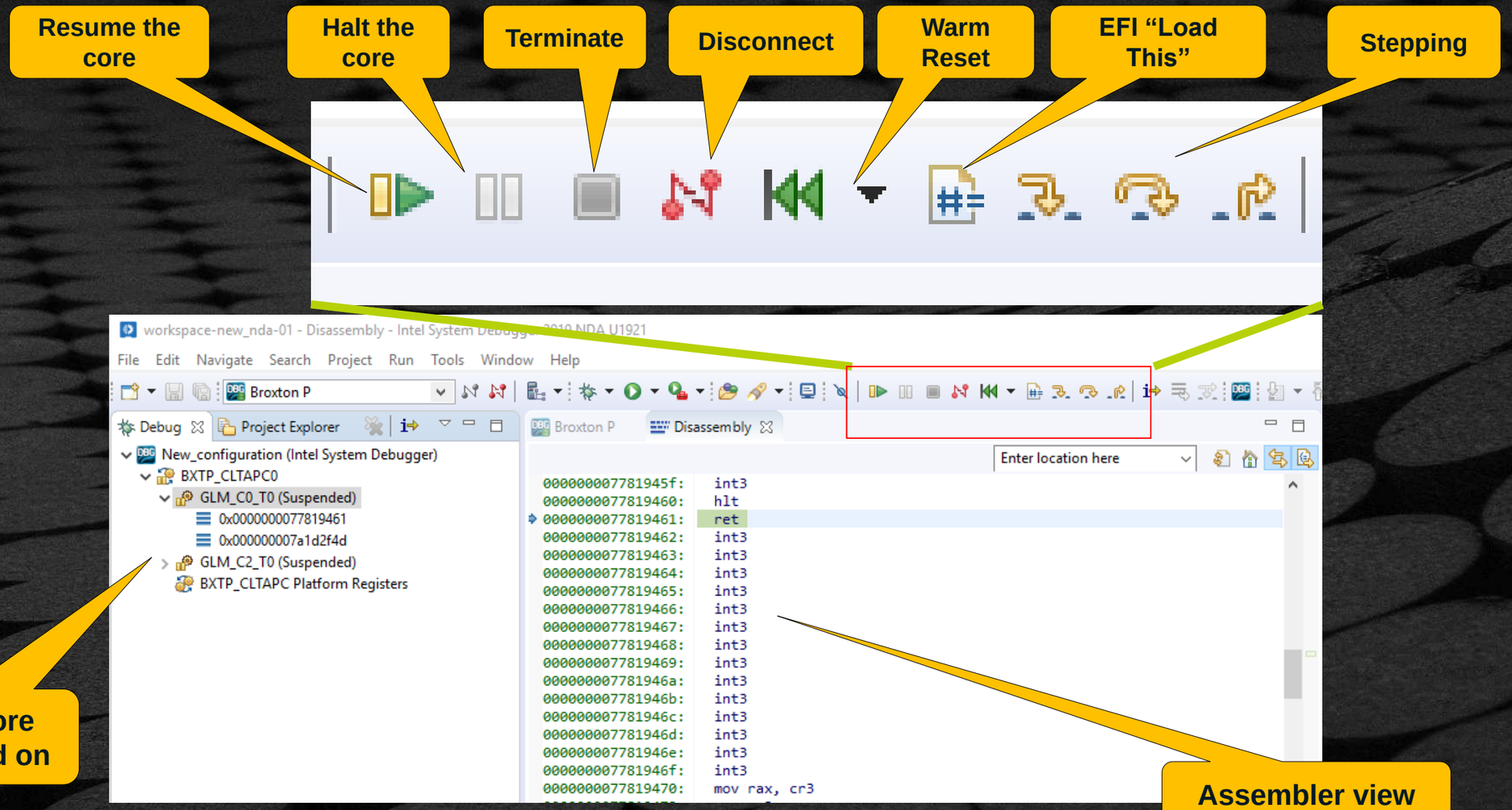


Serial port configuration on Host



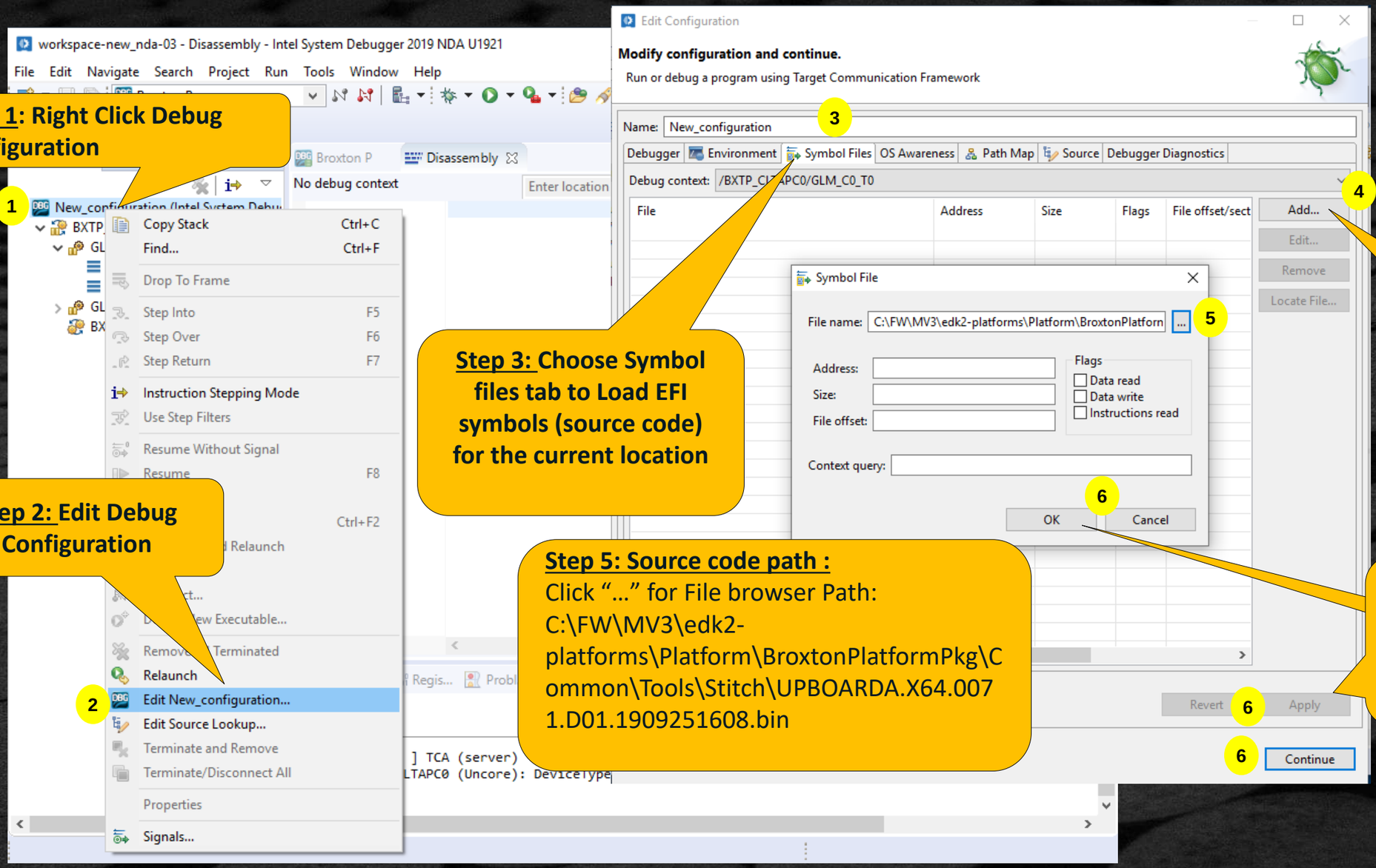
Basic Debug Buttons - TCF

Core status and Disassembly



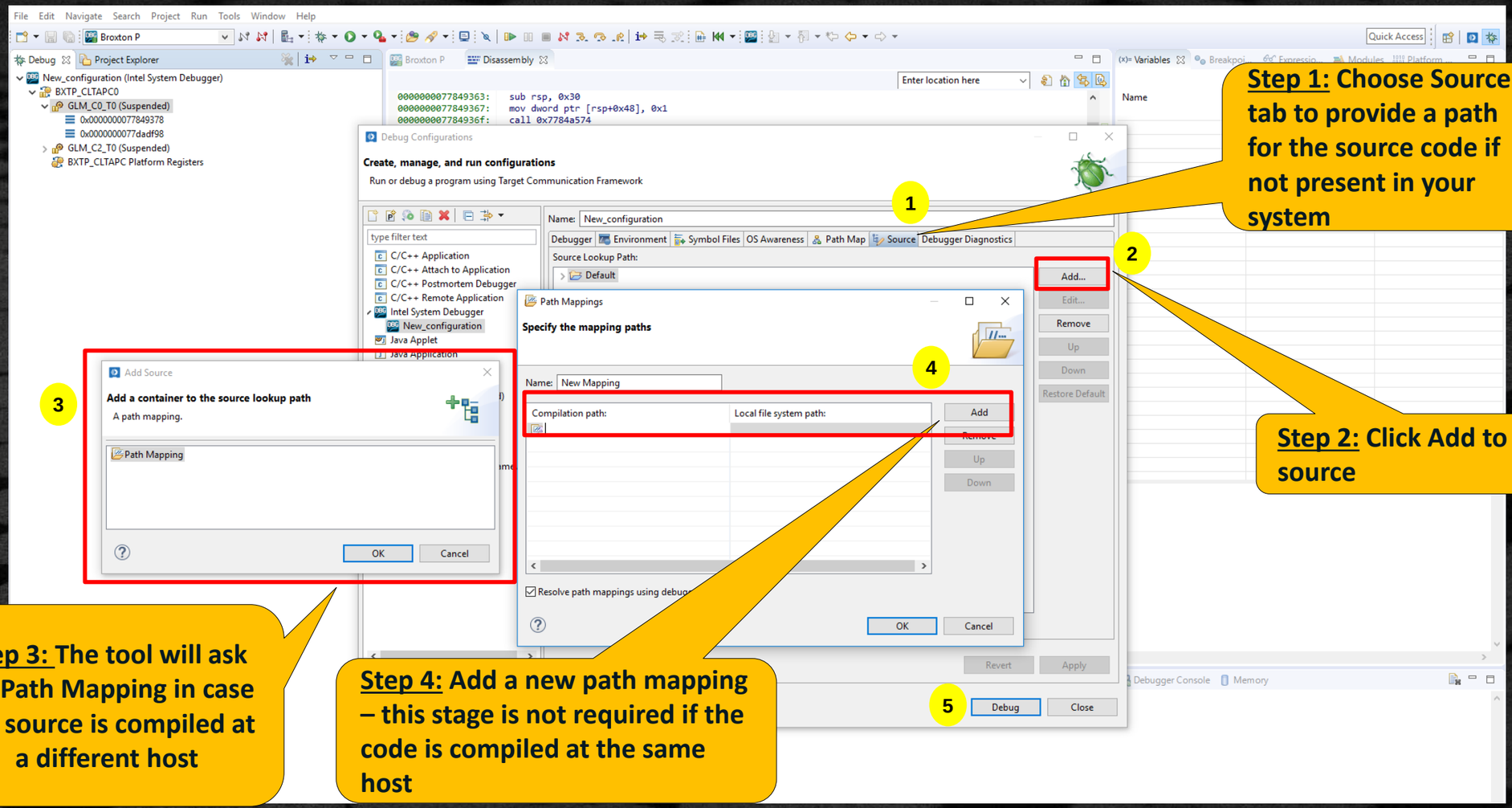
After pressing the “Halt” button

Load source codes



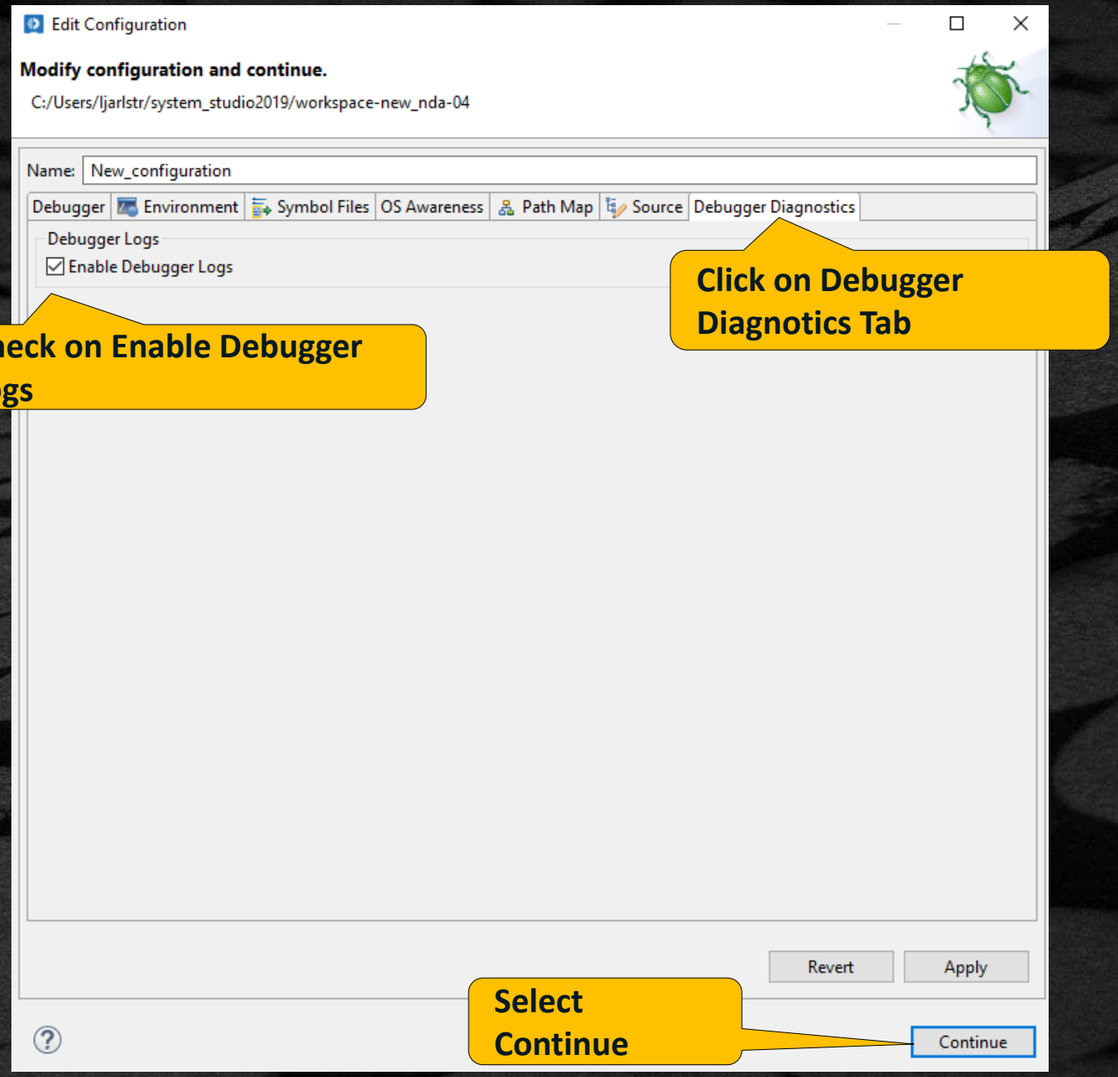
Load Source Code – Not Built on Host

Source code compiled on a different host system



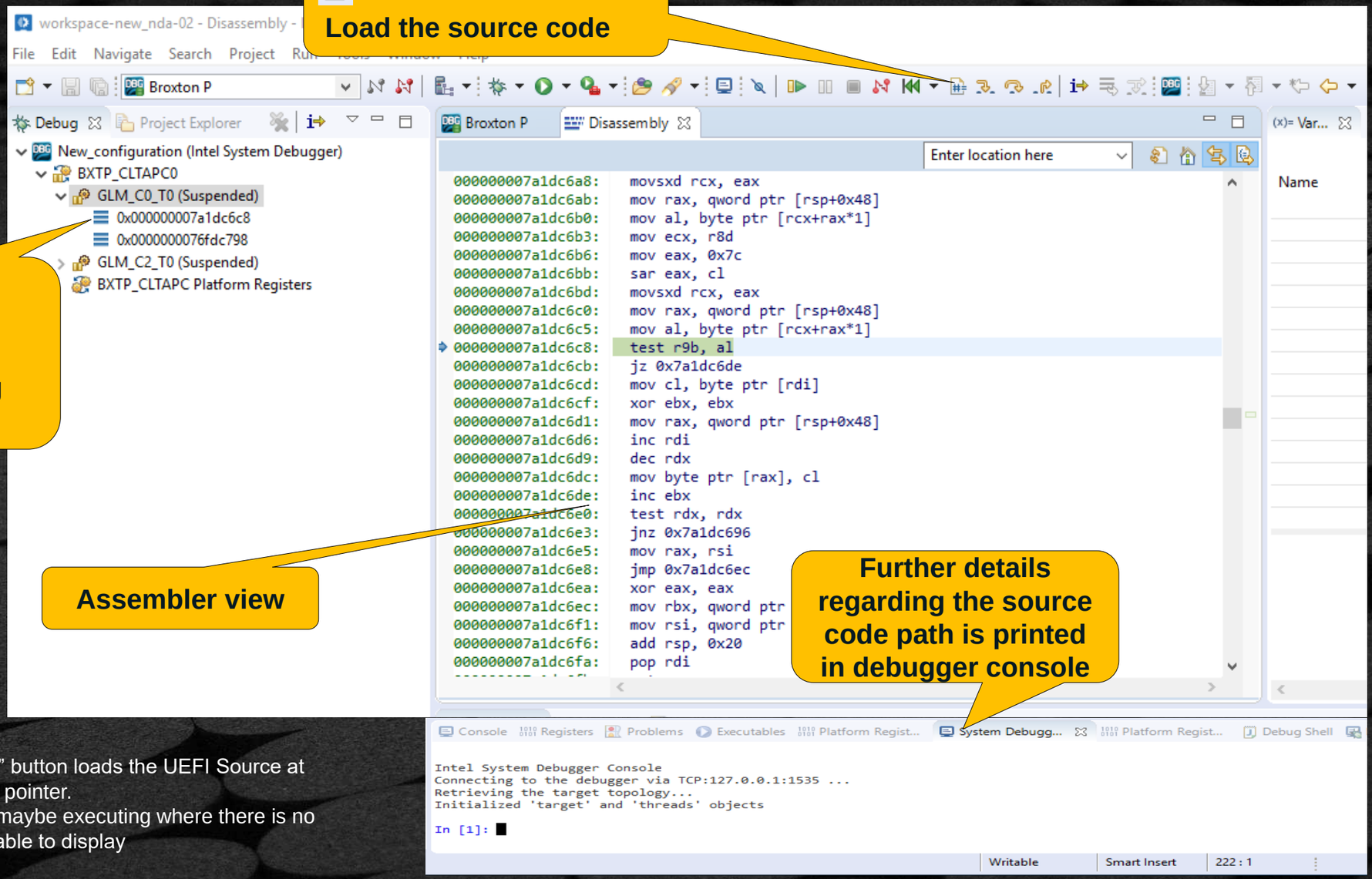
Enabling Debugger Diagnostic

- Enabling Debugger logs are necessary in case you face a problem while connecting to the target. Will help the engineering team to debug that issue
- Debugger logs can be enabled in the "Debugger Diagnostics" tab in the launch configuration. The debugger log files (isysdbg-<timestamp>.log) will be created in the Eclipse workspace folder.



Load the Source for UEFI

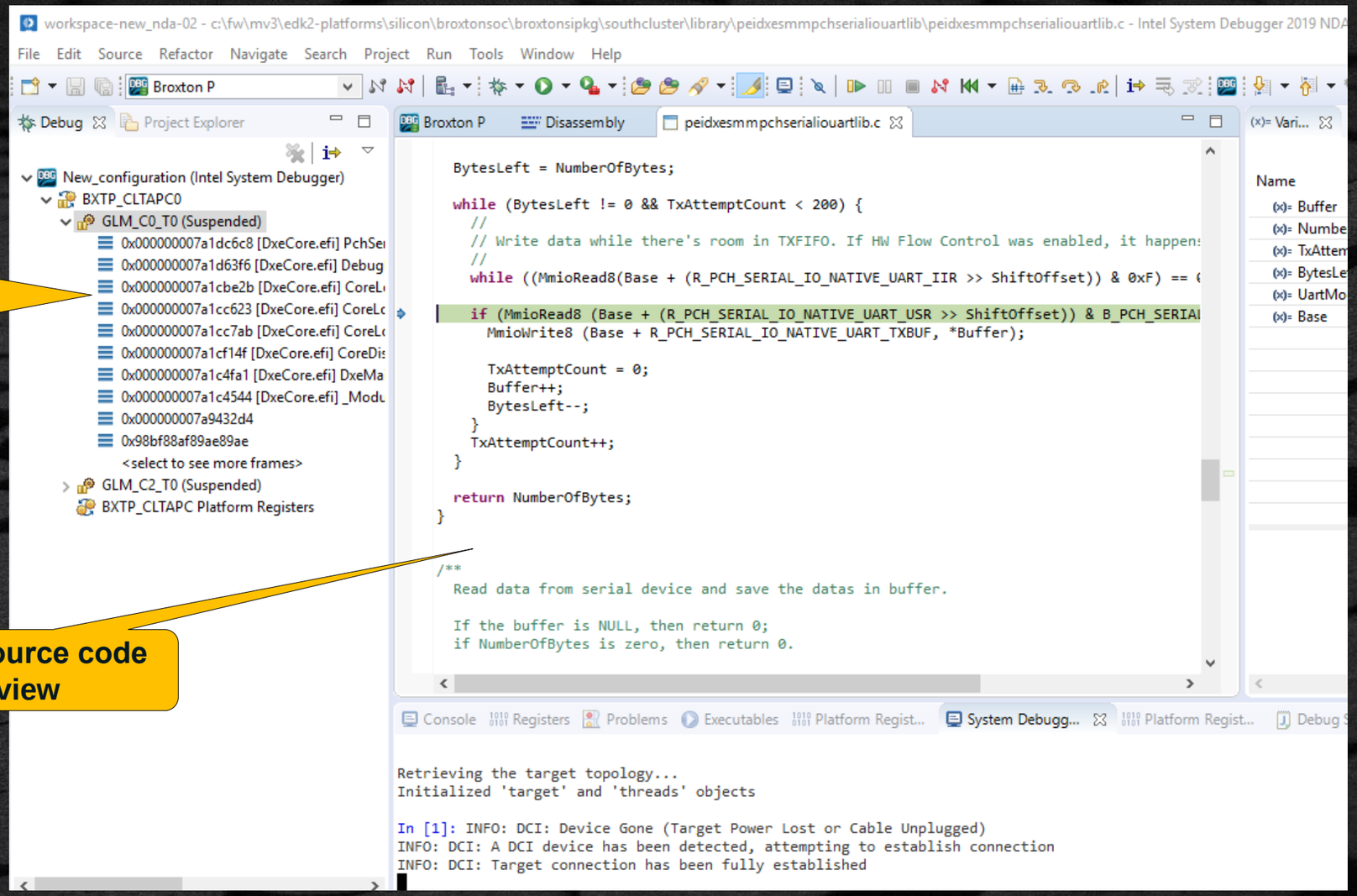
“Load This” Button



Note: The “Load This” button loads the UEFI Source at the current Instruction pointer.
Some UEFI Modules maybe executing where there is no “C” source code available to display

Load the Source for UEFI

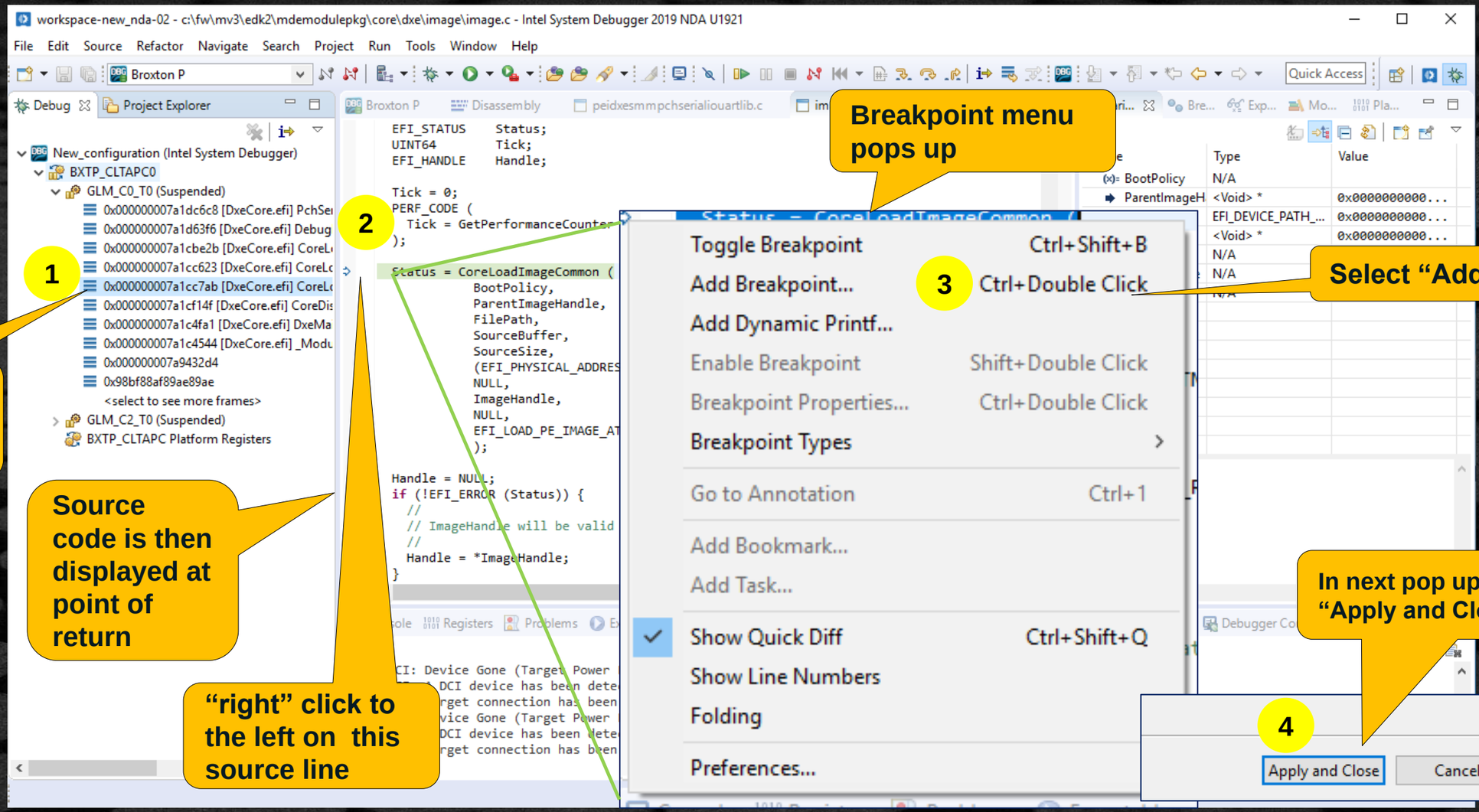
After “Load This” Button – notice the stack now has information on “DxeCore.efi”



