SourcePoint WinDbg

Getting Started Guide for the

AAEON UP Xtreme i11

Revision 1.2

Contents

Revision History 4
Welcome! 5
Introduction 6
Configuring the target and setting up pre-requisites – Getting Started 9
How to Establish a SourcePoint WinDbg Session 11
Step 1: Connect SourcePoint to the target 11
Step 2: Start WinDbg via a SourcePoint macro 13
Step 4: Load symbols with the LoadCurrent macro 16
Getting SourcePoint to display module names as well as function names 26
Using Intel Processor Trace 28
Event Trace 34
First Step: Configuring the Intel Trace Hub 34
Second Step: Set up Architectural Event Trace 35
Getting Started with Hyper-V/VBS Debug 39
Troubleshooting Tips and Errata 53
Windows crashes 53
WinDbg Classic is better than WinDbgX 53
Pause in Initial Symbol Load 54
LoadCurrent versus LoadAll 54
COM(32) Surrogate 54
Viewing the Stack 54
LoadCurrent intermittently fails in User code 55
Breaks are not process-aware 55
Mangled function names 55
WinDbg FP register display is not working 55
Troubleshooting Tips on Hyper-V/ VBS Enabled Targets 56
VM Resume breakpoint with Intel PT crashes the target 56
Hardware breakpoints don’t work well in the Secure Kernel 56
AET only partially functional 56
Support for VM Exit Reasons > 63 57

Intel PT Call Chart does not work reliably 58
Conclusion 60

Revision History

Revision Number	Description	Date
1.0	Original document.	December 6, 2023
1.1	Update for beta release SourcePoint 7.12.52. Initial support for HV/VBS debug.	March 31, 2024
1.2	Production release documentation for 7.12.53. WinDbg Classic (as opposed to WinDbgX) is now the preferred WinDbg application.	May 5, 2024

Welcome!

Thank you very much for your use of our SourcePoint WinDbg product! We hope that you get great value using our tool for your debugging efforts.

If you do encounter issues or have questions on the use of SourcePoint WinDbg, please visit our support site at https://www.asset-intertech.com/support/ to get in touch with our Support organization.

As with any new tool, mastering SourcePoint takes an investment in terms of time and effort. JTAG-based debug is a specialized area, and the JTAG, EXDI and Windows interactions sometimes behave non-deterministically – Windows in particular sometimes objects to a hardware-assisted debugger being present. We've done our best to mitigate these issues. Nonetheless, you may encounter behavior that seems non-intuitive or even wrong. If so, check out the Troubleshooting section of this document first. Secondly, view the Troubleshooting section of the Getting Started Guide for the AAEON UP Xtreme i11. Thirdly, refer to the Release Notes in the SourcePoint Academy. Finally, if you're still stuck, contact us at our Support page. We'll do our best to get you up and debugging again.

For those who are new to SourcePoint, it is ***highly recommended*** to review our Getting Started Guide for the AAEON UP Xtreme i11 to get a jumpstart before using SourcePoint WinDbg. That, and the rest of the content within the SourcePoint Academy, are ***essential background reading*** for those new to SourcePoint.

Introduction

It is recommended that all users have a working familiarity with SourcePoint installation, licensing, and basic usage. Installation and licensing are described fully in the SourcePoint Installation and Licensing Guide that is obtained from ASSET upon initial receipt of your shipment. For basic SourcePoint usage on the AAEON UP Xtreme i11, go to the SourcePoint Academy and read the Getting Started Guide for the AAEON UP Xtreme i11 to learn the basics of SourcePoint run-control and trace.

The content that follows is based upon our using the AAEON UP Xtreme i11 Tiger Lake board. Of course, any Intel board that can support either direct XDP (open-chassis) access, or the Intel Direct Connect Interface (DCI) (closed-chassis) is suitable. Intel customers with the appropriate NDA will have access to a plethora of Customer Reference Boards (CRBs) that have XDP and DCI enabled out of the box. The AAEON UP Xtreme Whiskey Lake board (for which UEFI source code is available) is also a good platform – it is the only publicly available platform that is available with a booting open-source Tiancore UEFI build, so you can debug Windows and the BIOS at the same time. More information on the Whiskey Lake board is available here:

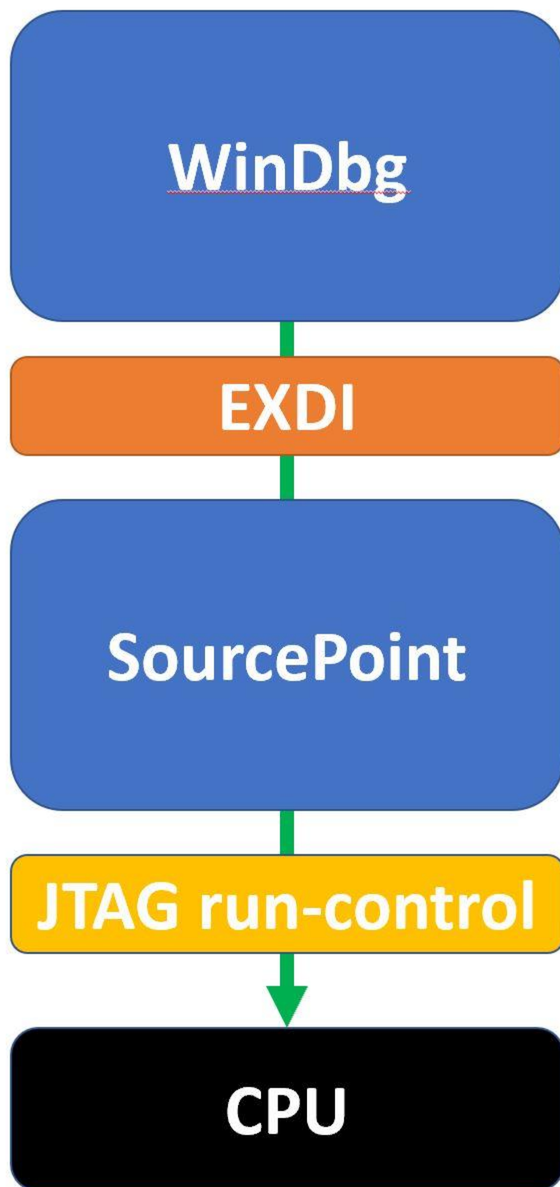
JTAG Debug using DCI on the AAEON UP Xtreme Whiskey Lake board

Hypervisor and OS Kernel Debug with DCI on the AAEON Whiskey Lake board

A key pre-requisite is that the platform must have debug consent enabled; that is, it must be in a debuggable state. If XDP access is available on the board, you can connect to it via the ASSET ECM-XDP3e hardware probe. Some small number of Commercial-Off-The-Shelf (COTS) boards support direct access via the Intel Direct Connect Interface (DCI). These include the AAEON UP Xtreme, and the AAEON UP Xtreme i11. Documenting the steps needed to enable JTAG-based debug on other boards is beyond the scope of this Guide; interested readers are referred to Satoshi Tanda's Debugging system with DCI and WinDbg.

The SourcePoint WinDbg application will work on Intel-based Windows targets, on all CPUs that are supported by SourcePoint run-control. As of the time of this writing, all mainstream Intel CPUs are supported. AMD support will be in a future release.

A block diagram of how WinDbg is integrated with our SourcePoint debugger is as below:



The EXtended Debug Interface (EXDI) is used to connect a WinDbg debugging session to an existing SourcePoint JTAG-based connection to a target.

WinDbg is the controller in all transactions over EXDI, and SourcePoint is the worker. That is, the solution is most stable when run-control based operations (that is, Break, Go, single-step, etc.) are initiated via WinDbg. There are exceptions, particularly in the cases of using enhanced breakpoints for Hyper-V debug and Intel Trace features, that we will discuss later. But, in general, WinDbg issues debug primitive commands down to SourcePoint, which in turn uses JTAG-based run-control to perform operations on the

target. Then, SourcePoint presents the results data back to WinDbg over the EXDI connection.

Power Tip: The UP Xtreme i11 boots to the UEFI shell when initially purchased. It is necessary to install Windows on the target. There are numerous references online on how to do this: it is recommended to go to the AAEON https://github.com/up-board/up-community/wiki/Windows-GSG site for helpful tips. In terms of driver installation, in most cases all that's needed is to install the Intel Graphics (igxpin.exe) – to improve the monitor resolution – and, optionally, the Intel LAN.

Power Tip: Be sure that your target has sufficient memory and storage to accommodate your Windows debugging needs. We typically recommend 16GB RAM, and a 256GB SSD.

Before we get started, the target needs to be configured to not interfere overmuch with JTAG-based run-control. Then, the steps needed to set up a debugging session will be covered.