Stack information Now shows UEFI Modules

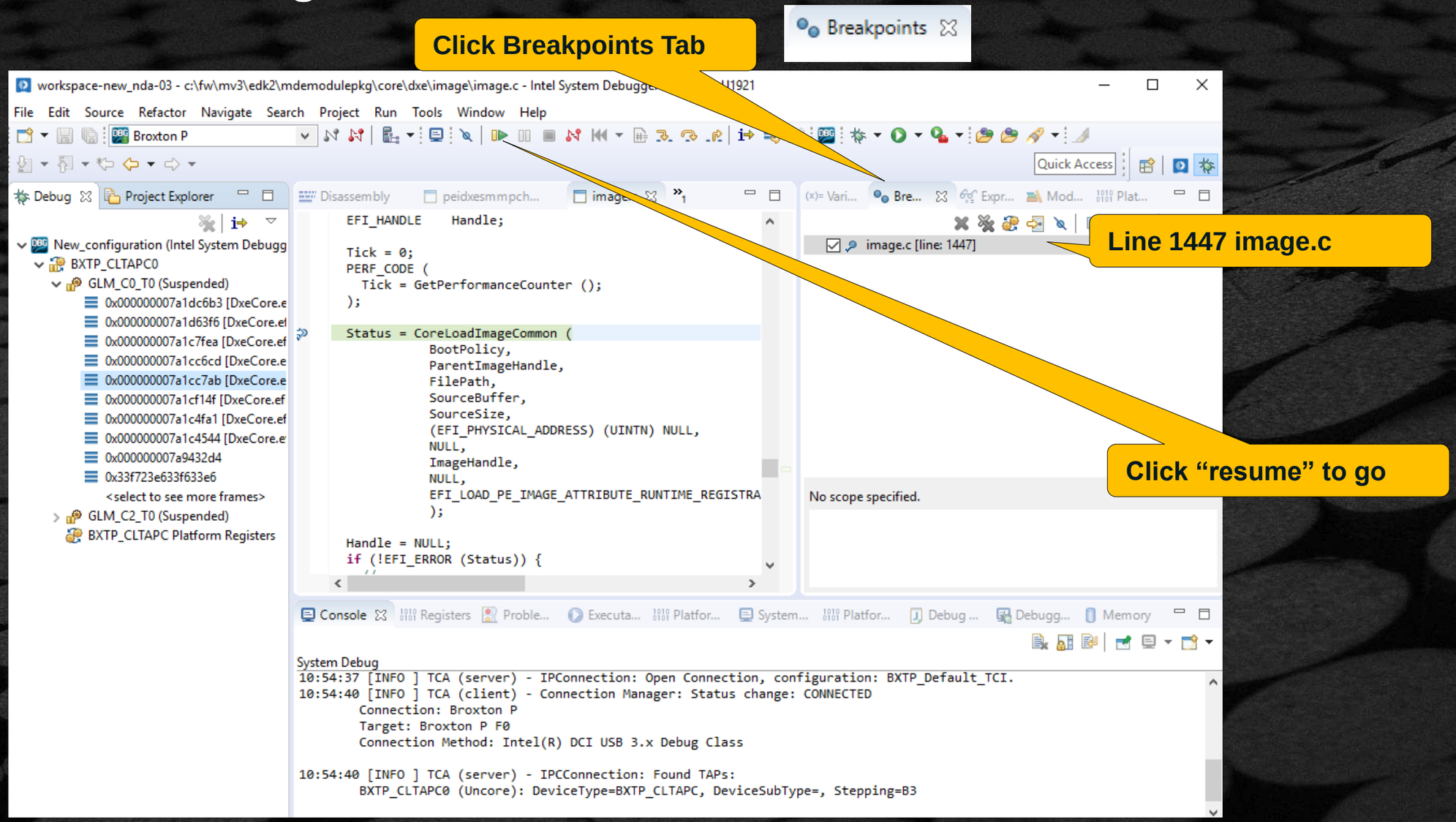
“C” Source code view

Set a Breakpoint Using the Stack



Check the Breakpoints Tab

Breakpoint set in `image.c` line 1447



Observe at the Breakpoint

Serial Debug Output shows execution of each DXE module

COM4 - Tera Term VT

File Edit Setup Control Window Help

```
PROGRESS CODE: V3040003 I0
Loading driver BB65942B-521F-4EC3-BAF9-A92540CF60D2
InstallProtocolInterface: 5B1B31A1-9562-11D2-8E3F-00A0C969723B 770CB0C0
Loading driver at 0x00077002000 EntryPoint=0x00077002384 SataController.efi
InstallProtocolInterface: BC62157E-3E33-4FEC-9920-2D3B36D750DF 770CBA18
ProtectUefiImageCommon - 0x770CB0C0
  - 0x0000000077002000 - 0x00000000000003CC0
PROGRESS CODE: V3040002 I0
InstallProtocolInterface: 18A031AB-B443-4D1A-A5C0-0C09261E9F71 770058F0
InstallProtocolInterface: 6A7A5CFF-E8D9-4F70-BADA-75AB3025CE14 77005920
PROGRESS CODE: V3040003 I0
Loading driver 19DF145A-B1D4-453F-8507-38816676D7F6
InstallProtocolInterface: 5B1B31A1-9562-11D2-8E3F-00A0C969723B 770CB540
Loading driver at 0x00076FF2000 EntryPoint=0x00076FF2384 AtaBusDxe.efi
InstallProtocolInterface: BC62157E-3E33-4FEC-9920-2D3B36D750DF 770CAF98
ProtectUefiImageCommon
  - 0x
InstallProtocolInterface: 18A031AB-B443-4D1A-A5C0-0C09261E9F71 76FF8C80
InstallProtocolInterface: 107A772C-D5E1-11D4-9A46-0090273FC14D 76FF8EA8
InstallProtocolInterface: 6A7A5CFF-E8D9-4F70-BADA-75AB3025CE14 76FF8E90
PROGRESS CODE: V3040003 I0
Loading driver 5E523CB4-D397-4986-87BD-A6DD8B22F455
```

COM4 - Tera Term VT

File Edit Setup Control Window Help

```
Loading driver 19DF145A-B1D4-453F-8507-38816676D7F6
InstallProtocolInterface: 5B1B31A1-9562-11D2-8E3F-00A0C969723B 770CB540
Loading driver at 0x00076FF2000 EntryPoint=0x00076FF2384 AtaBusDxe.efi
InstallProtocolInterface: BC62157E-3E33-4FEC-9920-2D3B36D750DF 770CAF98
ProtectUefiImageCommon
  - 0x0000000076FF2000 - 0x0000000000007660
PROGRESS CODE: V3040002 I0
InstallProtocolInterface: 18A031AB-B443-4D1A-A5C0-0C09261E9F71 76FF8C80
InstallProtocolInterface: 107A772C-D5E1-11D4-9A46-0090273FC14D 76FF8EA8
InstallProtocolInterface: 6A7A5CFF-E8D9-4F70-BADA-75AB3025CE14 76FF8E90
PROGRESS CODE: V3040003 I0
Loading driver 5E523CB4-D397-4986-87BD-A6DD8B22F455
InstallProtocolInterface: 5B1B31A1-9562-11D2-8E3F-00A0C969723B 770CA840
Loading driver at 0x00076FDA000 EntryPoint=0x00076FDA3A4 AtaAtapiPassThruDxe.efi
InstallProtocolInterface: BC62157E-3E33-4FEC-9920-2D3B36D750DF 770CA718
ProtectUefiImageCommon - 0x770CA840
  - 0x0000000076FDA000 - 0x
PROGRESS CODE: V3040002 I0
InstallProtocolInterface: 18A031AB-B443-4D1A-A5C0-0C09261E9F71 76FE4650
InstallProtocolInterface: 107A772C-D5E1-11D4-9A46-0090273FC14D 76FE4680
InstallProtocolInterface: 6A7A5CFF-E8D9-4F70-BADA-75AB3025CE14 76FE4698
PROGRESS CODE: V3040003 I0
Loading driver 0167CCC4-D0F7-4F21-A3EF-9E64B7CDCE8B
```

Last Dxe Module Loaded and Executed

Press “resume”

Next Dxe Module executed

mdemodulepkg\core\dxe\image\image.c - Intel System Debugger 2019 NDA U1921

arch Project Run Tools Window Help

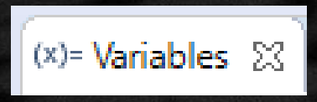
Disassembly peidxesmmpch...

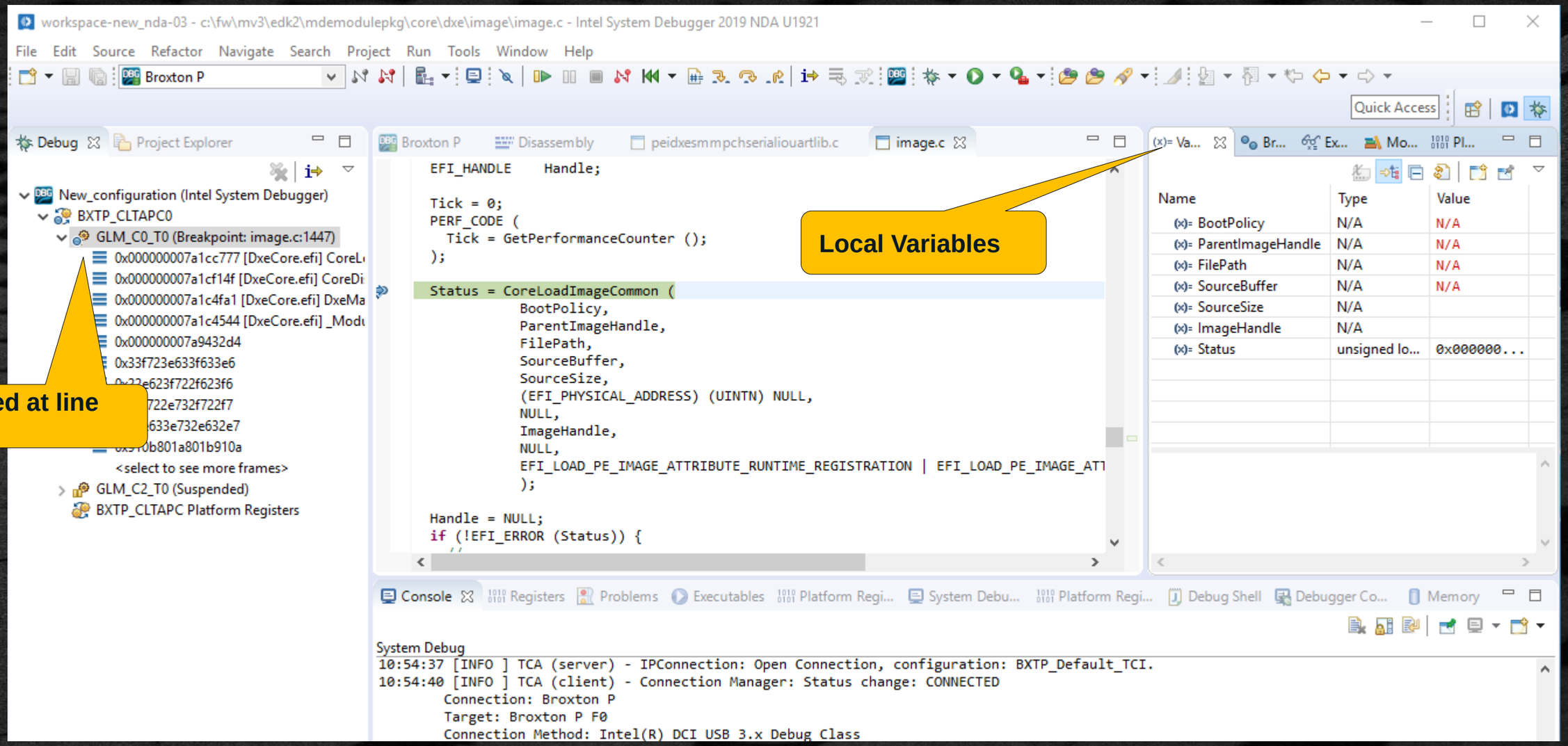
```
EFI_HANDLE    Handle;

Tick = 0;
PERF_CODE (
  Tick = GetPerformanceCounter ();
);
```

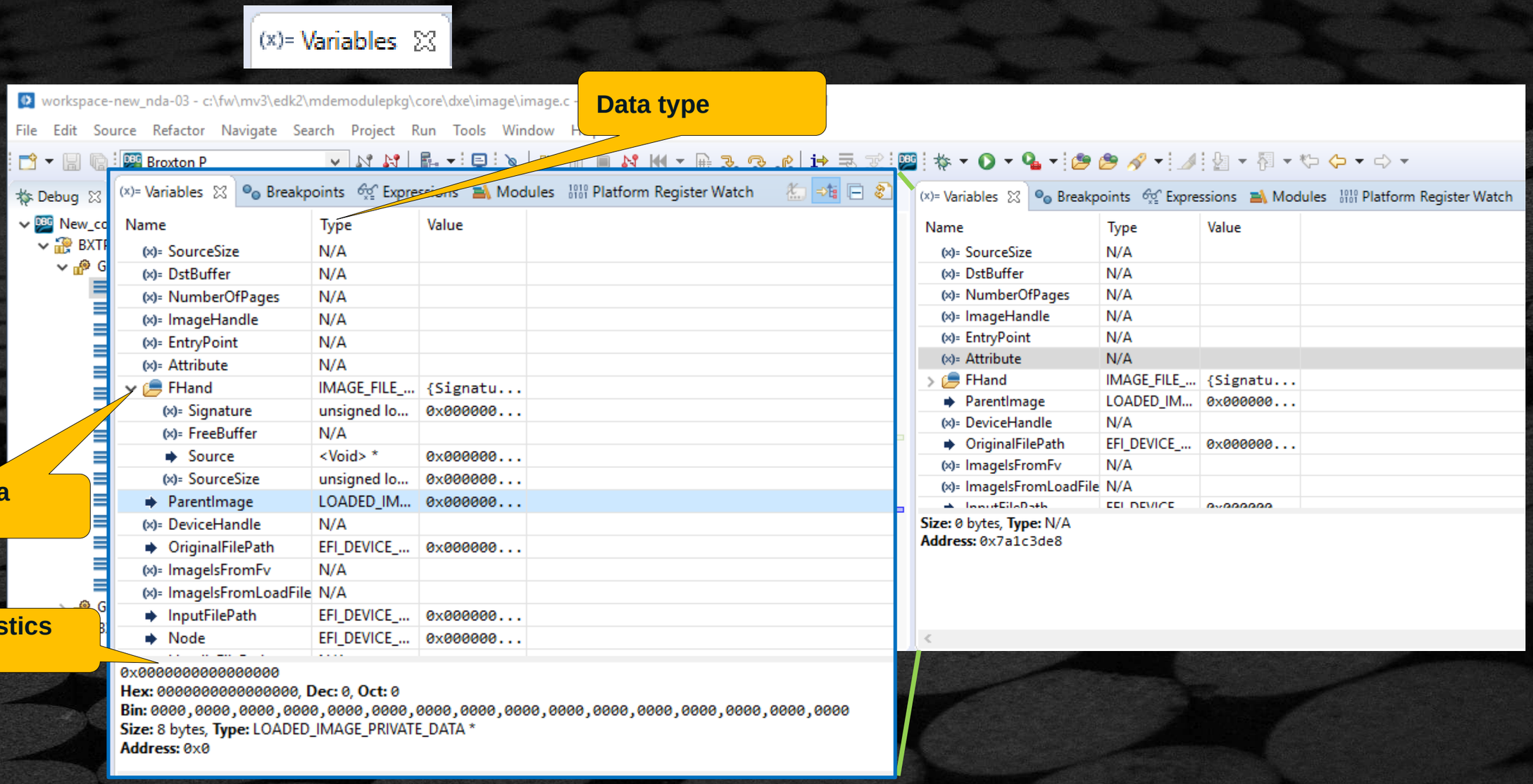
Observe at the Breakpoint

Variable tab

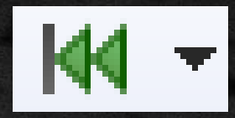




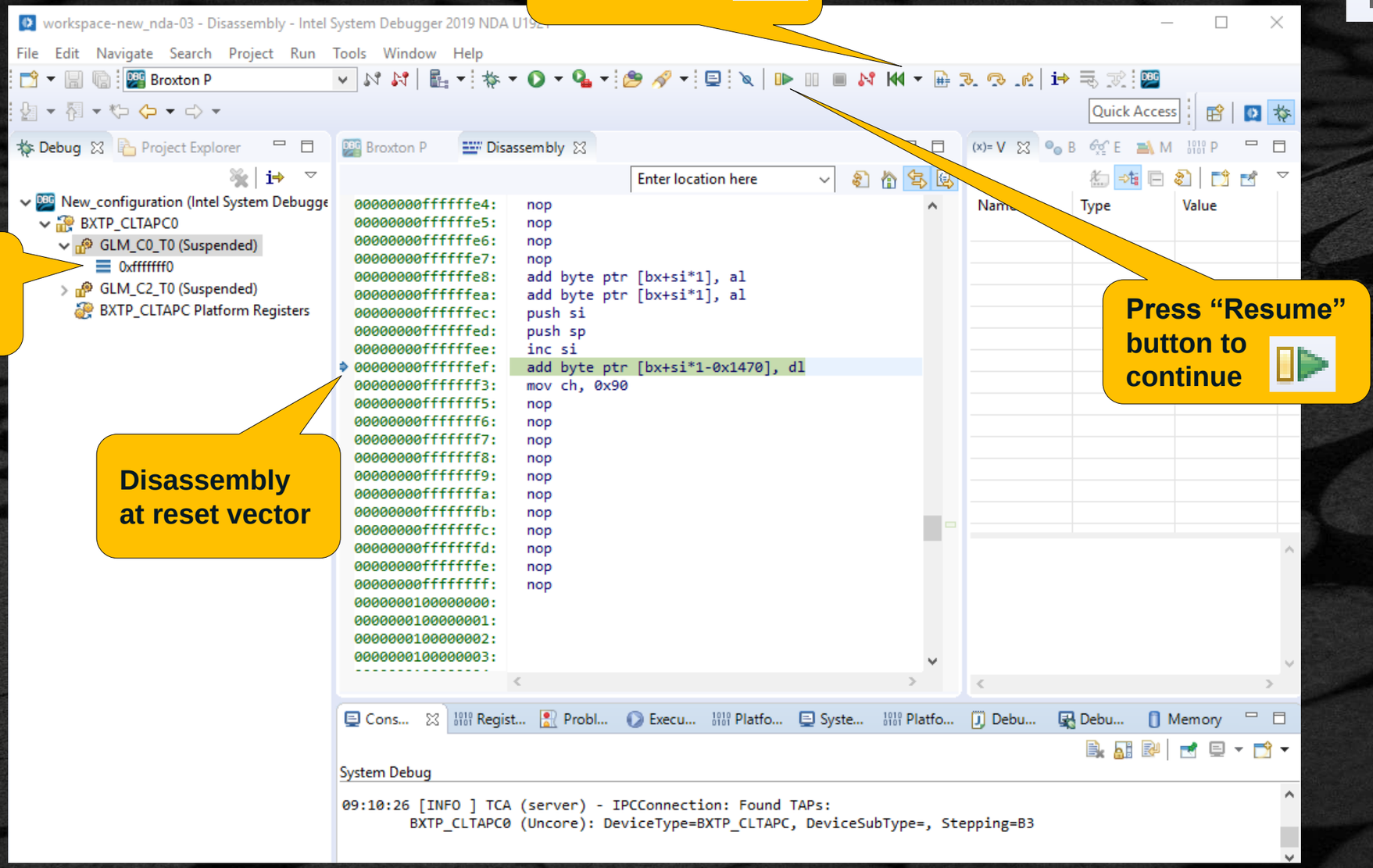
Local Variables



Reset the Target



Press “reset” button 



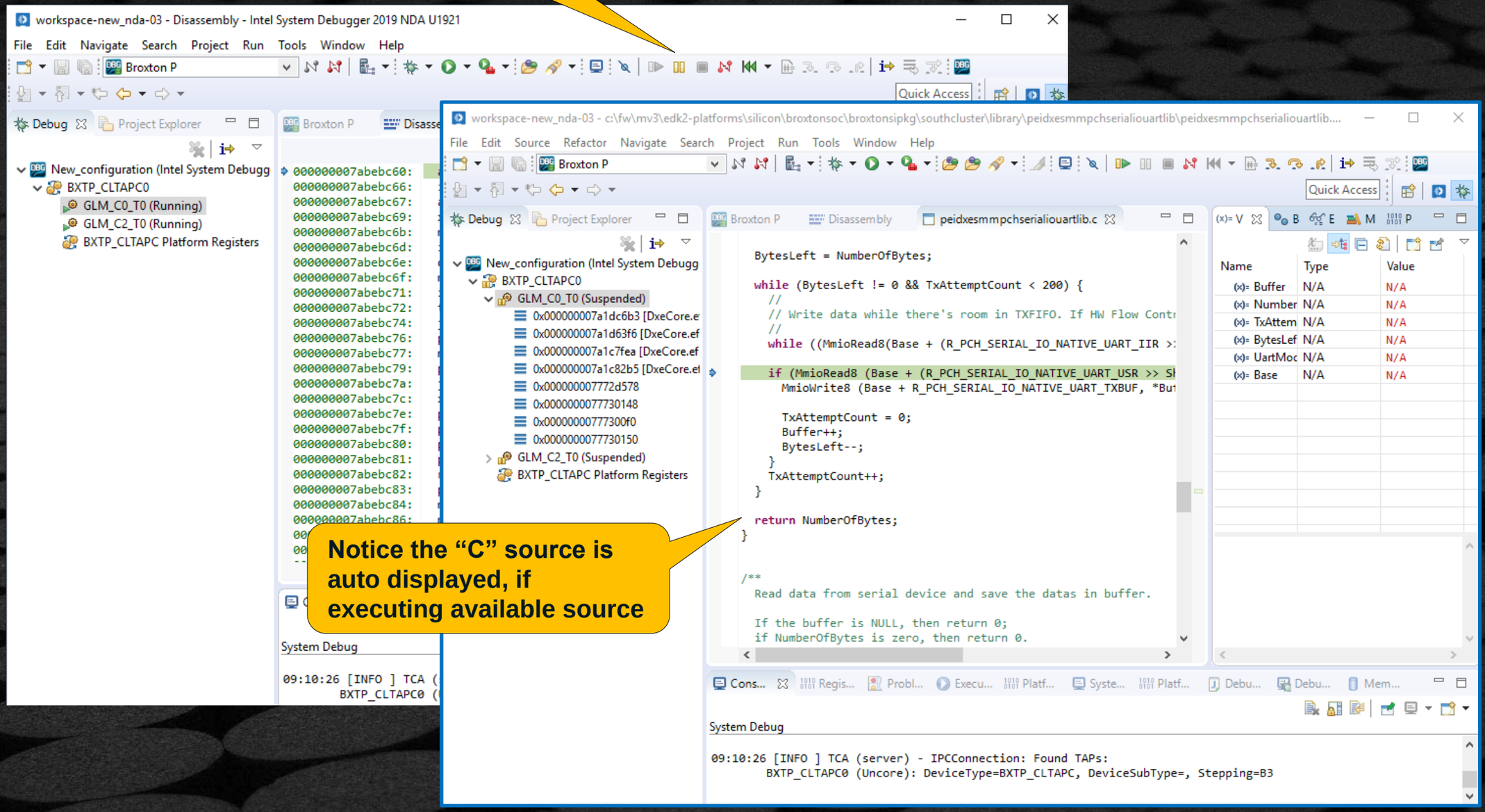
Stack at reset vector

Disassembly at reset vector

Press “Resume” button to continue

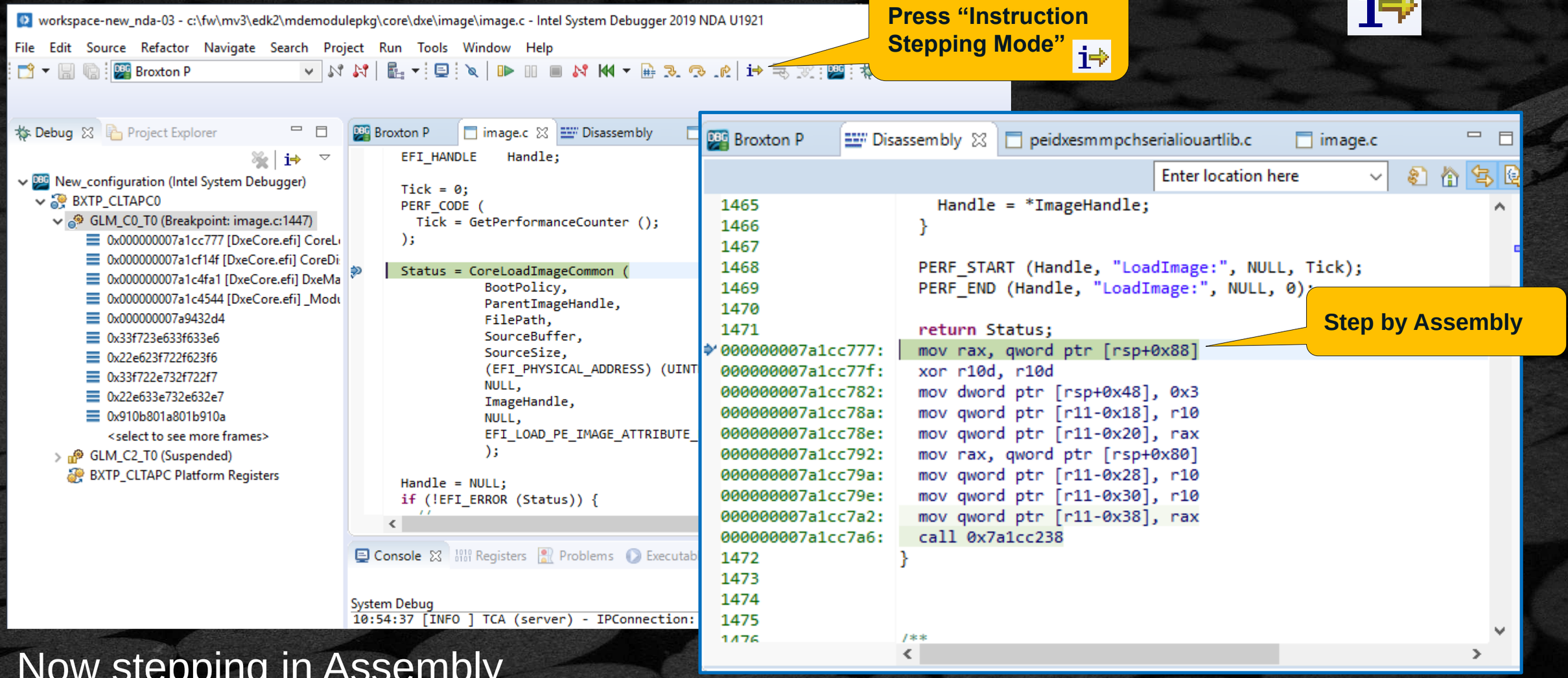
Run then “suspend”, Halt

Press “suspend” button to halt



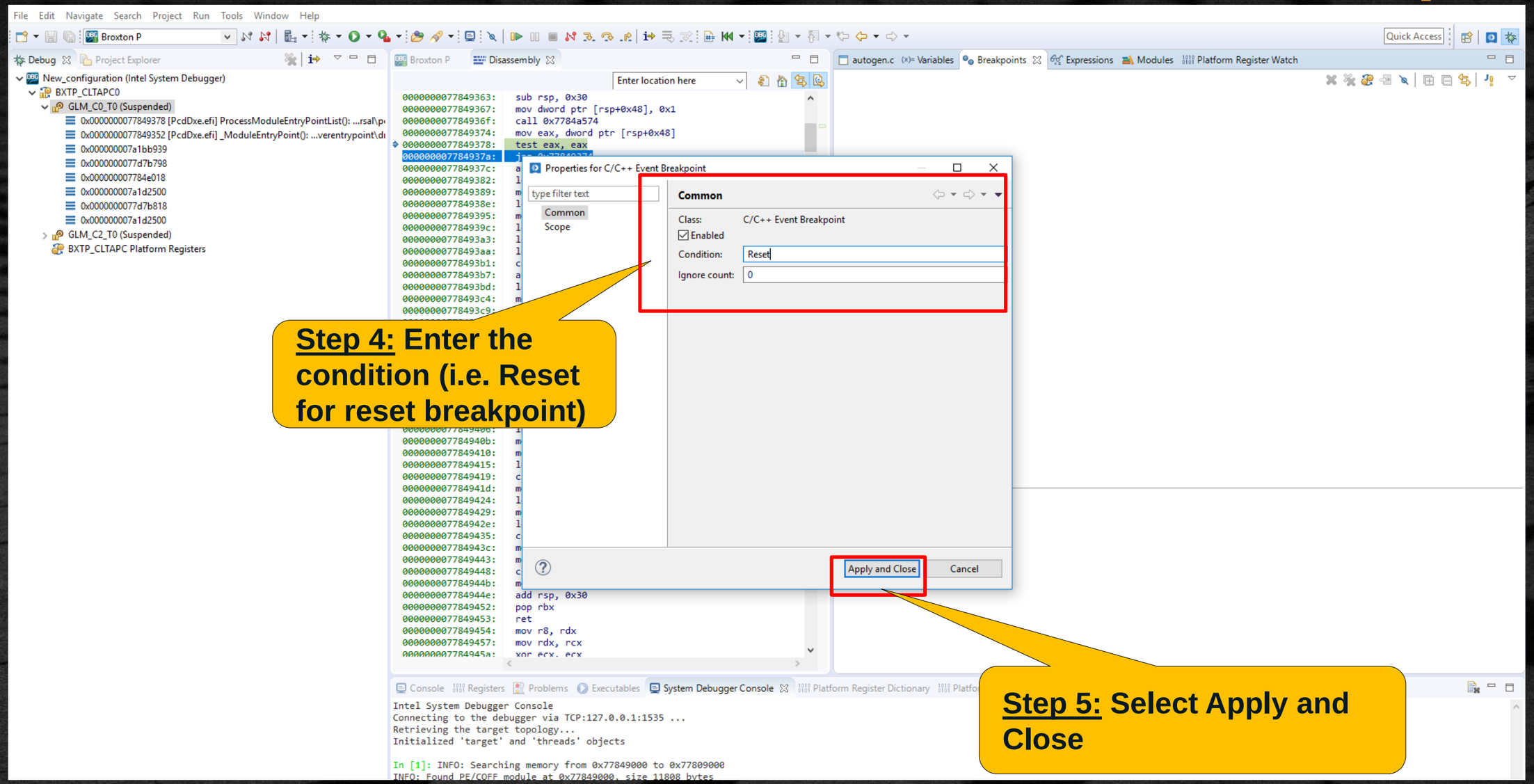
Notice the “C” source is auto displayed, if executing available source

Switch Between Stepping Assembly & “C” Source



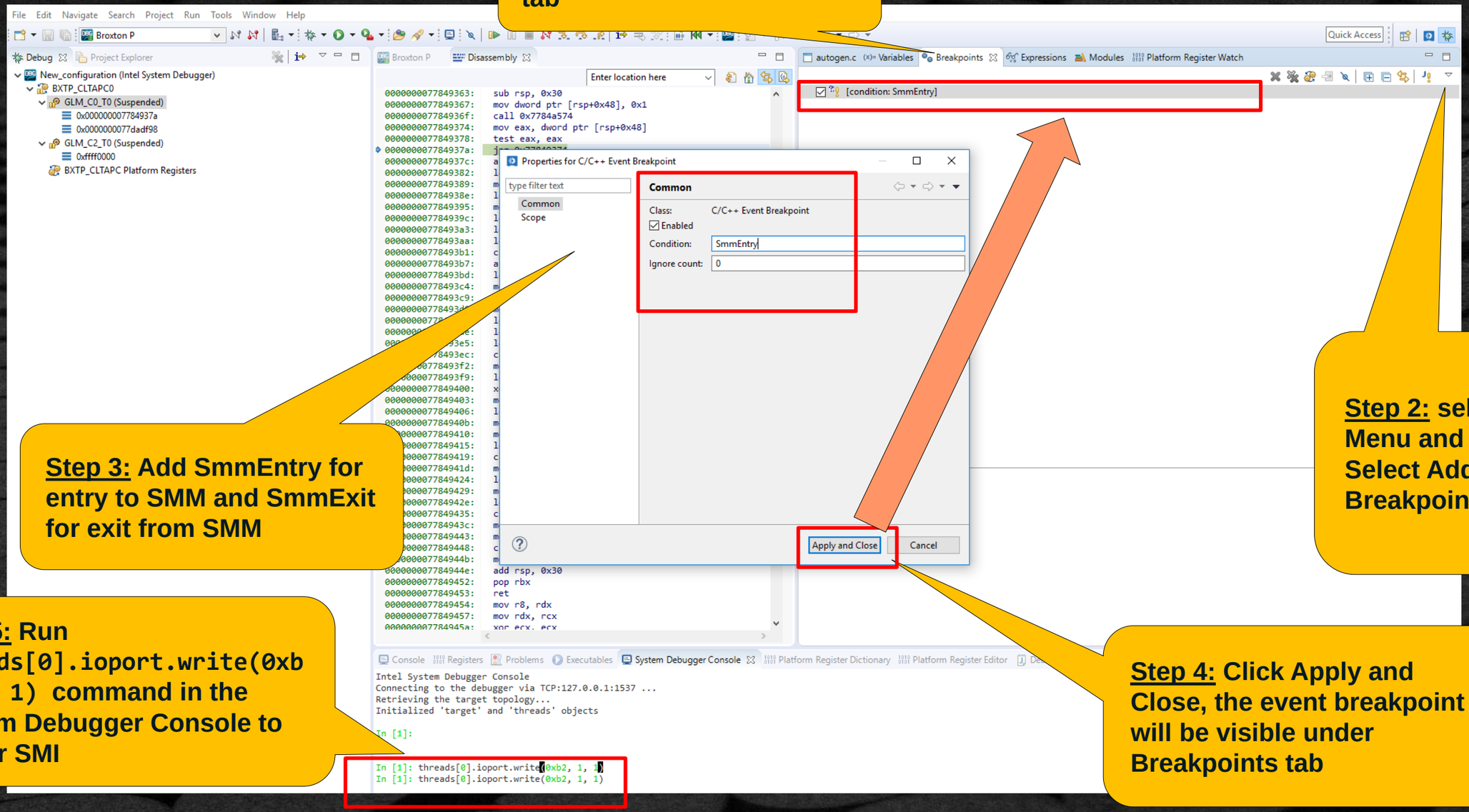
Now stepping in Assembly

Set Event Breakpoints



SMM Debugging

Step 1: Select Breakpoints tab



Step 2: select View Menu and choose Select Add Event Breakpoint (C/C++)

Step 3: Add SmmEntry for entry to SMM and SmmExit for exit from SMM

Step 4: Click Apply and Close, the event breakpoint will be visible under Breakpoints tab

Step 5: Run `threads[0].ioport.write(0xb2, 1, 1)` command in the System Debugger Console to trigger SMI

tianocore

Intel® System Studio Debugger - 2 GUI versions

TCF based debugger

- Target Communication Framework(TCF) works with the Eclipse IDE
- TCF provides a complete modern debugger for C/C++

XDB Based debugger

- Intel ITP based debugger supporting JTAG
- Moving to Legacy debugger by Intel® System Studio

The underlying debugging in similar but the user interface is different

Getting Started The System Debugger -XDB legacy

Step 1:

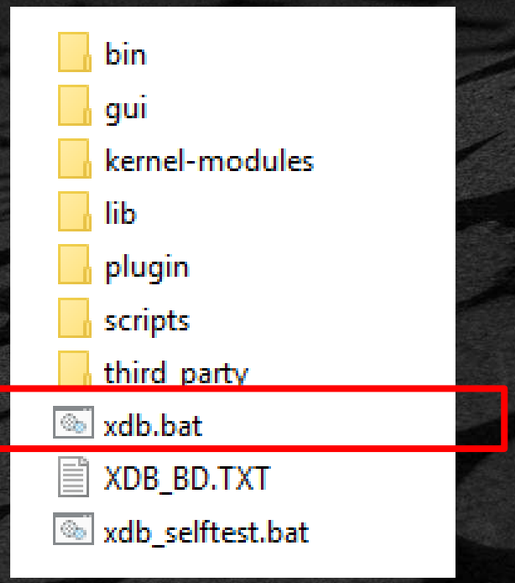
Open the folder where Intel® System Debugger was installed

Step 2:

Select a startup script and launch it - Batch files.

Step 3:

Connect to your target using the “Connect” button (or Ctrl-R)



:

```
opt/intel/system_stud
io_2019/system_debugg
er_2019/system_debug/
xdb.sh
```

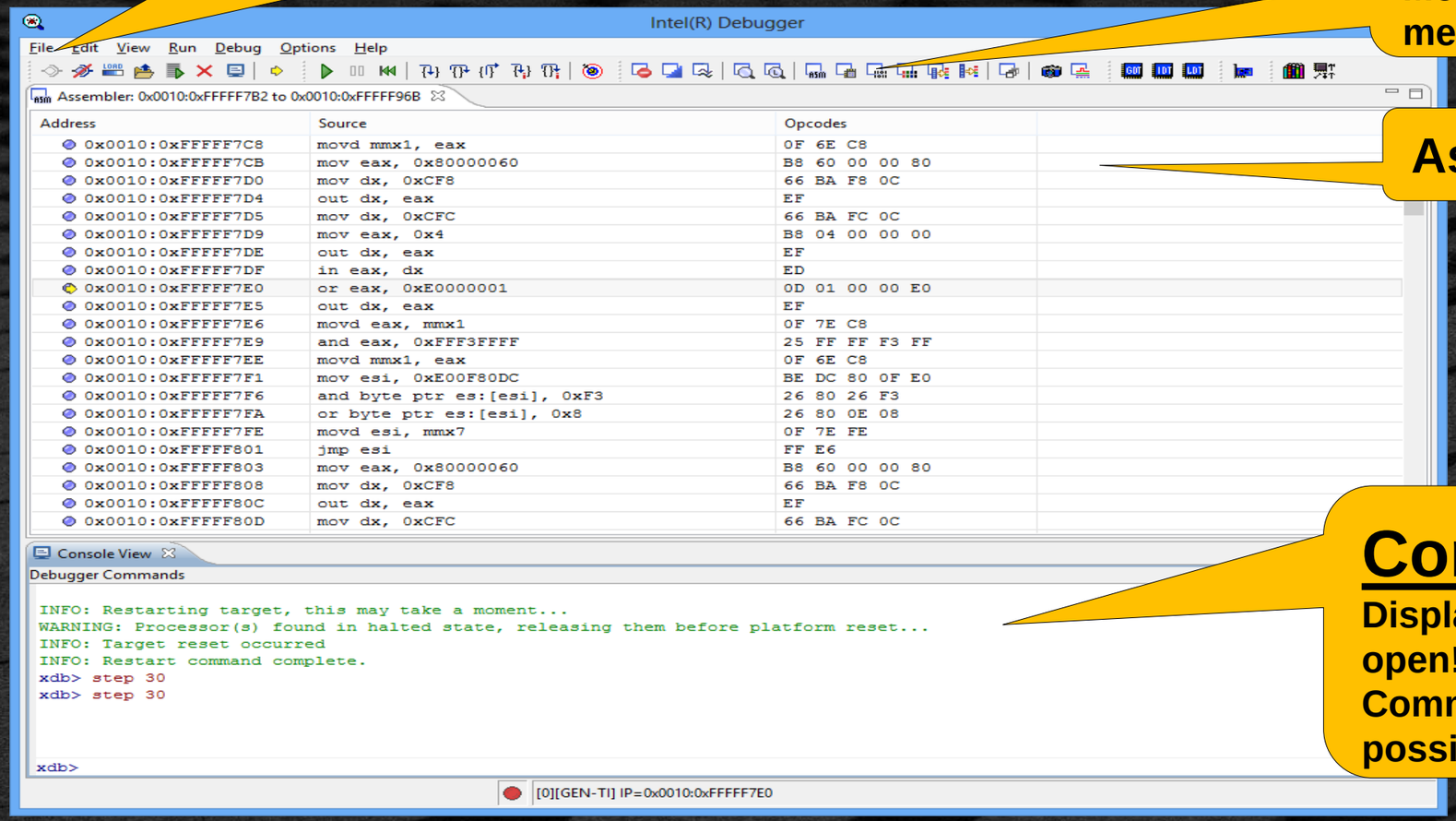
Initial Startup & overview

Connect/Disconnect

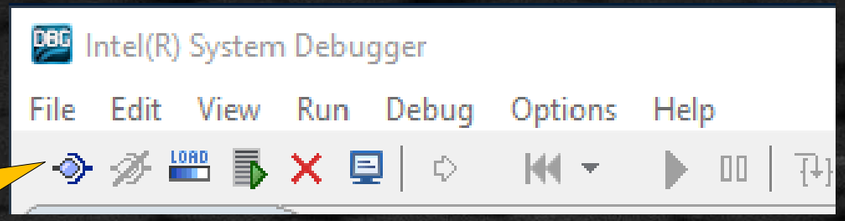
Toolbars / Menus
Most functions available from menus & buttons

Assembly Window

Console Window
Displays status messages (keep it open!)
Command-driven operation also possible



Connecting To Targets



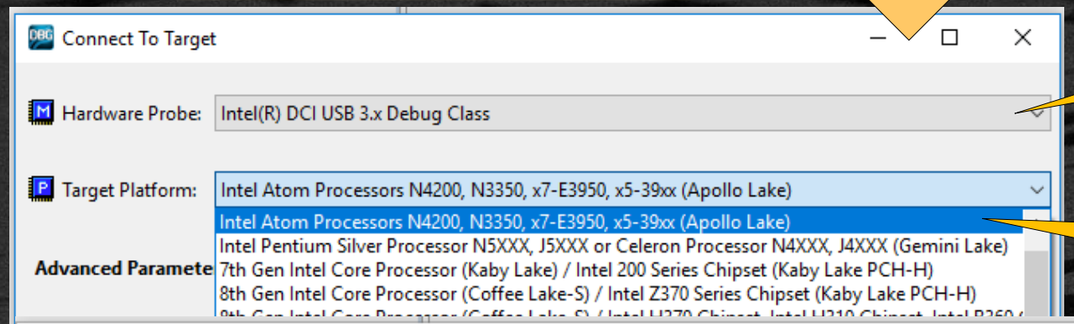
Step 1: Click the connect icon

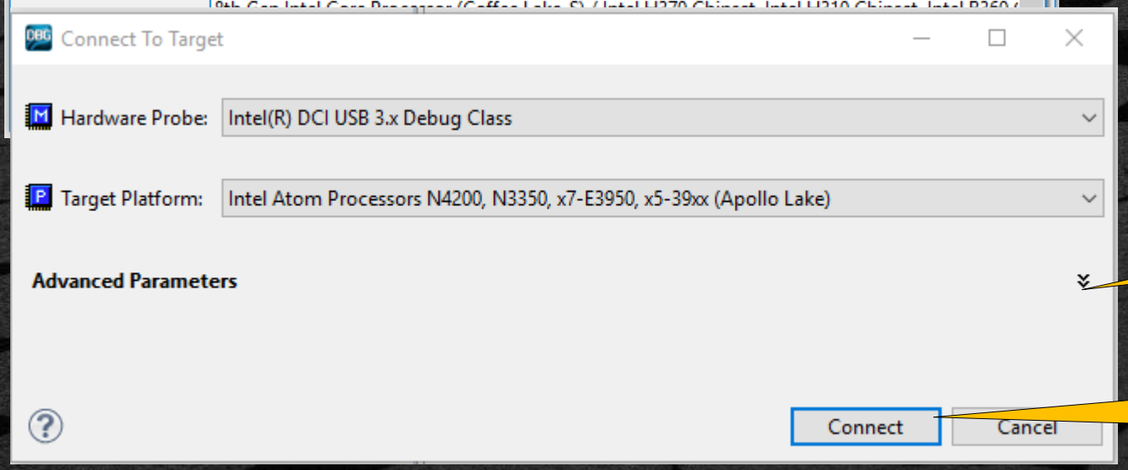
Step 2: Select your probe

Step 3: Select your target platform

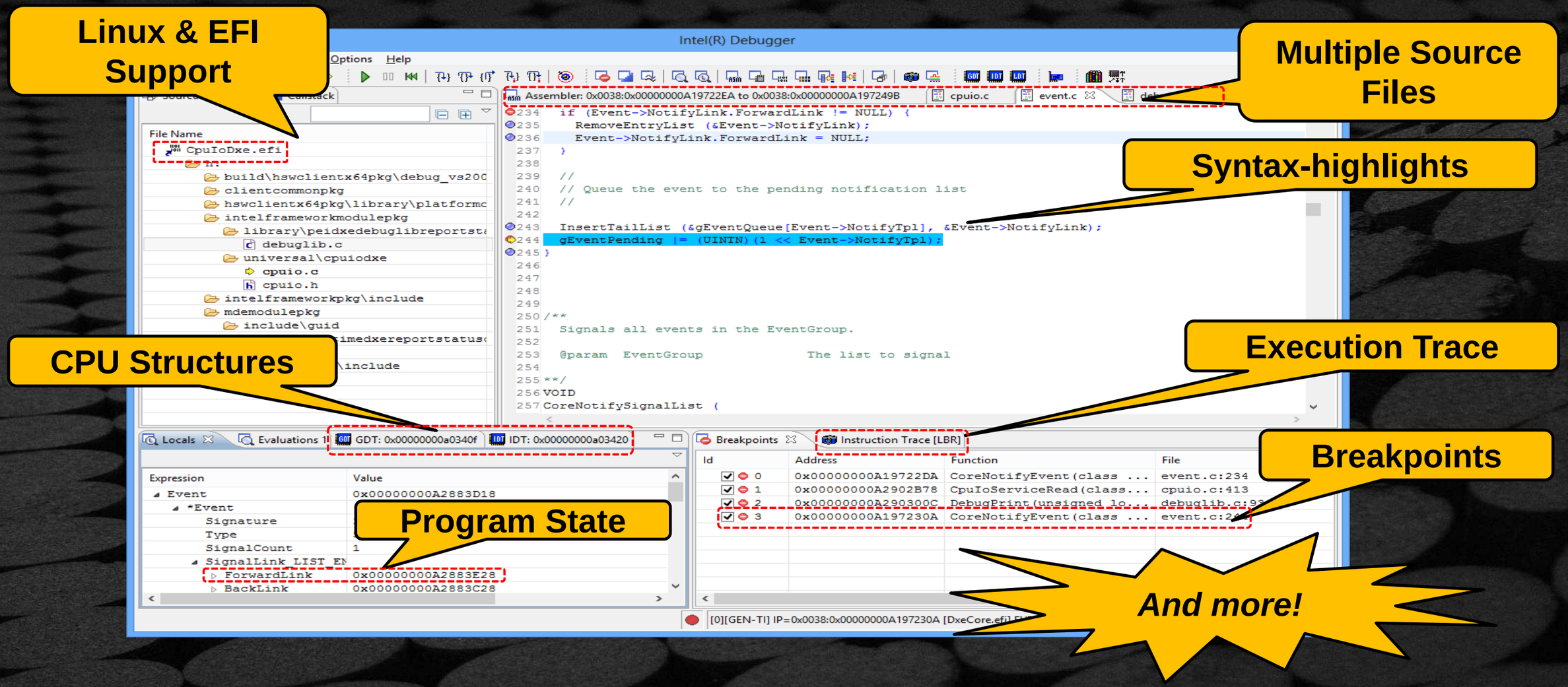
Step 3a (optional): Enter advanced connection parameters

Step 4: Connect!

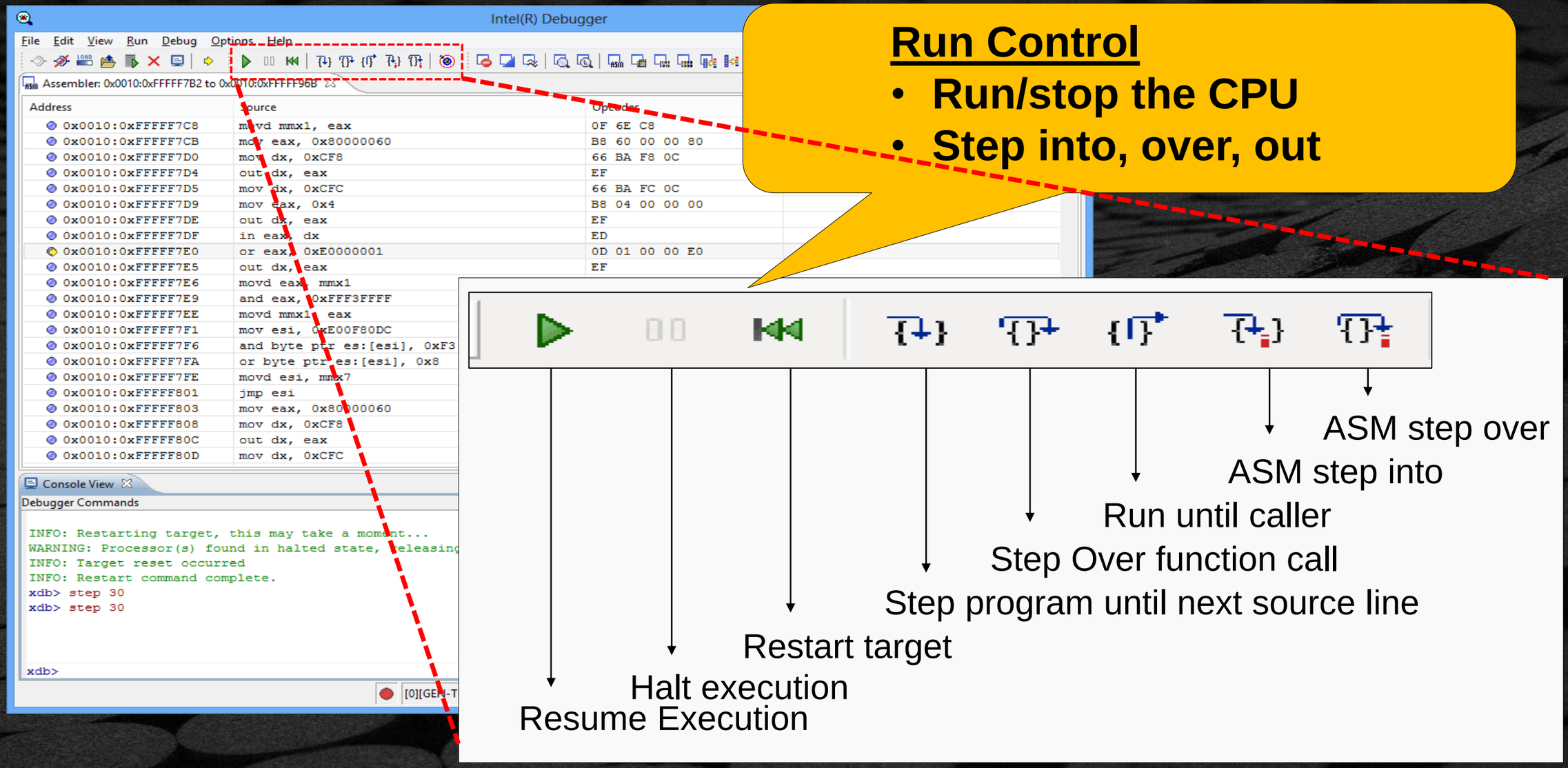




Standard Debugger Features



Target Run Control

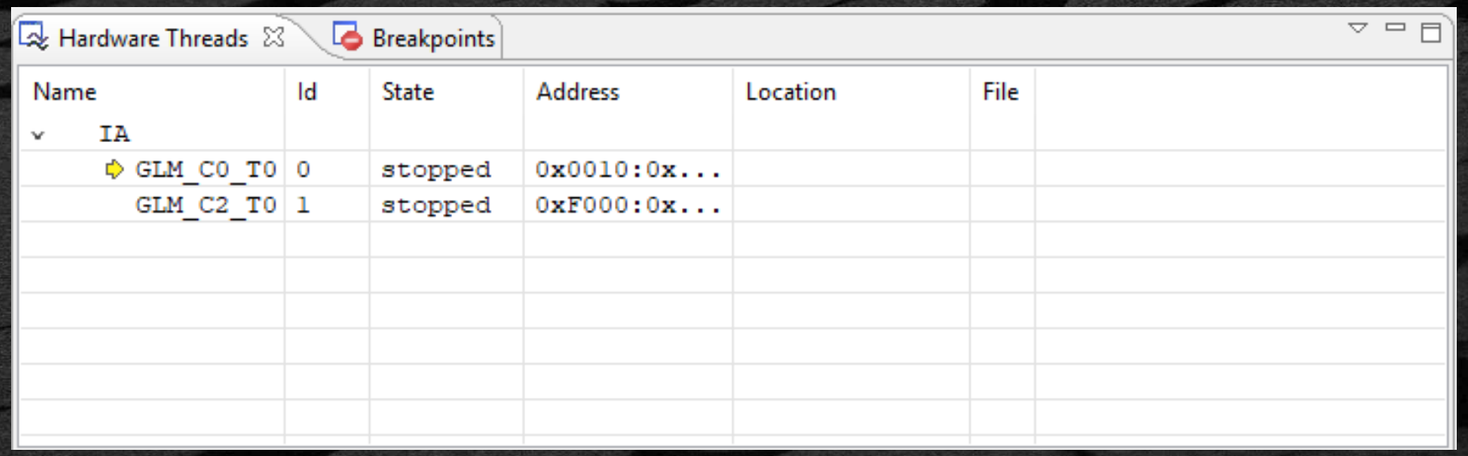


Multiple HW threads

Select View → Hardware Threads

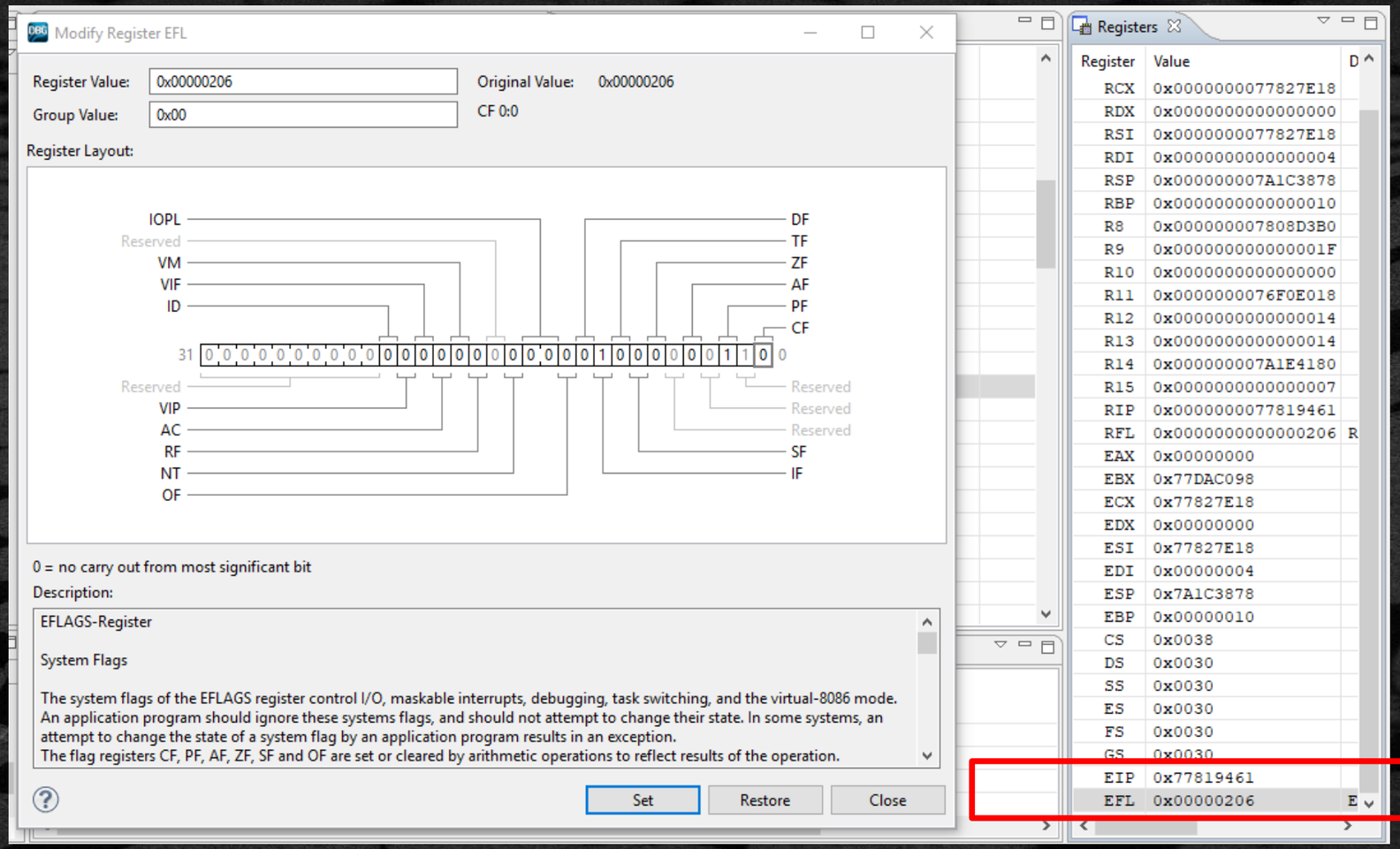
The debugger can display how the multiple logical cores are used and indicate ➡ which logical core is used by the current code displayed.

You can select a logical core and continue debugging the SW running there.



Modify Register Flags

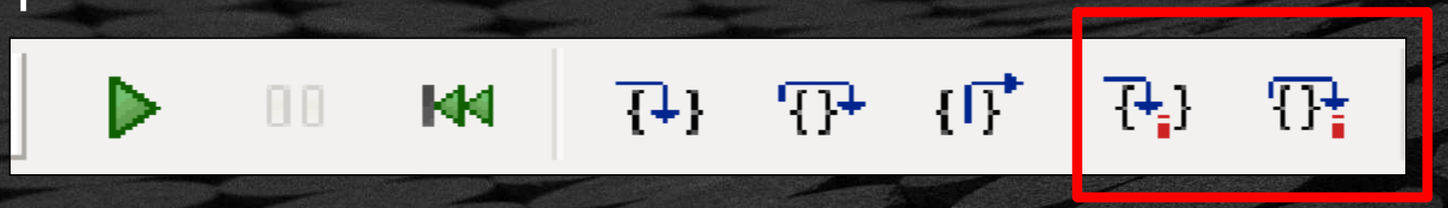
View → Registers Double left click “EFL”



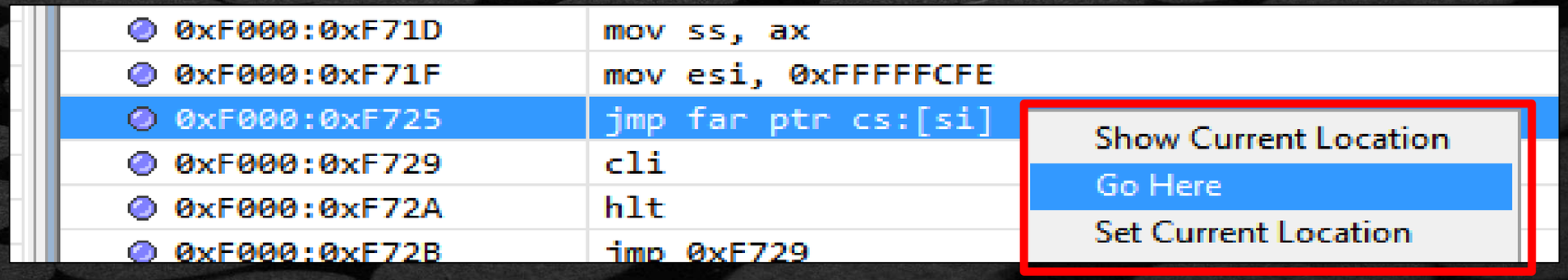
Stepping into UEFI / BIOS code…

Before we can load symbols/source for UEFI we need to be in protected mode

Using the Assembly-level Run Control features you can step from the reset vector to the jump to protected mode:



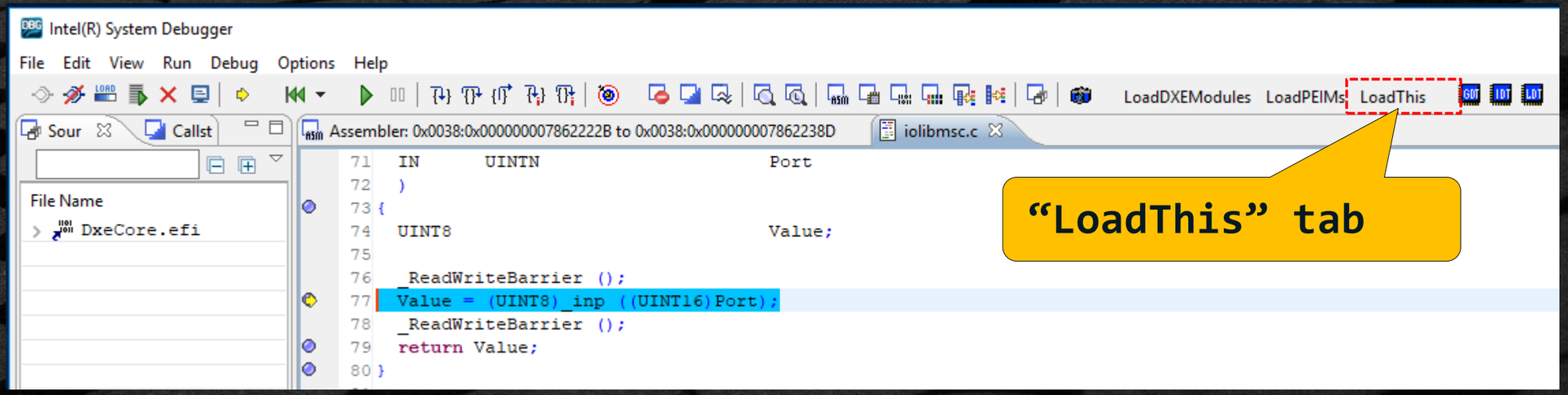
Use the context menu in the ASM window:



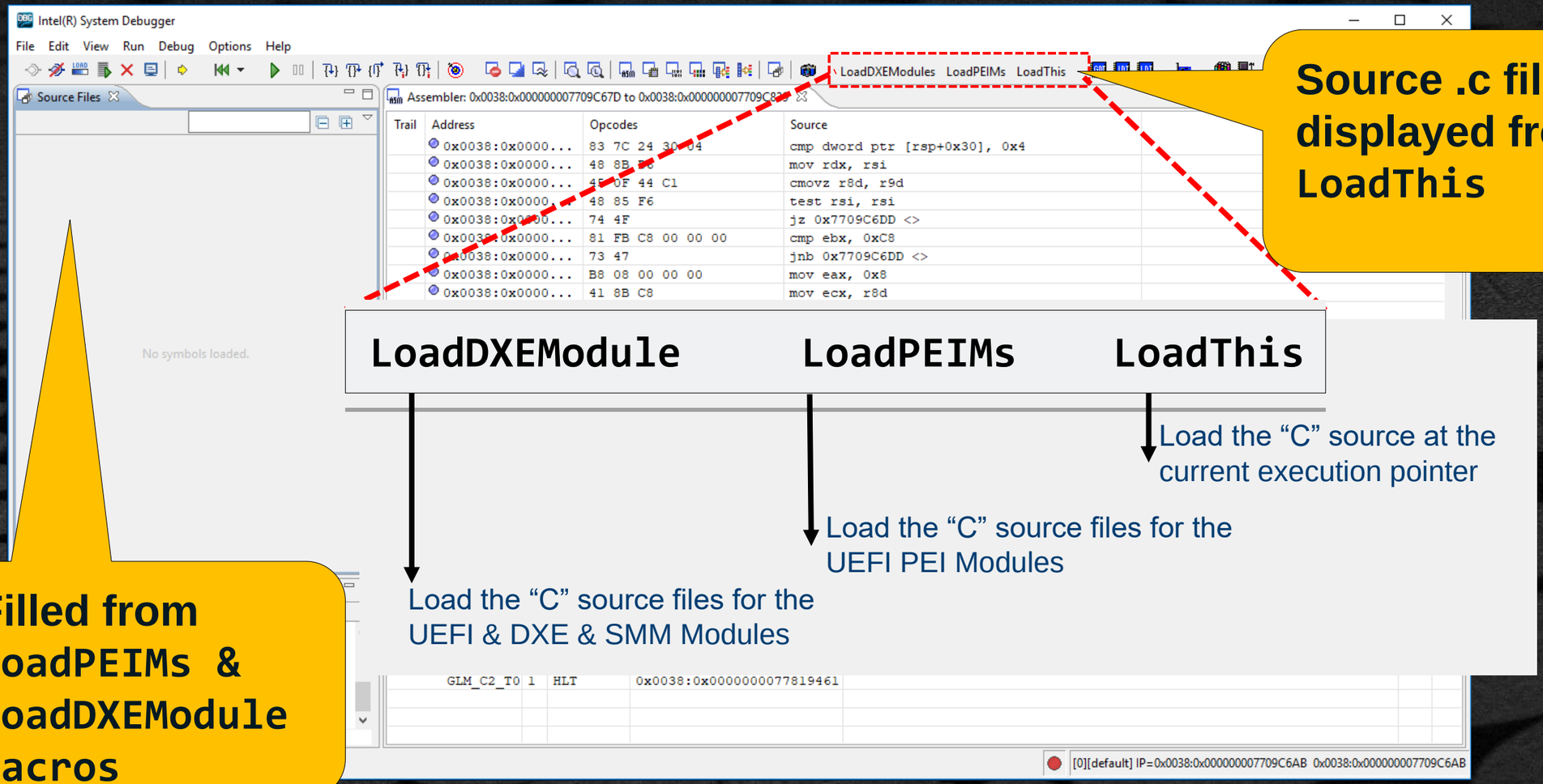
Loading Symbols for UEFI BIOS

Once in protected mode the System Debugger function can load symbols for UEFI using: “loadthis” command or “LoadThis” tab

- Key to UEFI debug: searches memory for a relocatable UEFI module, uses module metadata to locate debug symbols & sources
- Works in all phases of UEFI (SEC, PEI, DXE) (only PEI and DXE with Debug Agent Mode)
- Searches from the current IP or from the a supplied address

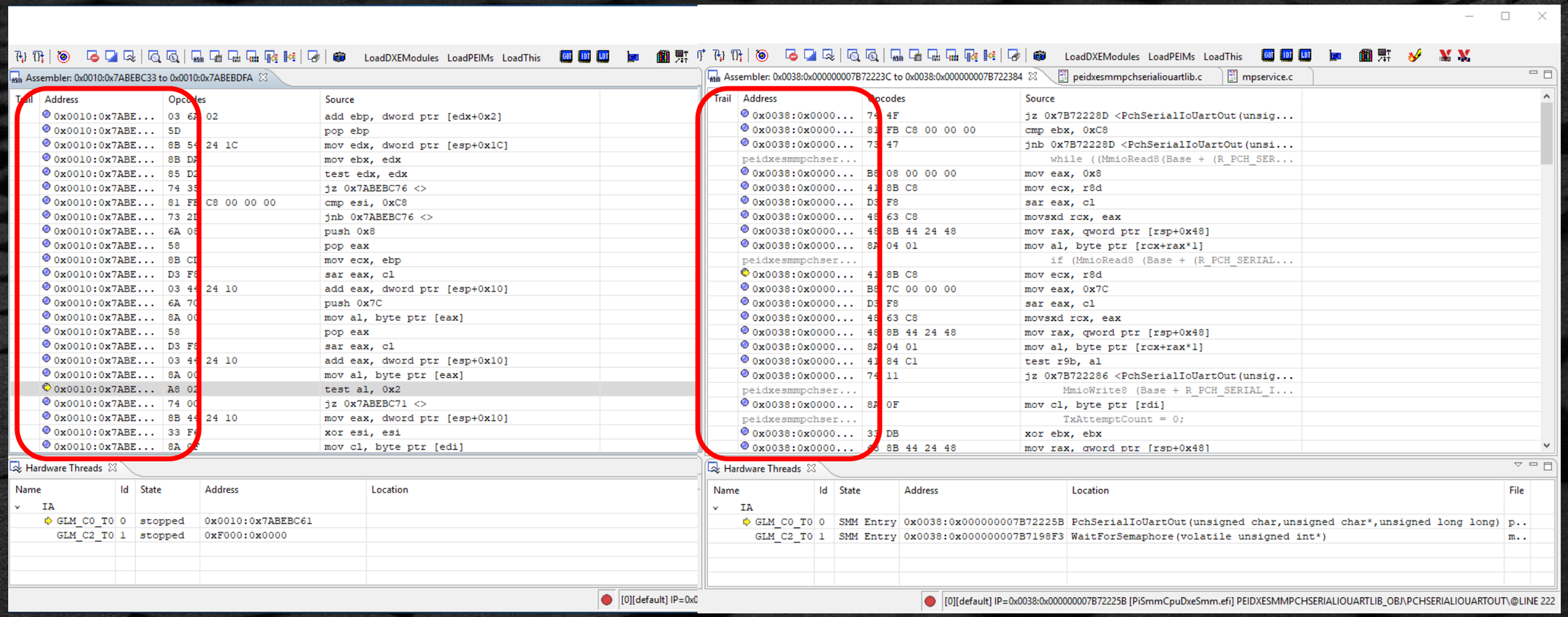


UEFI Viewing Source

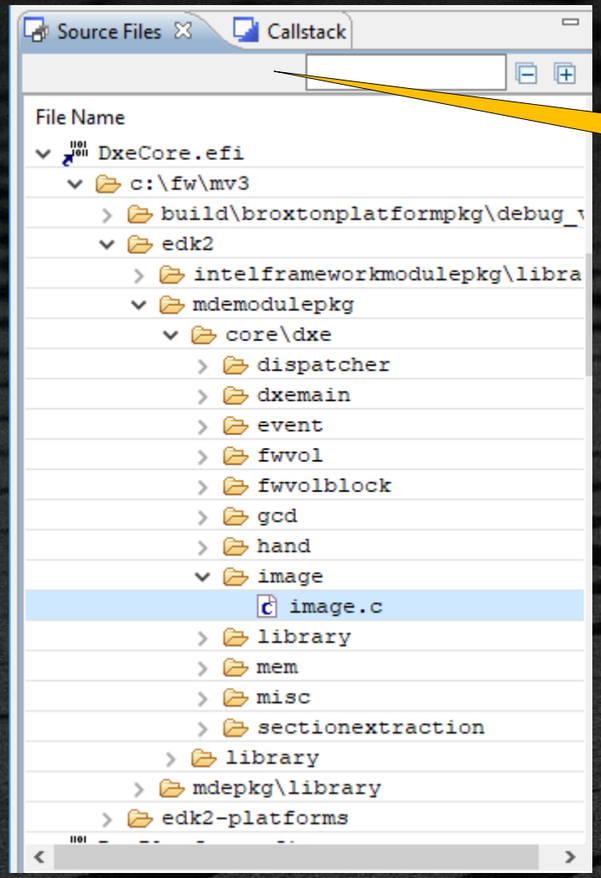


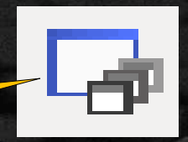
UEFI Viewing Source PEI vs. DXE

PEI segment at 0x0010:0xnnnn DXE segment at 0x0038:0xnnnn

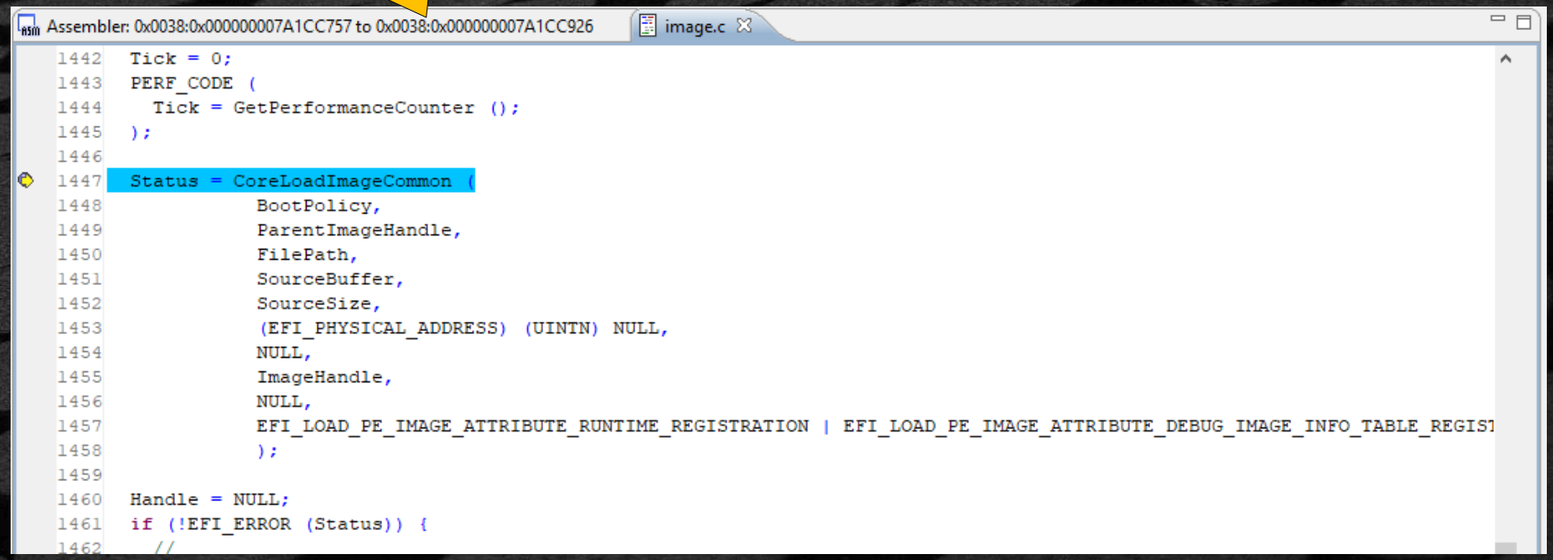


Select source files to view





With the ‘Source Files’ icon you can open a window which contains the source tree as found in the binary file loaded in the debugger.
Just double click on any of the source file names to open the file in the source window



Evaluate symbols

```
469         trace_cpu_idle(1, smp_processor_id());
470         if (this_cpu_has(X86_FEATURE_CLFLUSH_MONITOR))
471             clflush((void *)&current_thread_info()->flags);
472
473         __monitor((void *)&current_thread_info()->flags, 0, 0);
474         smp_mb();
```

&current_thread_info : 0x00000000 = (nil) [(&\\CURRENT_THREAD_INFO)]

EDI
ESP
EBP 0xF
DS 0x0

Hover the cursor over a variable and the debugger will show you its value

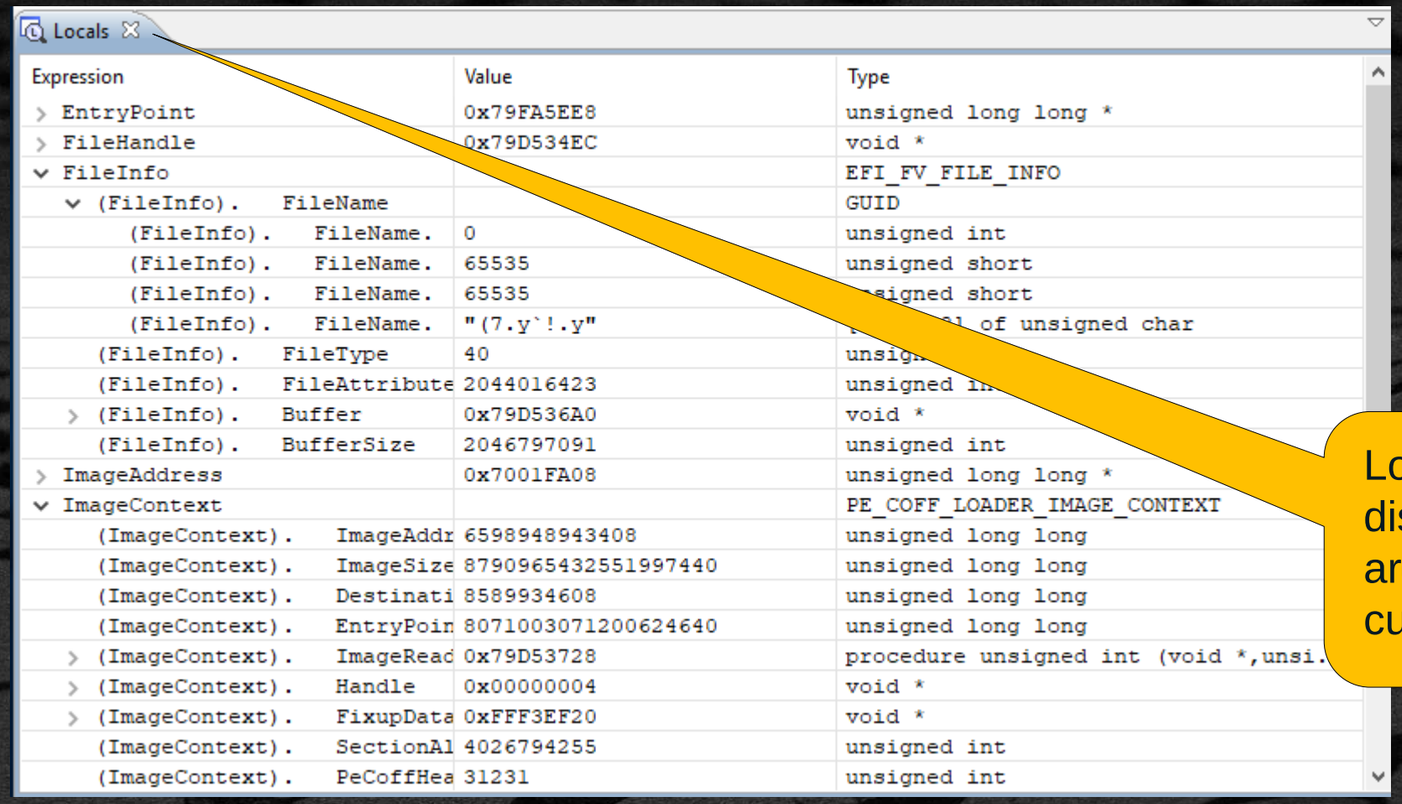
Highlight a variable and use the ‘right mouse click’ – additional options are now available

```
471             clflush((void *)&current_thread_info()->flags);
472
473         __monitor((void *)&current_thread_info()->
474         smp_mb();
475         if (!need_resched())
476             __sti_mwait(0, 0);
477         else
478             local_irq_enable();
479         trace_power_end(smp_processor_id());
480         trace_cpu_idle(PWR_EVENT_EXIT, smp_process
481     } else
482         local_irq_enable();
483 }
484
```

On Selection "current_thread_info ..."
Show Current Location
Run Until Caller
Copy Ctrl+C
Go to Line... Ctrl+G
Input Methods

Create Breakpoint
Locate In Source
Set Value...
Add To Eval Window
Add To Eval From Pointer
Add To Eval From Address
Add To Vector Eval Window
Evaluate
Evaluate From Pointer
Evaluate Address

Local variables



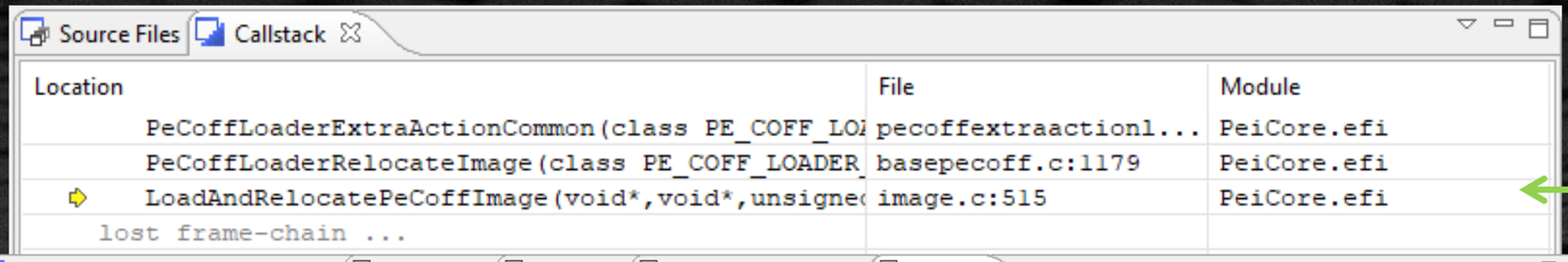
Local variable window will display all variables which are accessible from within current scope

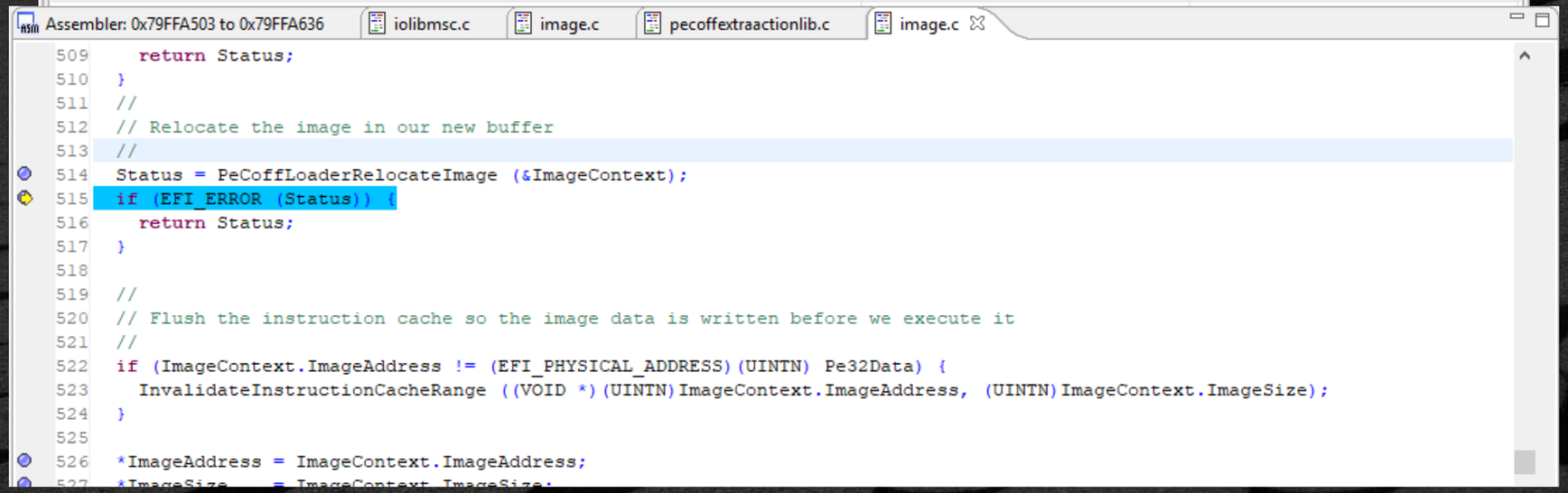
The example above shows part of the ‘FileInfo’ structure

V In expression field indicate that this is a compound element and that it has been Expanded (> for not expanded)

How did I reach the current location?

Callstack



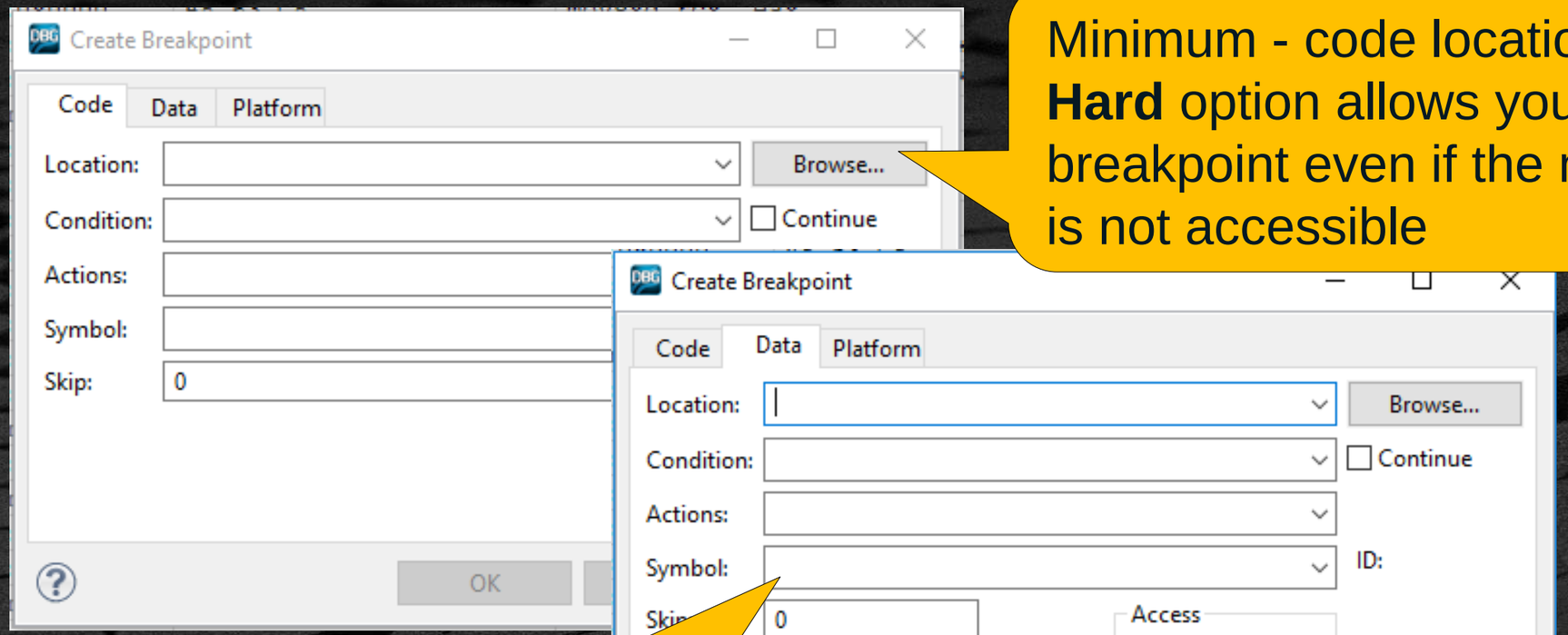


Source file: line number

Current file

Create a breakpoint

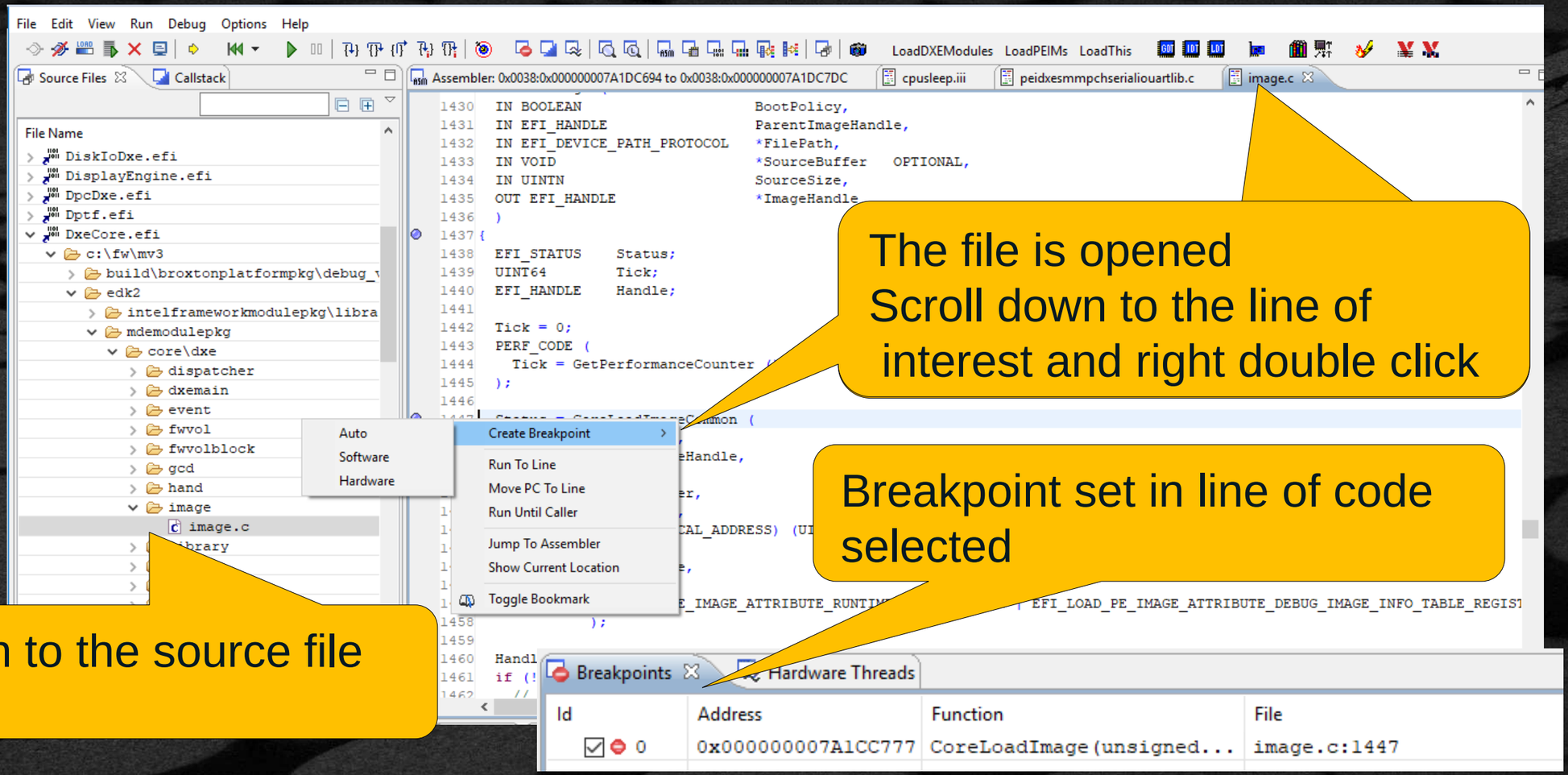
Debug → create Breakpoint



Minimum - code location is required. **Hard** option allows you to set a breakpoint even if the memory location is not accessible

For Watchpoints the location, access type and length are important

Create a breakpoint in Source



The file is opened
Scroll down to the line of interest and right double click

Breakpoint set in line of code selected

Scroll down to the source file of interest

tianocore

What happened? Inspecting System State

Intel System Debugger presents state in a “Human Readable” format

- Relationships: ***“show the nesting of structure members”***
- High-level decode***: “show the names of bits in a register”***
- Context: ***“show the documentation for a register”***

Program State:

- Memory contents -> “Memory Window”
- Program State -> “Locals Window / Eval Window”

CPU State:

- General and Extended Registers
- Page Tables
- IA CPU data structures (GDT, IDT, LDT)

tianocore

Overview of State Windows:

Callstack & HW Threads

Program Variables

Memory & Registers

Page Tables

Linux Kernel State

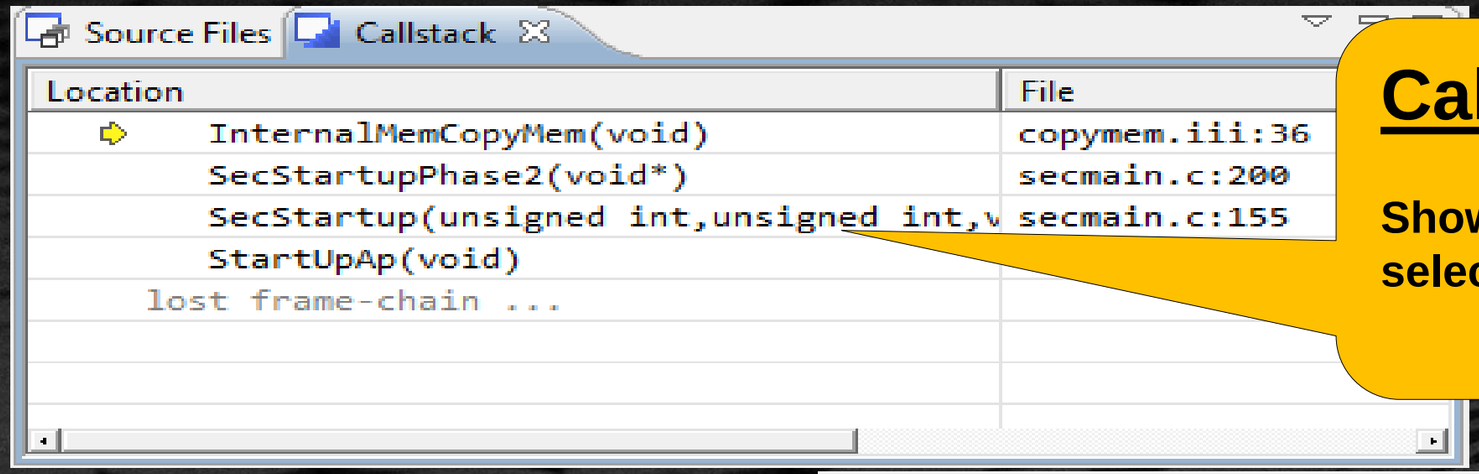
PCI Topology & Devices

Processor Structures

Execution Trace

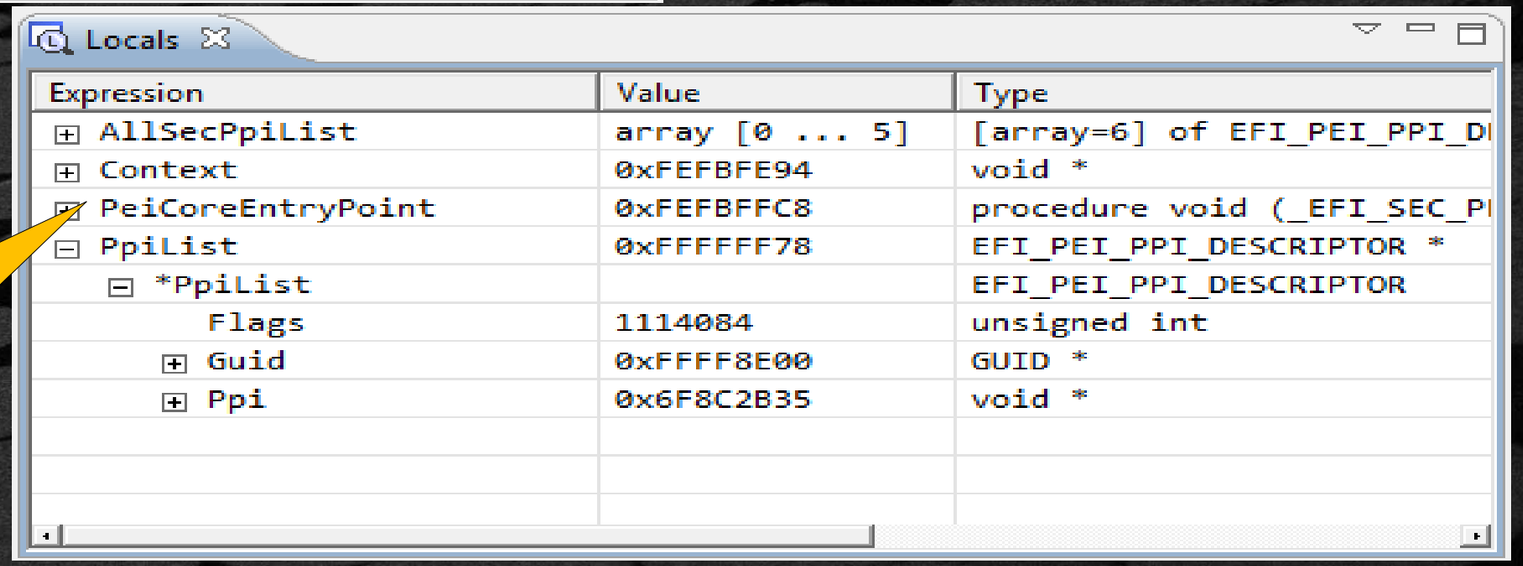
tianocore

Program State:



Callstack Window

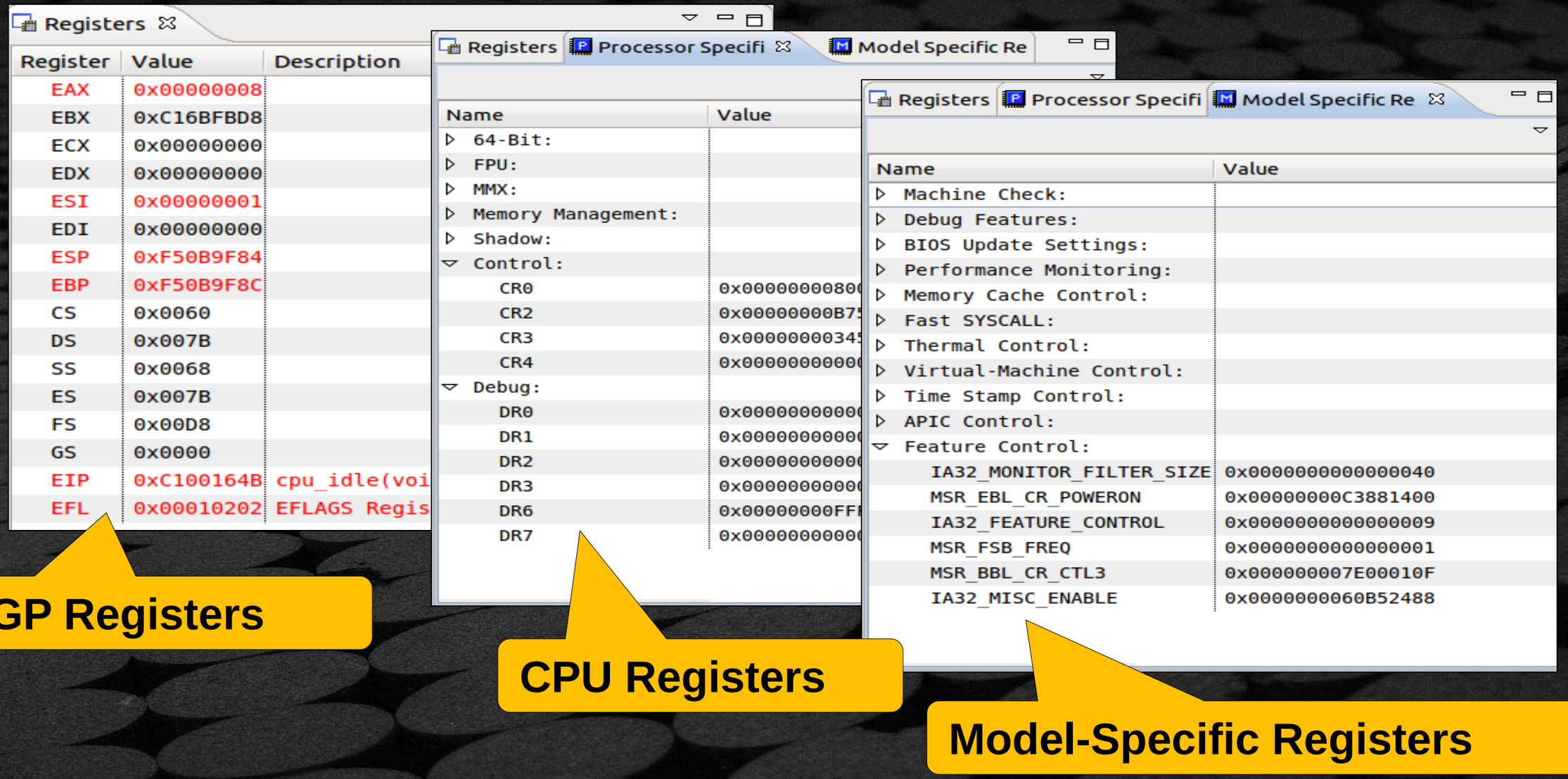
Show call stack for the currently selected hardware thread



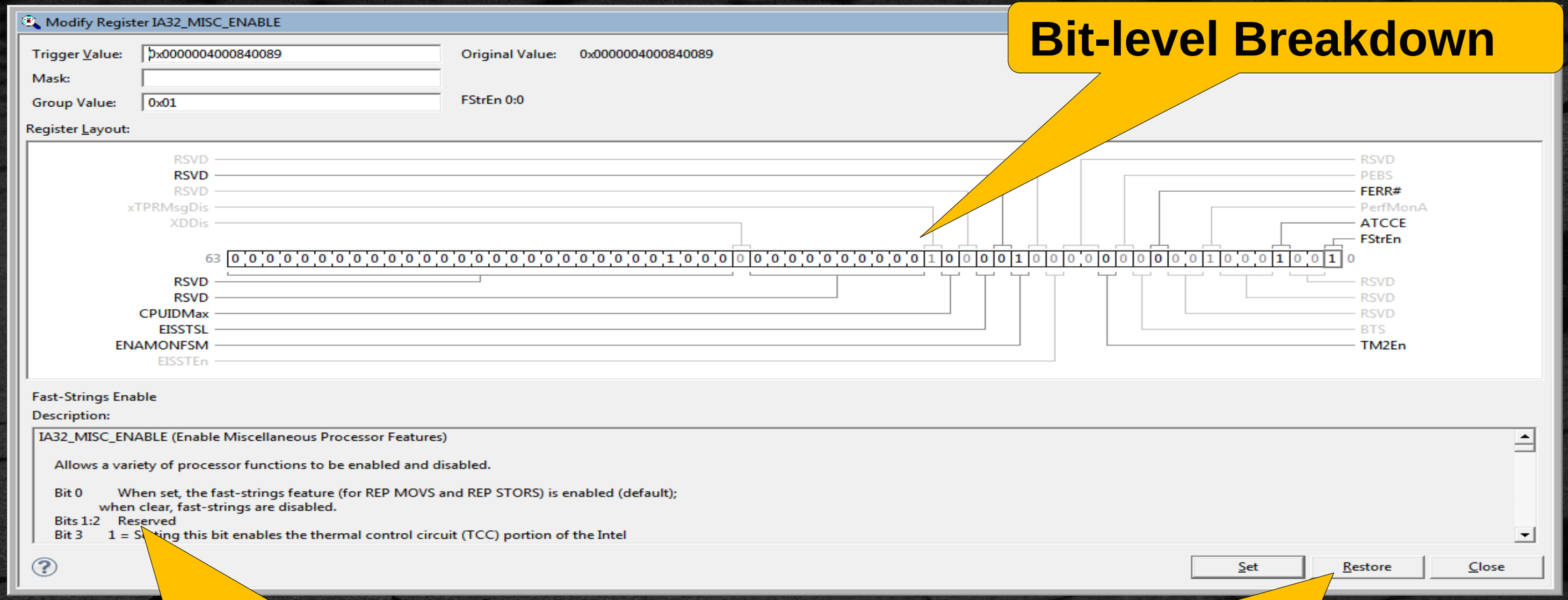
“Locals” Window

Program state for the currently selected callstack frame

CPU State: Register Windows



CPU State: Detailed Register View

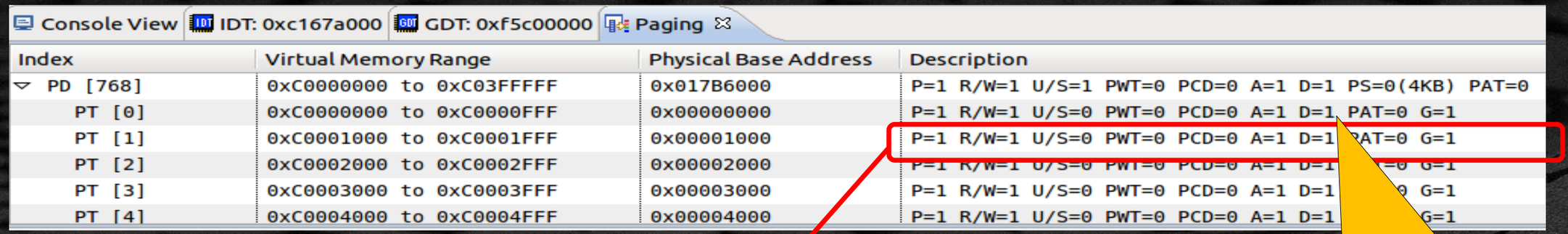


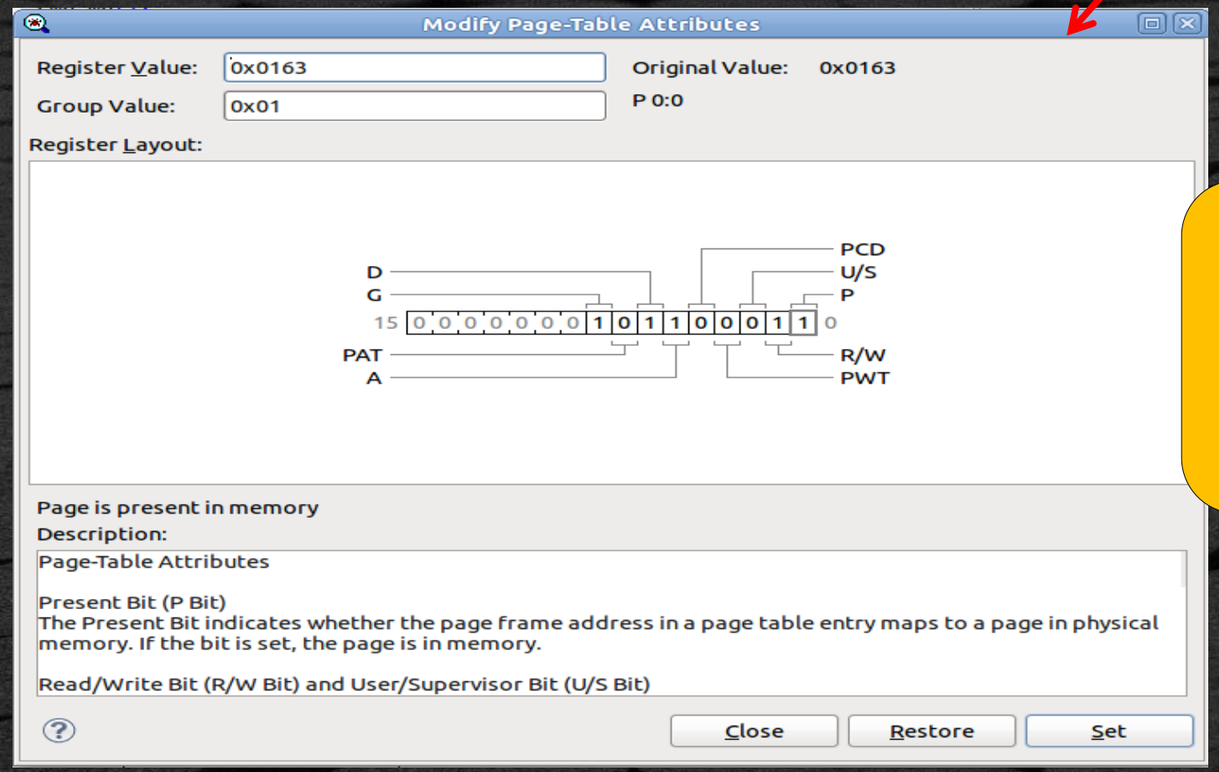
Bit-level Breakdown

Text Documentation

Set/Restore Values

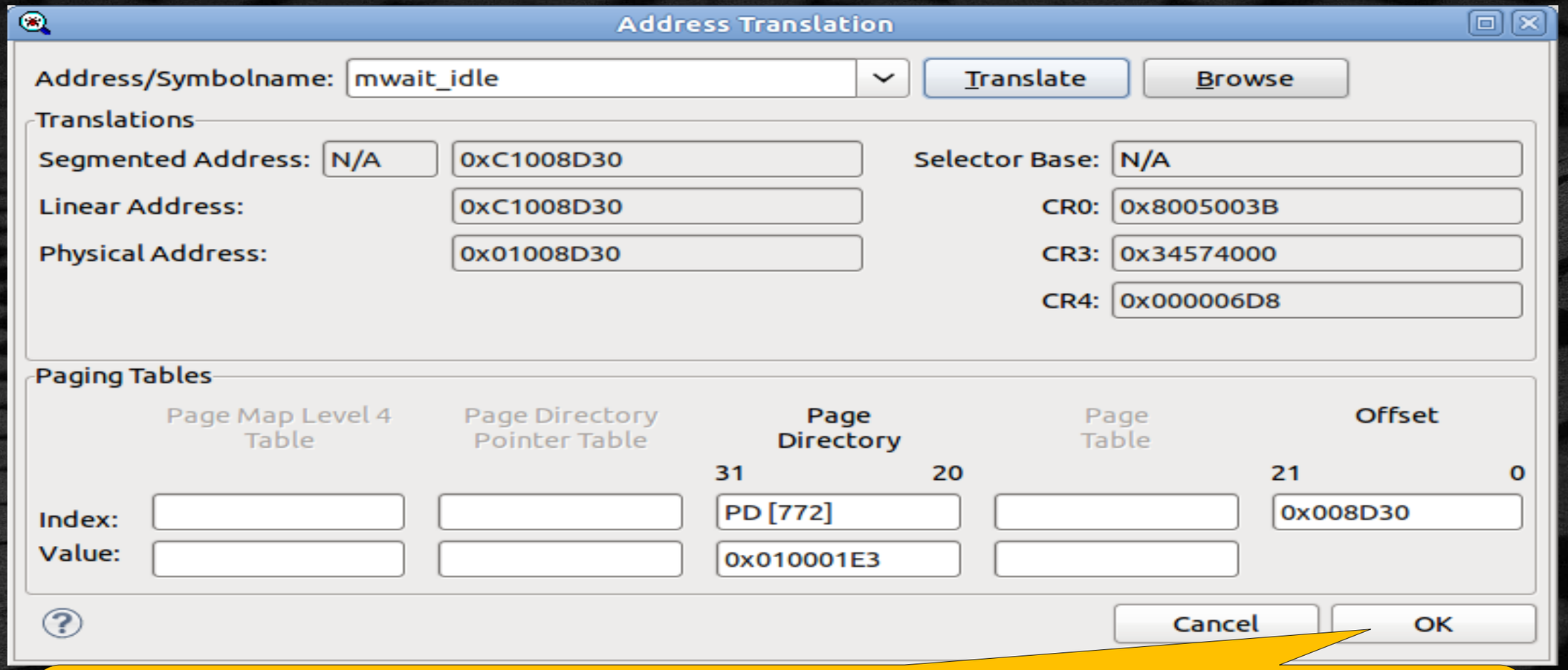
What about paging?





Double click on a page table/directory entry and You will have the ability to modify the attribute bits with the bit field editor

Virtual -> physical address mapping



Select an address and press Translate to find the address mapping. When you press OK the page directory/table involved will be shown in the Paging window

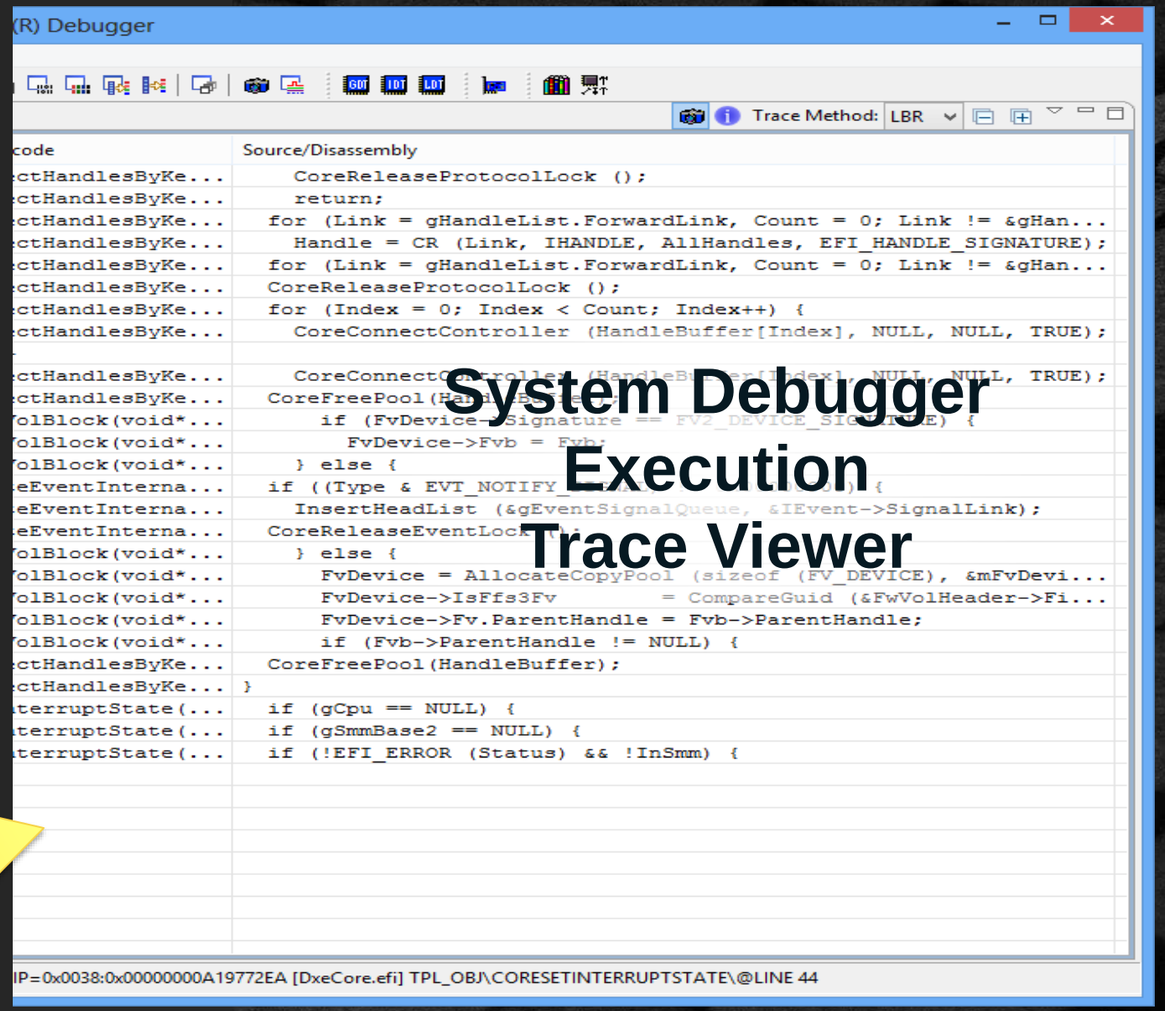
Execution Trace

System Debugger supports execution trace via:

- **I**ntel **P**rocessor **T**race (Intel PT)
- **L**ast-**B**ranch **R**ecord (LBR)

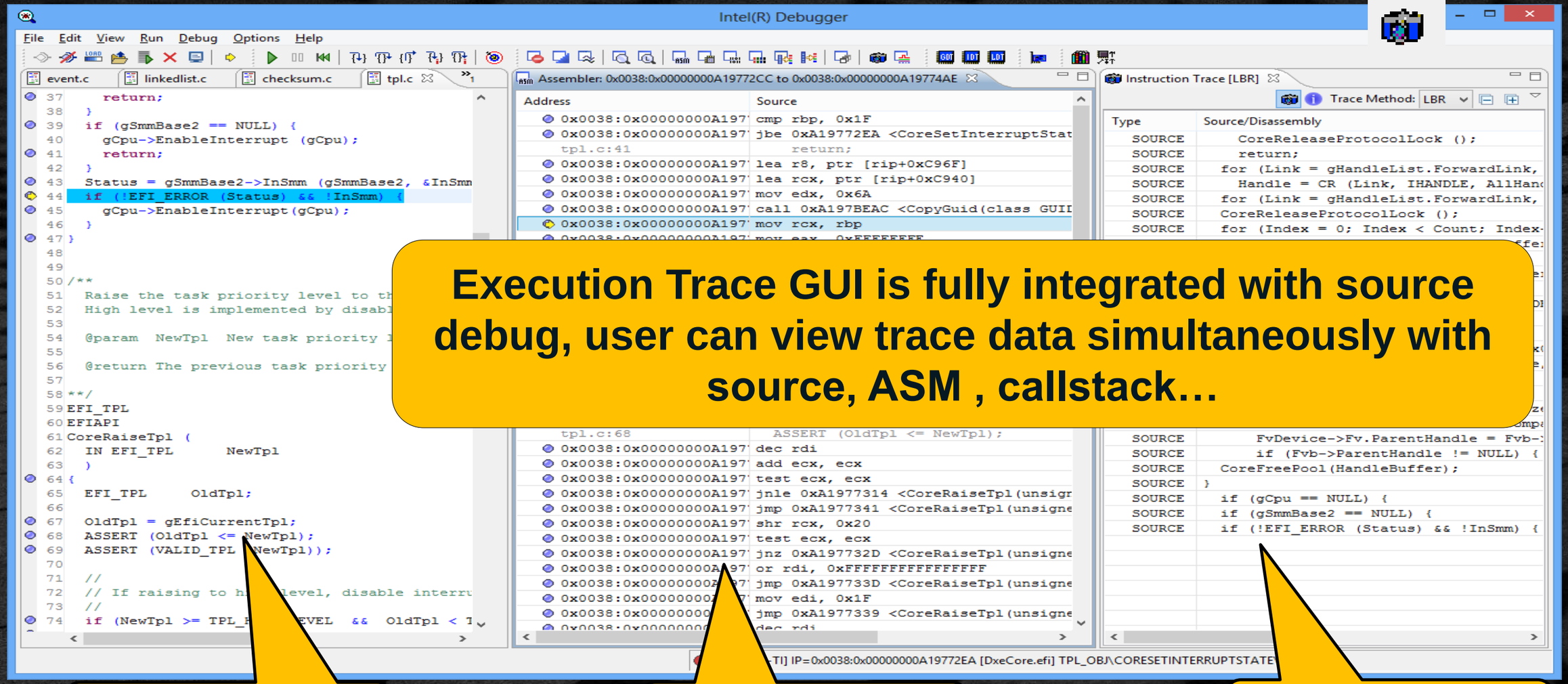
Trace data is presented as C source code, view is integrated with other source debug features

User is not concerned with underlying mechanism, it “just works”

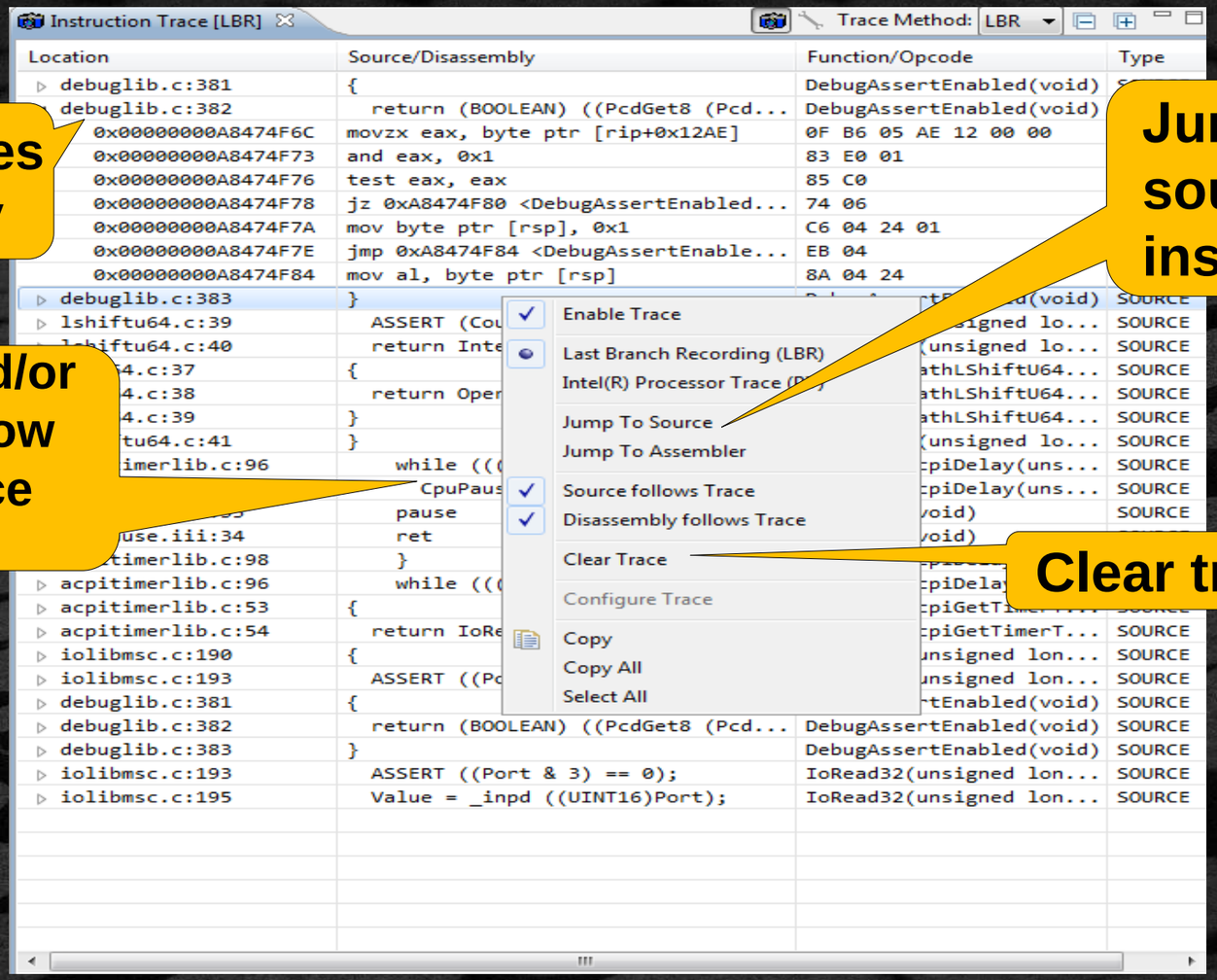


System Debugger Execution Trace Viewer

Execution Trace GUI Overview



Execution Trace Features



Expand source lines to reveal assembly

Jump to this line's source file or instruction

Keep source and/or assembler window in-sync with trace window

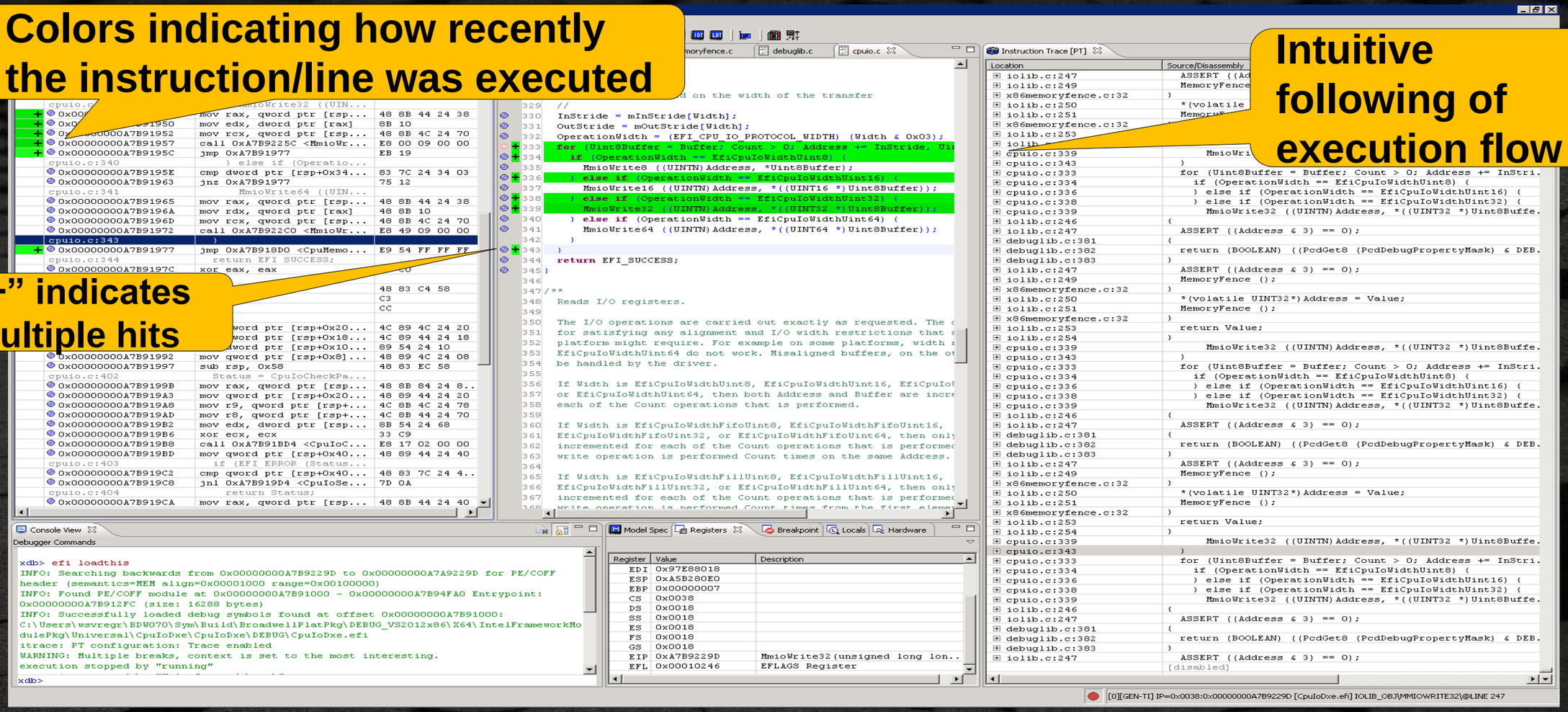
Clear trace buffers

Execution Trace Features (continued)

Colors indicating how recently the instruction/line was executed

“+” indicates multiple hits

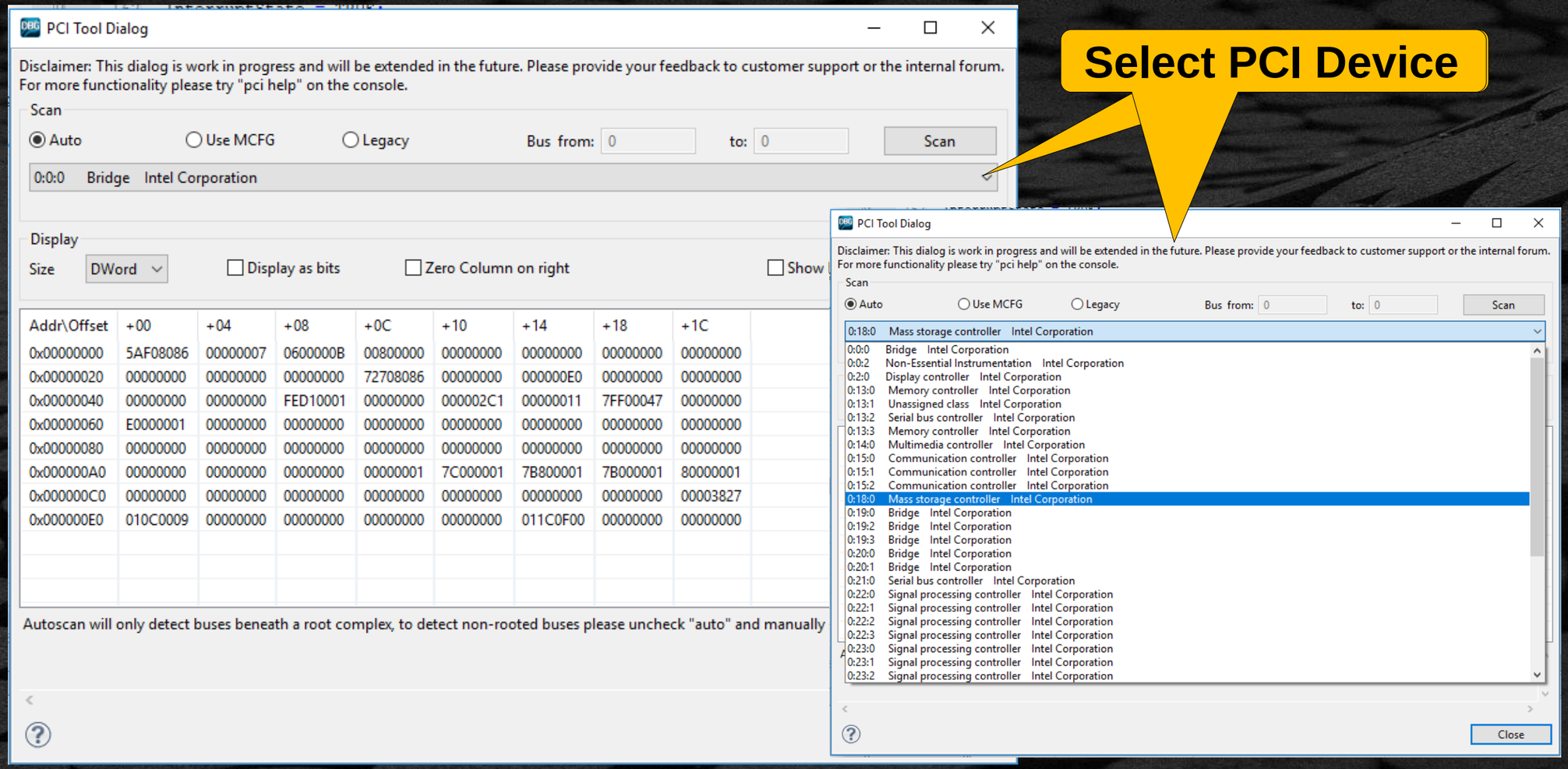
Intuitive following of execution flow



Coloring can be configured in Options -> GUI Preferences

PCI Scan Tool Technology Preview

Scan for devices, display device-specific registers:



[U]EFI Commands

Use “efi help” at the Console command line.

> efi help

EFI Debugger extension
Enter any of the following on the command line:

EFI "SHOWCONFIG"
  Displays the debugger configuration.
EFI "SETSUFFIX %s"
  Set the suffix for EFI debug information files. This setting controls the expected debug info format (Visual Studio/GCC). Accepted values are (efi|debug).
EFI "SHOWSUFFIX"
  Show the active debug information suffix.
EFI "SETSWMODE %s"
  Set the software execution mode of the debugger. Accepted values are (auto|efi32|efi64).
EFI "SHOWSYSTAB [FORCE=%s(OFF|ON)]"
  Print out information parsed from DXE system table. If system table address it not set or the force flag is ON, memory is scanned for it. Command can be used in DXE boot phase.
EFI "SHOWDXEMODULES"
  Print out the list of DXE EFI modules loaded to memory from the DXE system table. Command can be used in DXE boot phase.
EFI "LOADFROMMAP %s"
  Load debug information for modules from the given EFI flash map file.Map file contains modules in flash (SEC and PEI phases).

• • •

Command “efi showsysstab”

> efi showsystab

INFO: Using cached EFI State Information
EFI System table at 0x0000000077FED018

Configuration Tables:

GUID:	Pointer:	Name:
GUID fc1bcdb0, 7d31, 49aa, {...}	0x77dadb98	
GUID ee4e5898, 3914, 4259, {...}	0x77dac718	
GUID 05ad34ba, 6f02, 4214, {...}	0x7a1e1fc0	DXE_SERVICES_TABLE
GUID 7739f24c, 93d7, 11d4, {...}	0x77d8c018	HOB_LIST
GUID 4c19049f, 4137, 4dd3, {...}	0x7a1e2830	
GUID 49152e77, 1ada, 4764, {...}	0x7a1e31e0	EFI_DEBUG_IMAGE_INFO_TABLE
GUID dba6a7e3, bb57, 4be7, {...}	0x77856998	
GUID 4e28ca50, d582, 44ac, {...}	0x79b5c018	

BootServices:

EFI_RAISE_TPL	0x000000007A1D2D4C
EFI_RESTORE_TPL	0x000000007A1D2DE8
EFI_ALLOCATE_PAGES	0x000000007A1CAD54
EFI_FREE_PAGES	0x000000007A1CAE08
EFI_GET_MEMORY_MAP	0x000000007A1CAFB0
EFI_ALLOCATE_POOL	0x000000007A1D14EC
EFI_FREE_POOL	0x000000007A1D19A8
EFI_CREATE_EVENT	0x000000007A1CD884
EFI_SET_TIMER	0x000000007A1D1218
EFI_WAIT_FOR_EVENT	0x000000007A1CDBFC
EFI_SIGNAL_EVENT	0x000000007A1CDAC8
EFI_CLOSE_EVENT	0x000000007A1CDC9C
EFI_CHECK_EVENT	0x000000007A1CDB54
EFI_INSTALL_PROTOCOL_INTERFACE	0x000000007A1C7F88
EFI_REINSTALL_PROTOCOL_INTERFACE	0x000000007A1CD644
EFI_UNINSTALL_PROTOCOL_INTERFACE	0x000000007A1C8450
EFI_HANDLE_PROTOCOL	0x000000007A1C8680
EFI_REGISTER_PROTOCOL_NOTIFY	0x000000007A1CD568
EFI_LOCATE_HANDLE	0x000000007A1D009C
	0x000000007A1D0400
EFI_INSTALL_CONFIGURATION_TABLE	0x000000007A1D3E10
EFI_IMAGE_LOAD	0x000000007A1CC770
EFI_IMAGE_START	0x000000007A1CC80C
EFI_EXIT	0x000000007A1CCAA8
EFI_IMAGE_UNLOAD	0x000000007A1CCBD8
EFI_EXIT_BOOT_SERVICES	0x000000007A1C5158
EFI_GET_NEXT_MONOTONIC_COUNT	0x000000007A1C5118
EFI_STALL	0x000000007A1D3378
EFI_SET_WATCHDOG_TIMER	0x000000007A1CED78
EFI_CONNECT_CONTROLLER	0x000000007A1C8E7C
EFI_DISCONNECT_CONTROLLER	0x000000007A1C9744
EFI_OPEN_PROTOCOL	0x000000007A1C86A4
EFI_CLOSE_PROTOCOL	0x000000007A1C89B4
EFI_OPEN_PROTOCOL_INFORMATION	0x000000007A1C8ADC
EFI_PROTOCOLS_PER_HANDLE	0x000000007A1C8C34
EFI_LOCATE_HANDLE_BUFFER	0x000000007A1D06F4
EFI_LOCATE_PROTOCOL	0x000000007A1D05F4

. . .

UEFI DEBUG AGENT

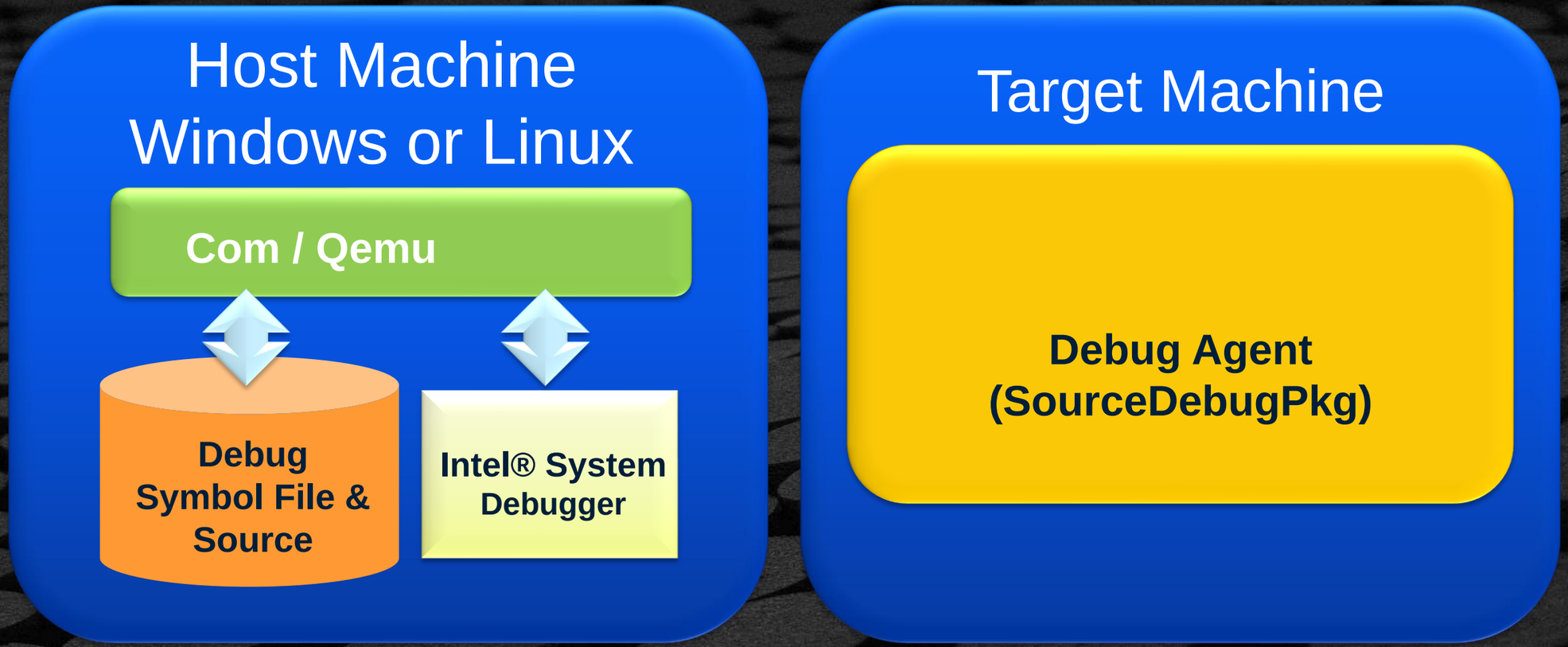
Configure Intel® System Studio Debugger

UEFI BIOS Support – Active Mode

Active Mode (agent-based):

- Receive notifications from agent as modules are loaded/unloaded
- Break at init of a named module, regardless of load position
- Requires debug agent “SourceLevelDebugPkg” in Github EDK II
- Uses Serial Port to connect to Target – no JTAG or DCI

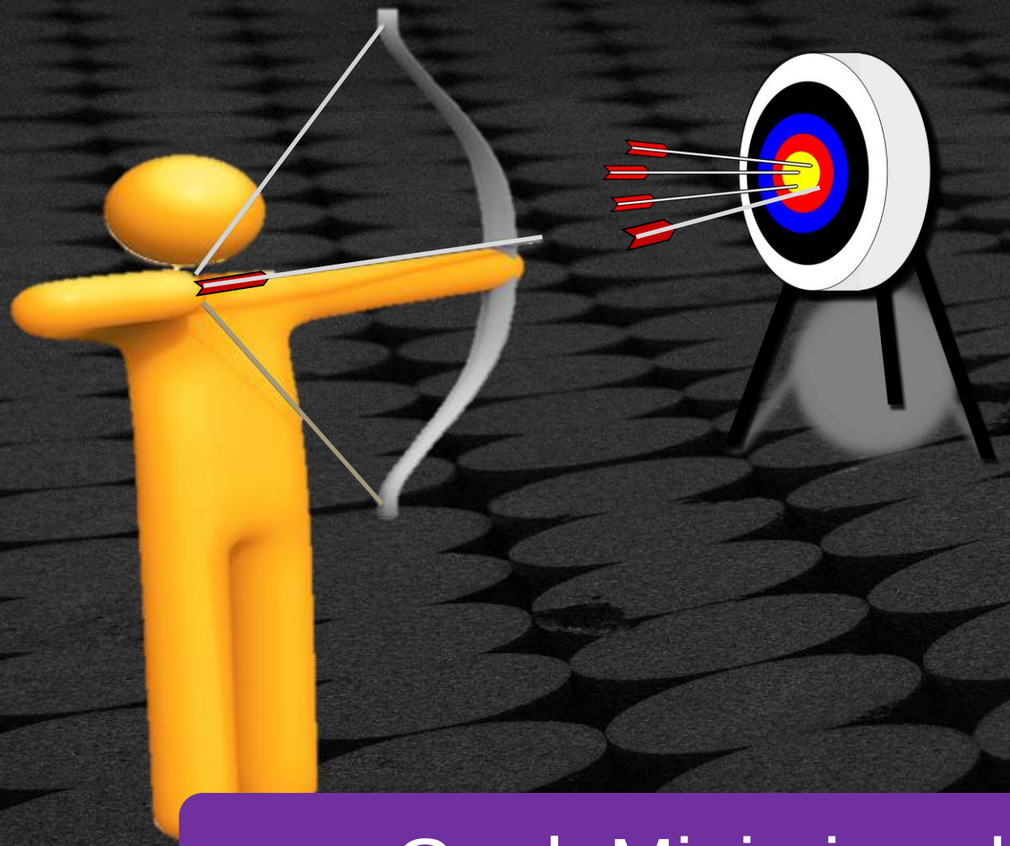
EDK II SourceLevelDebugPkg



Source Level Debugger for UEFI

Changes to Target Firmware

Set “`DEFINE SOURCE_DEBUG_ENABLE = TRUE`” in DSC file



Add call to new library class (**`DebugAgentLib`**) In SEC, DXE Main, and SMM CPU Mod.

Or if you don’t want to add one

A **NULL** implementation of **`DebugAgentLib`** is checked into open source

Goal: Minimize changes to target firmware

Updates to DSC

Libraries

SourceLevelDebugPkg Lib Instance

```
[LibraryClasess]   General
PeCoffExtraActionLib
DebugCommunicationLib
```

- PeCoffExtraActionLibDebug.inf
- **COM1 or USB**
  - DebugCommunicationLibSerialPort.inf
  - or
  - DebugCommunicationLibUsb.inf

```
[LibraryClasses.IA32] SEC / PEI
DebugAgentLib
```

- SecPeiDebugAgentLib.inf

```
[LibraryClasses.X64] DXE
DebugAgentLib
```

- DxeDebugAgentLib.inf

```
[LibraryClasses.X64.DXE_SMM_DRIVER] SMM
DebugAgentLib
```

- SmmDebugAgentLib.inf

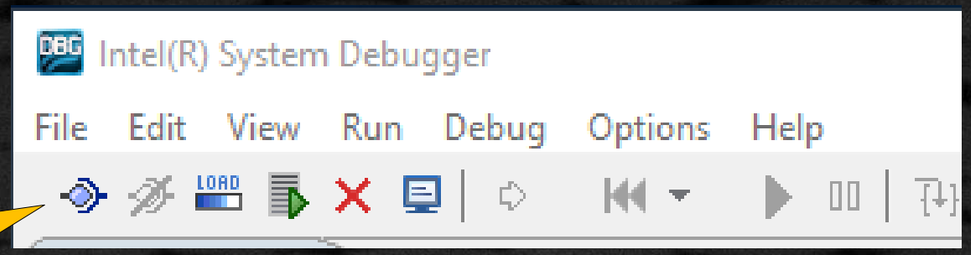
Updates to DSC for USB 3.0

Libraries

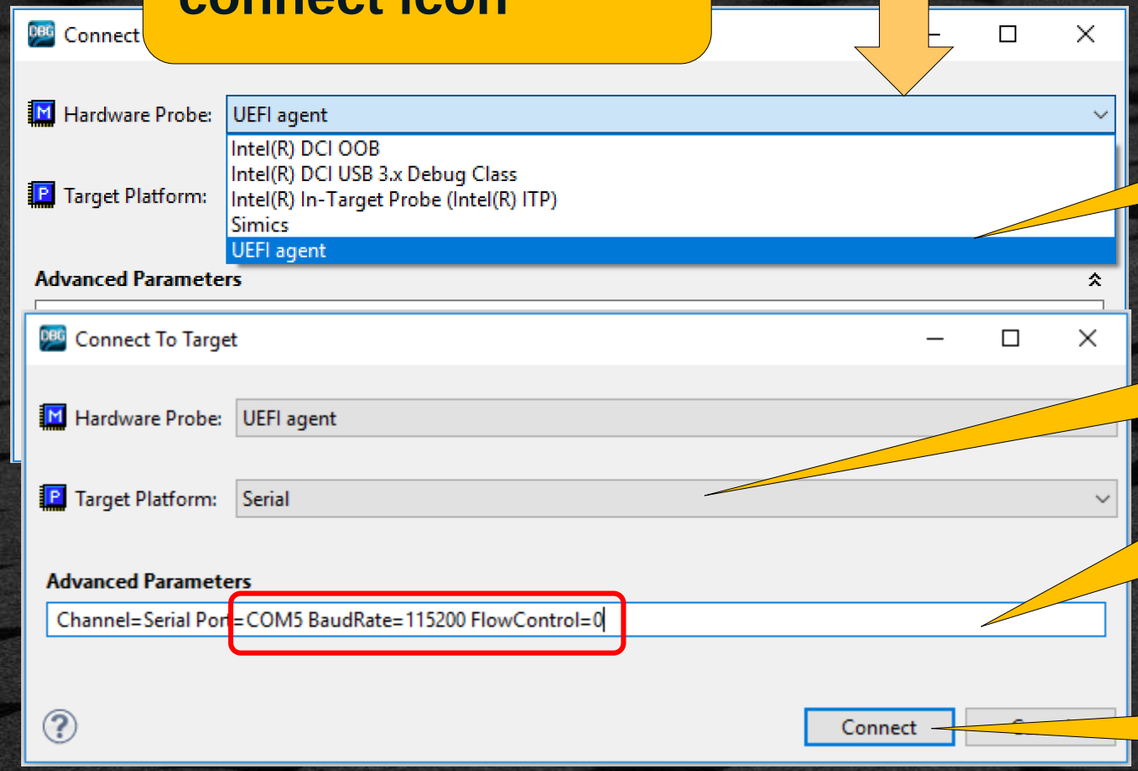
SourceLevelDebugPkg Lib Instance

[LibraryClasess] **General**	
PeCoffExtraActionLib	PeCoffExtraActionLibDebug.inf
[LibraryClasses.IA32] **SEC/PEI**	
DebugCommunicationLib	DebugCommunicationLibUsb3Pei.inf
DebugAgentLib	SecPeiDebugAgentLib.inf
[LibraryClasses.X64] **DXE**	
DebugCommunicationLib	DebugCommunicationLibUsb3Dxe.inf
DebugAgentLib	DxeDebugAgentLib.inf
[LibraryClasses.X64.DXE_SMM_DRIVER] **SMM**	
DebugCommunicationLib	DebugCommunicationLibUsb3Dxe.inf
DebugAgentLib	SmmDebugAgentLib.inf

Connecting To Targets



Step 1: Click the connect icon



Step 2: Select “UEFI agent”

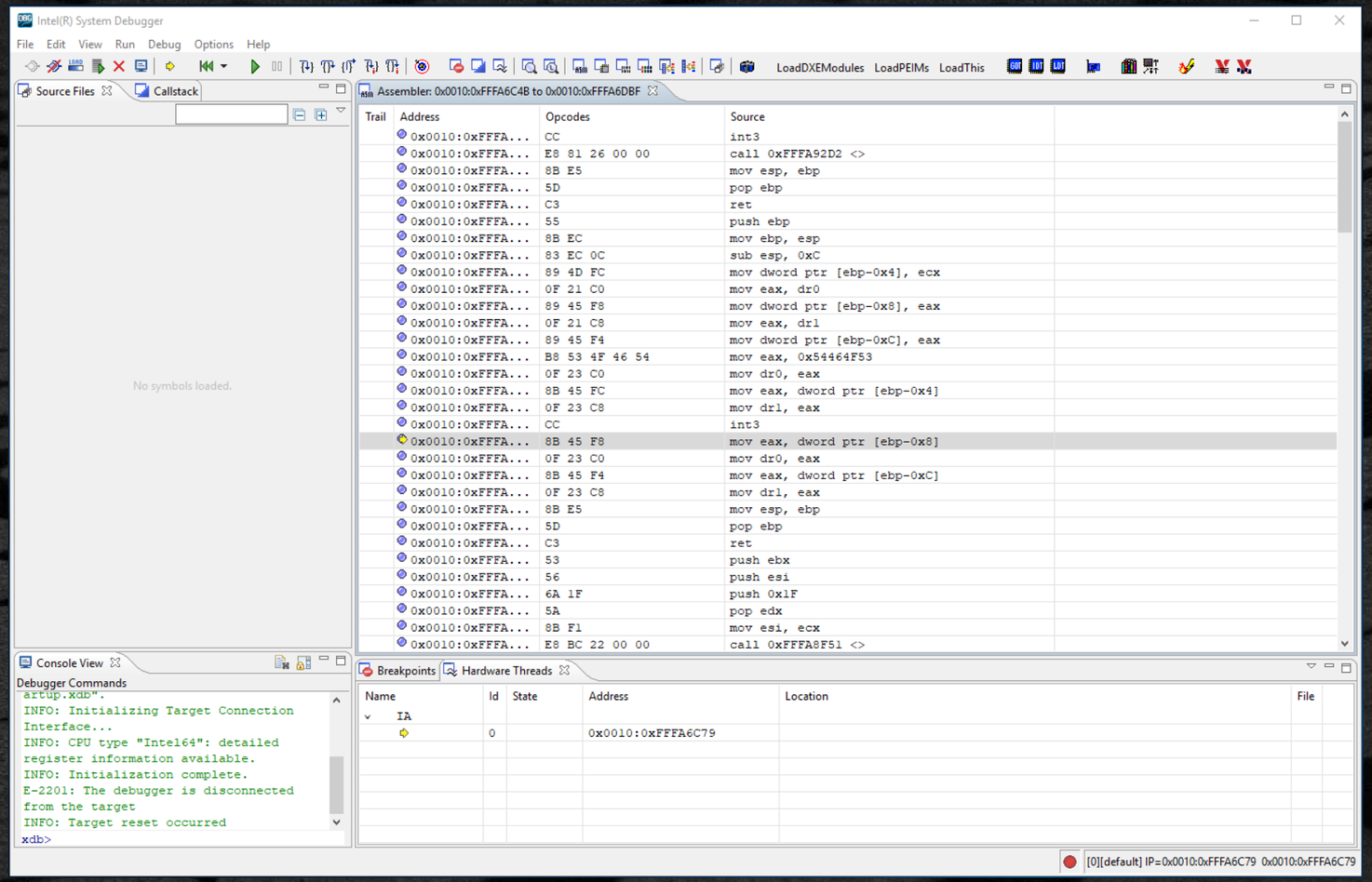
Step 3: Select “Serial” for target platform

Step 4: Enter the COM number and configuration

Step 5: Connect!

Power on Target

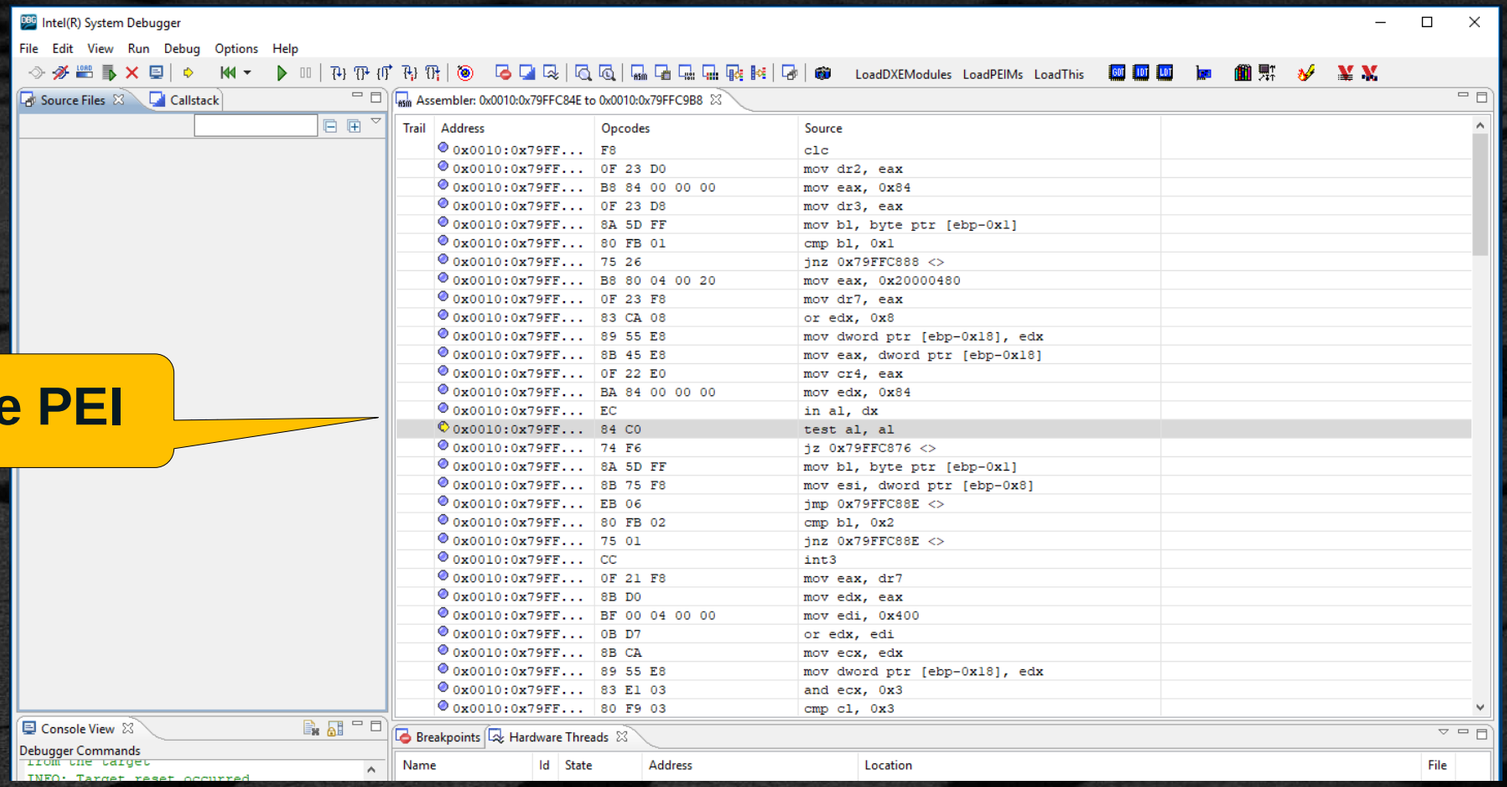
At Power on the target is in the Debug Agent code



tianocore

Debugging UEFI

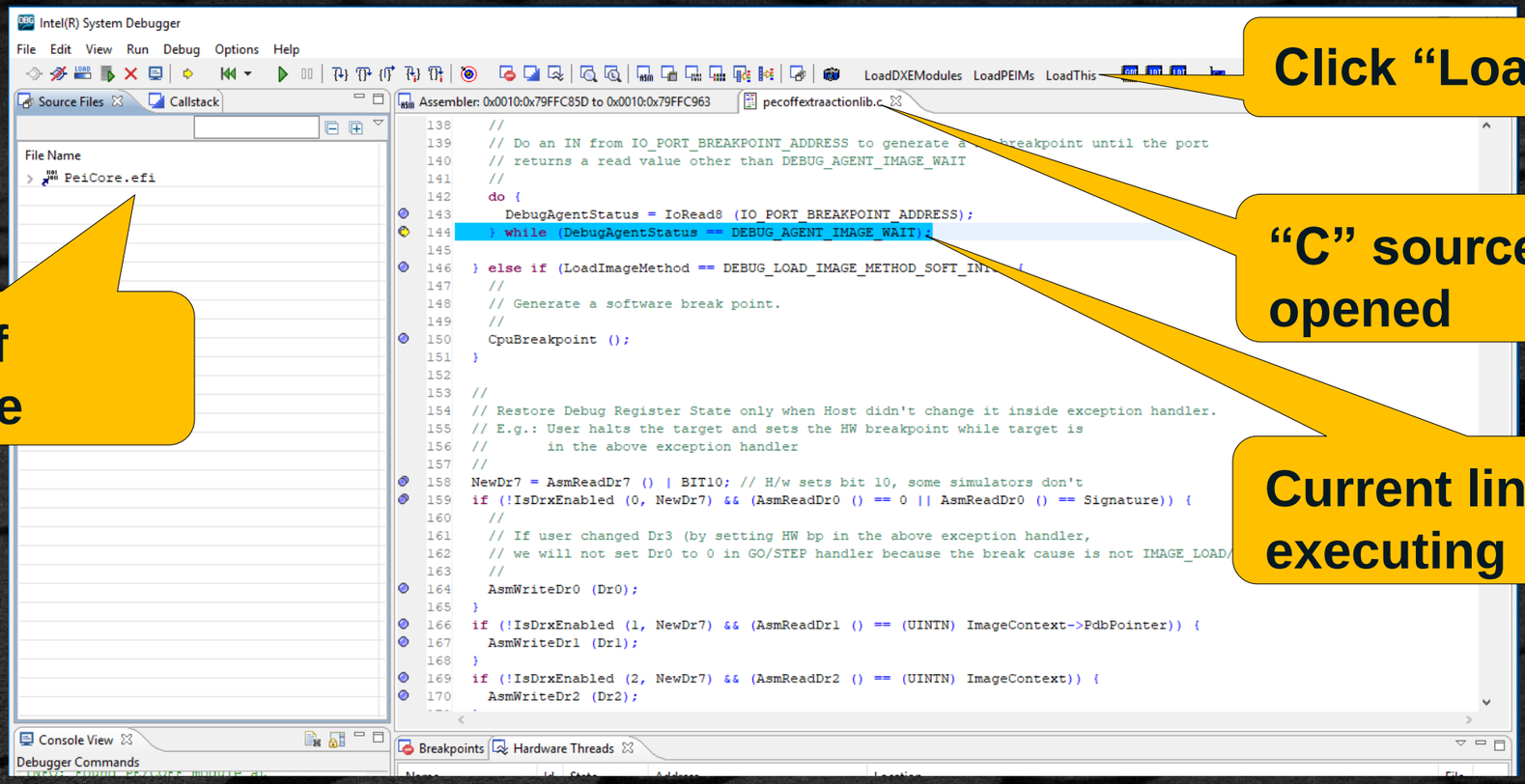
Select run “F5” ▶ then halt “pause” ❚❚ quickly



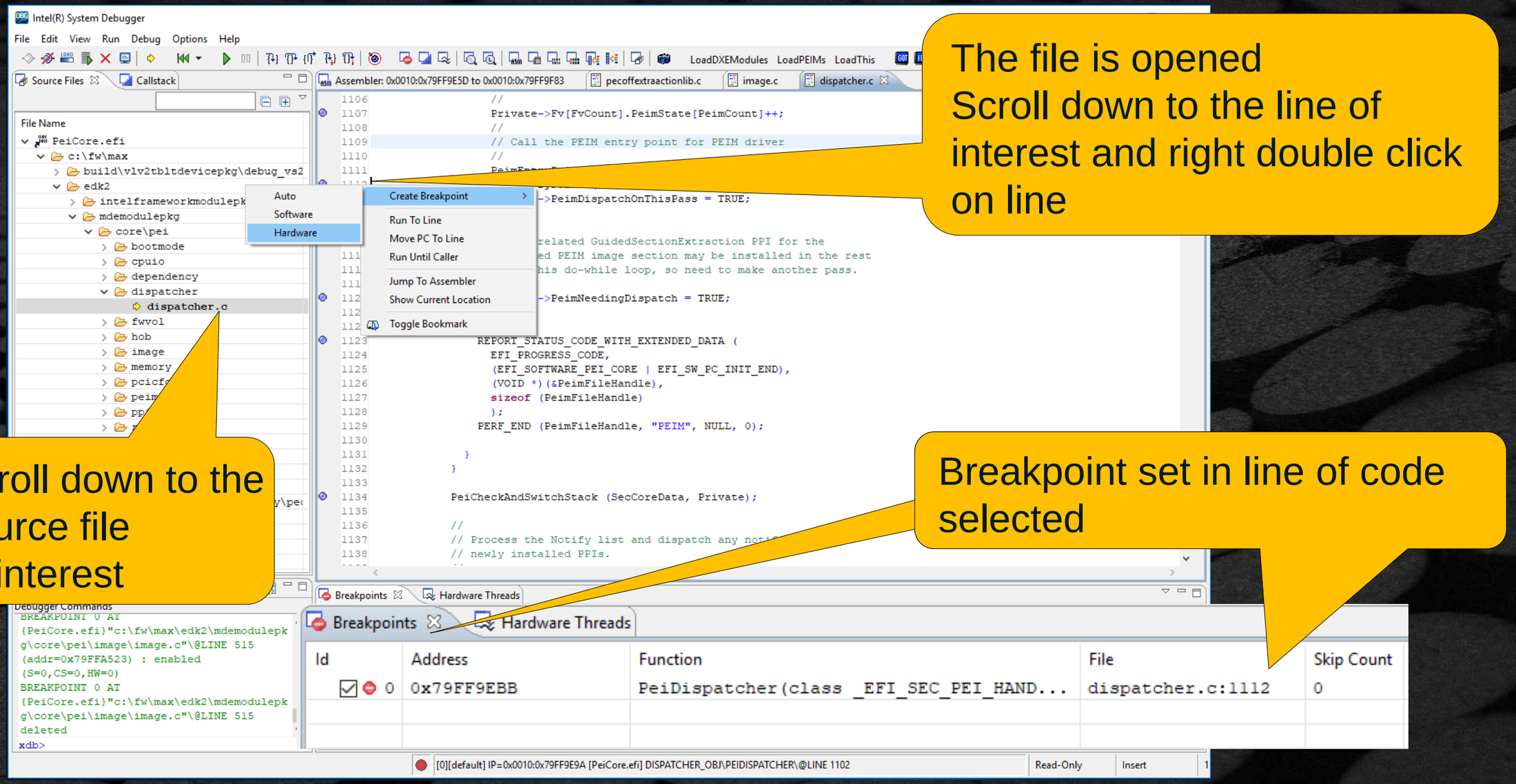
Halt inside PEI

Debugging UEFI

Use System Debugger function “LoadThis” to load symbols



Breakpoint in PEI



The file is opened
Scroll down to the line of interest and right double click on line

Scroll down to the source file of interest

Breakpoint set in line of code selected

BREAKPOINTS

Add Breakpoint to the Compiled BIOS / Firmware Source Code

EDK II Breakpoints w/ System Debugger

`CpuBreakpoint`

When using a UEFI agent connection:

- Debug agent – `SourceLevelDebugPkg`

`CpuDeadLoop`

When using a Hardware debugger:

- In-Target Probe (ITP)
- Intel® SVT DCI cable
- Intel® SVT Closed Chassis Adapter (CCA)
- other 3rd Party Hardware (i.e. Lauterbach w/ JTAG)

The functions `CpuBreakpoint()` and `CpuDeadLoop()` are part of the EDK II Base Libraries and can be compiled with any UEFI or PI Module at any phase of the boot flow (SEC, PEI, DXE, BDS, TSL)

Special DCI Breakpoint

`CpuIceBreakpoint`

The Intel Architecture has a special op-code for a breakpoint: `int1`
Better than a `CpuDeadLoop()` since it halts the processor. Better trace information
Downside, there is no “C” equivalent – needs to be assembly code

CpuIceBreakpoint.nasm ⇦ Needs to be in /Ia32 & /X64 directory

```
DEFAULT REL
SECTION .text
global ASM_PFX(CpuIceBreakpoint)
ASM_PFX(CpuIceBreakpoint):
   int1
   ret
```

What is a ICEBP

Intel® Processor PMCR	• Interrupt Redirection is enabled in the Probe Mode Control Register PMCR
An undocumented op code	• not software accessible • ICE must be connected to modify PMCR
Makes debugging easier	• to figure out where your program was loaded in memory since it halts the processor

Compiling for CpuIceBreakpoint

“C” File

```
extern VOID  CpuIceBreakpoint();

VOID
EFIAPI
CpuIceBreakpoint();

. . .

UefiMain(
{
. . .

CpuIceBreakpoint();

. . .
}
```

INF File

```
. . .

[Sources]
  SampleApp.c
[Sources.Ia32]
  Ia32/CpuIceBreakpoint.nasm
[Sources.X64]
  X64/CpuIceBreakpoint.nasm
```

See example: link

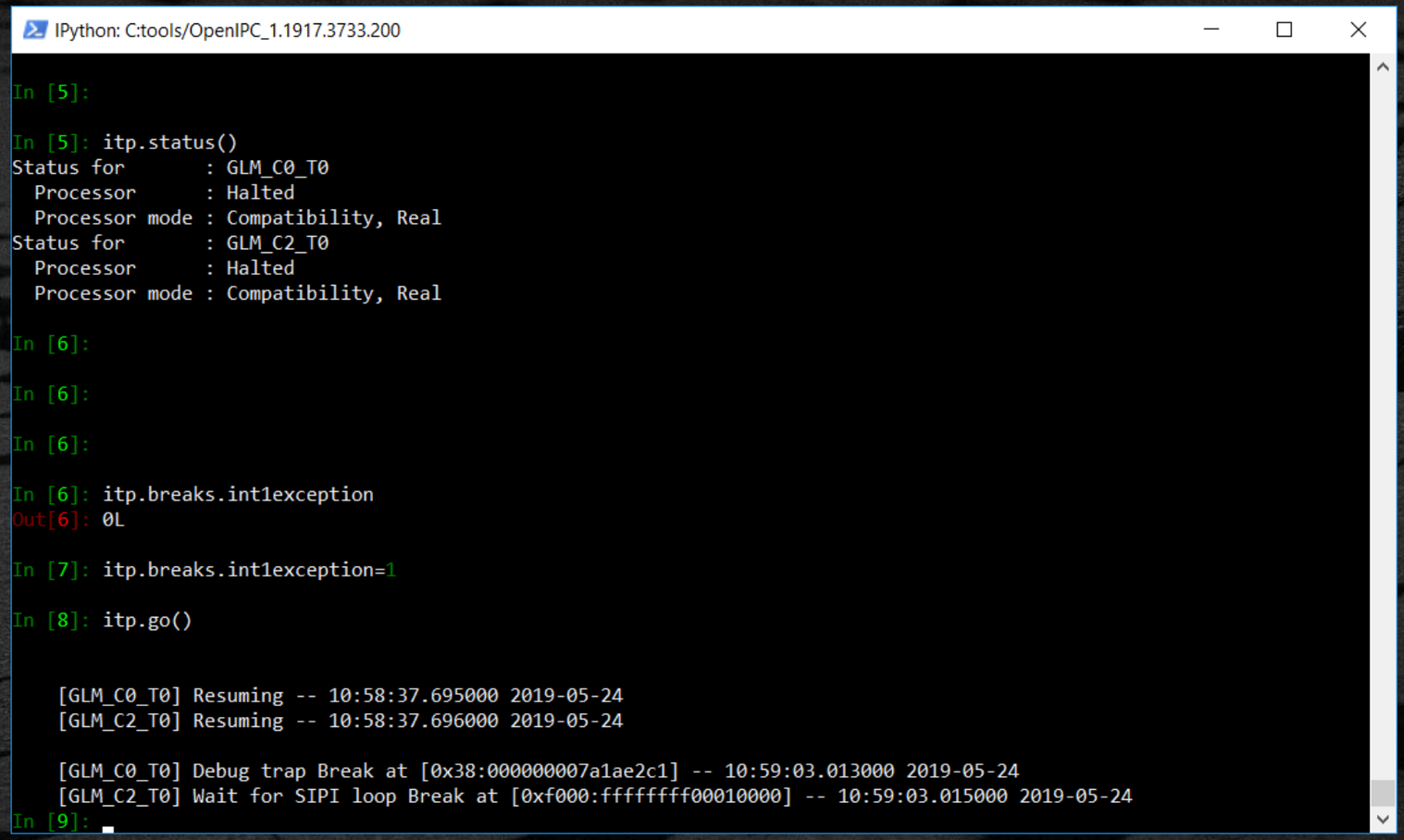
Enable INT1 Exception handler

How to enable INT1 Exception handling in Intel System Debugger to trap the CPU and get the control for source debugging. Target Communication Framework (TCF)

1. On "Ipccli"
2. In System Debugger TCF debugger's command line console

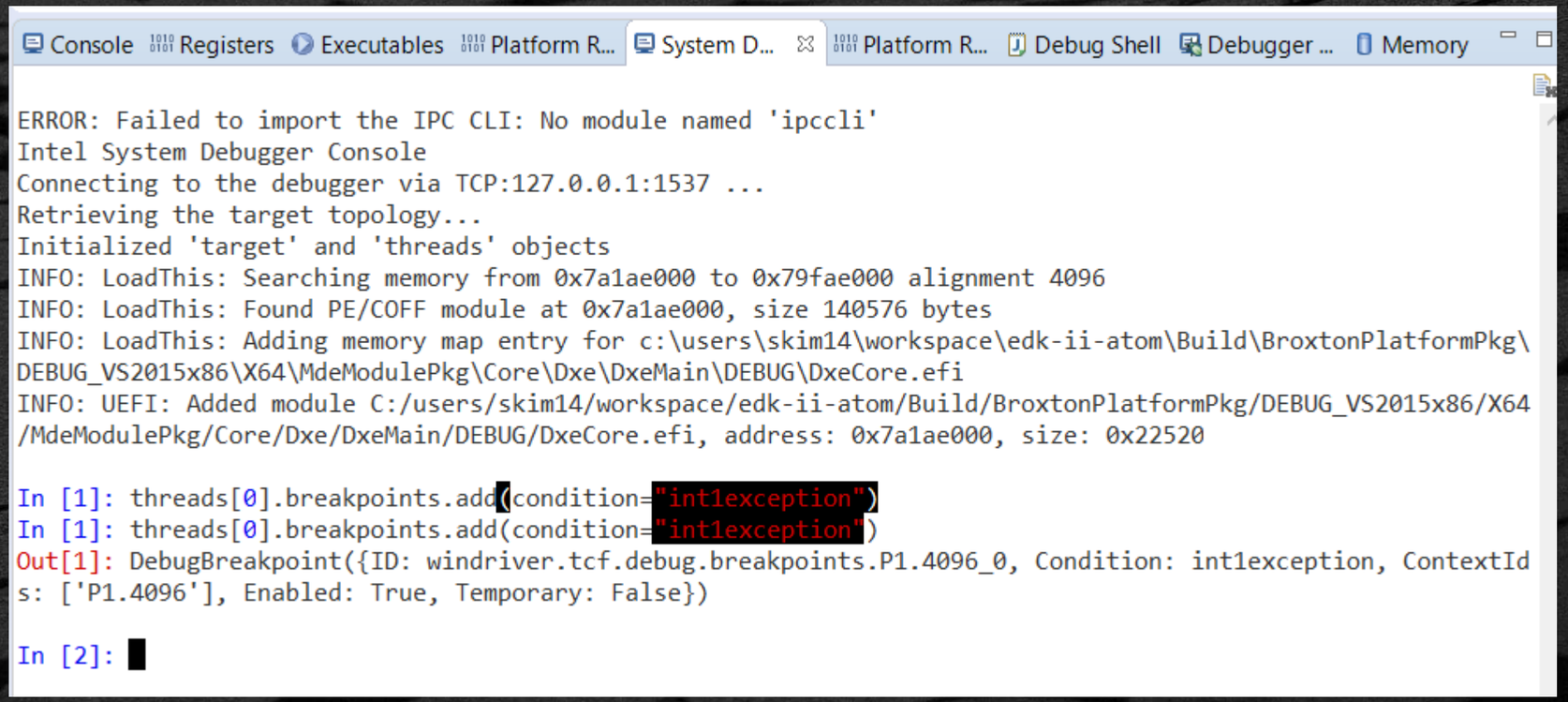
Enable INT1 Exception handler- “Ipccli”

On "ipccli", after halt the process then issue "itp.breaks.int1exception=1" which enable int1 trap now.



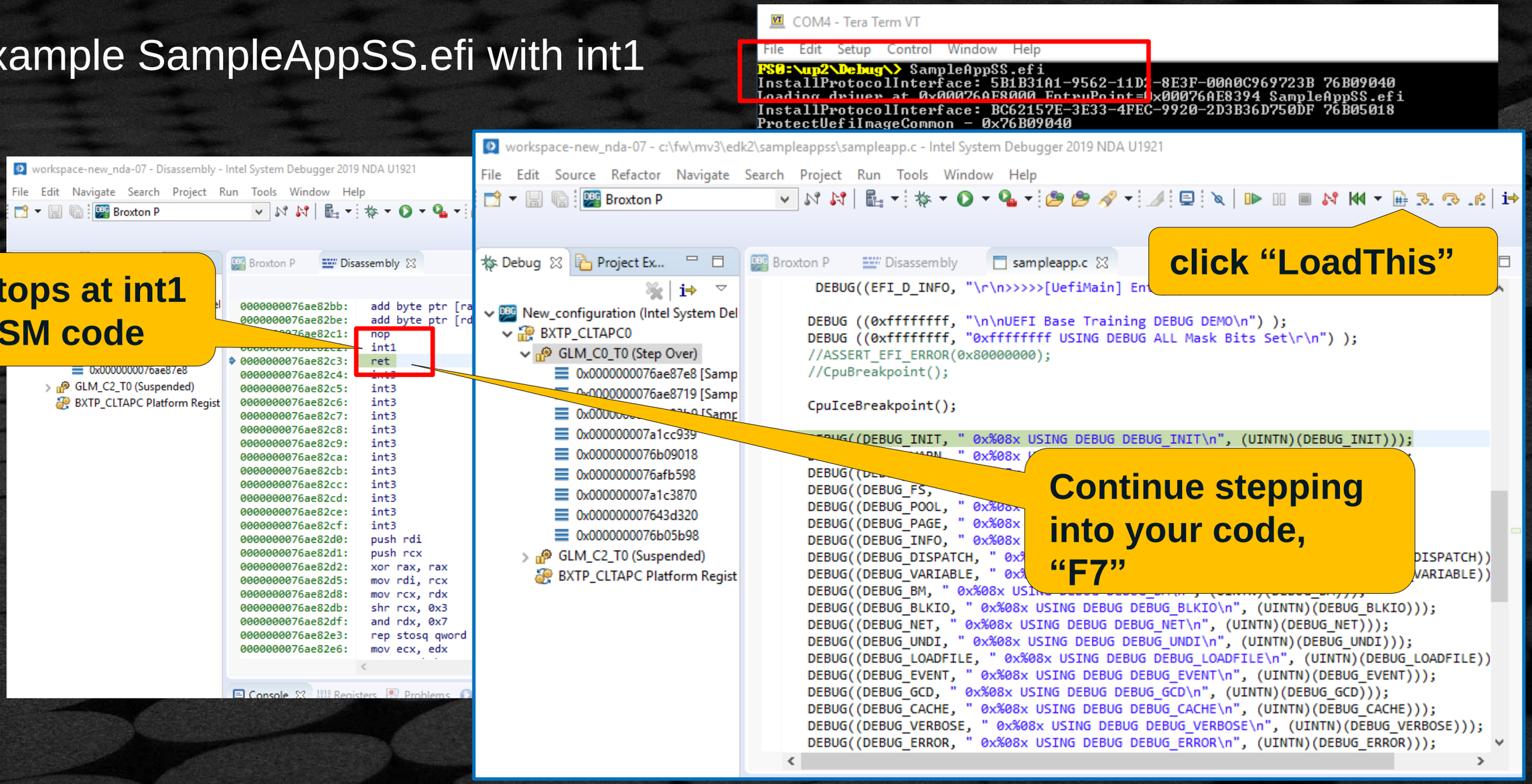
Enable INT1 Exception handler- Console

On "System Debugger Console" of TCF debugger, issue the command - "threads[0].breakpoints.add(condition="int1exception") ", then it will enable INT1Exception trap from now

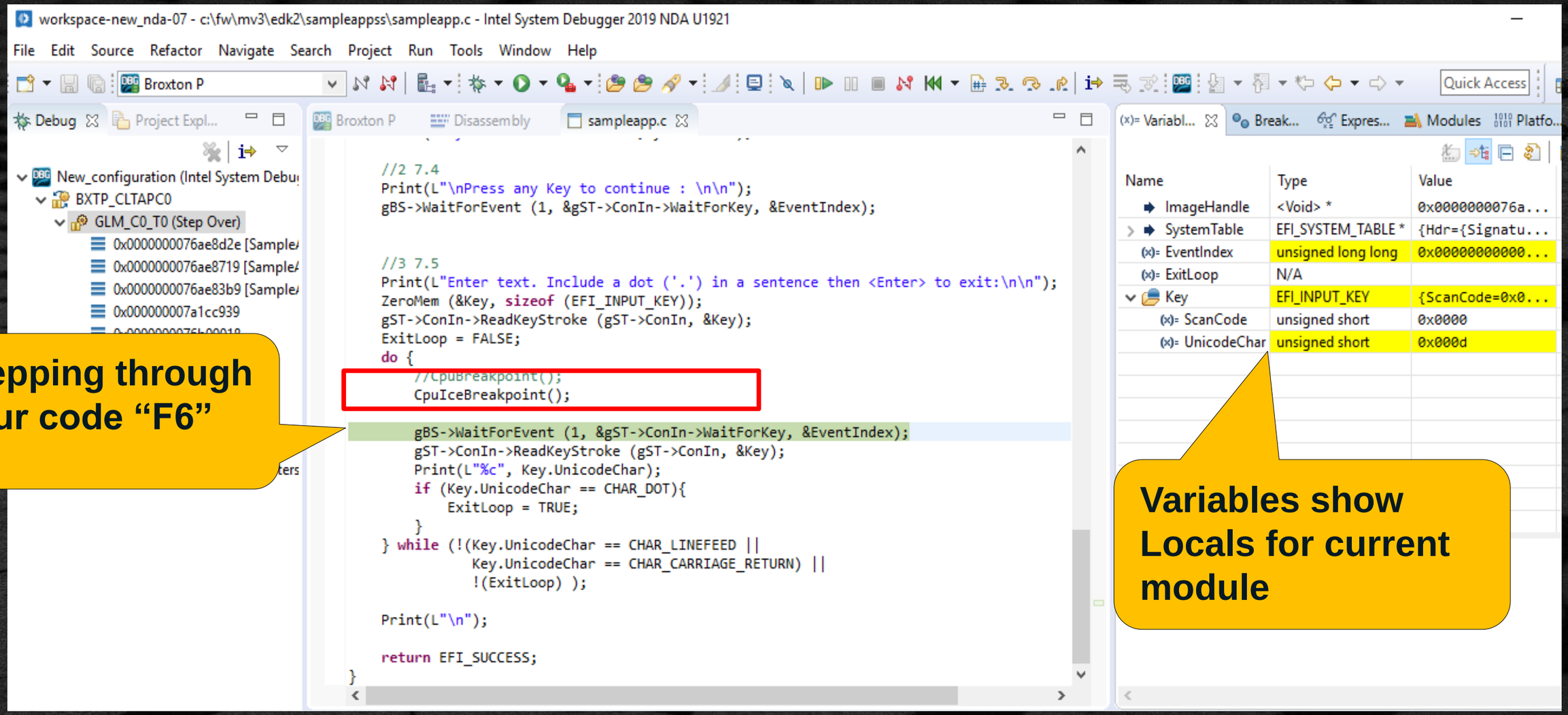


UEFI Application w/ CpuIceBreakpoint

Example SampleAppSS.efi with int1



UEFI Application w/ CpuIceBreakpoint



Intel® System Debugger

- Intel® System Debugger software solution for
  - ✓ UEFI BIOS, firmware, bootloader, OS and device driver debug.
- Intel's system software trace solution for
  - ✓ complex hardware/software interaction visible
  - ✓ isolate tricky software bugs quicker
- It supports JTAG via ITP-XDP3 / DCI hosting BSSB / USB3
- It supports UEFI BIOS debug via EDKII debug agent and JTAG
- It is available as part of Intel System Studio

tianocore

SUMMARY

- Intel® System Studio Debugger Overview
  - Identify the Intel® System Studio Debugger host and target
- Getting started with Intel® System Studio Debugger
  - Using IPCCLI
  - TCF GUI
  - XDB GUI
- Configure when target has UEFI debug agent
- Using the INT1 ICEBP instruction as a debug break point


Questions?

Return to Main Training Page

Return to Training Table of contents for next presentation link

tianocore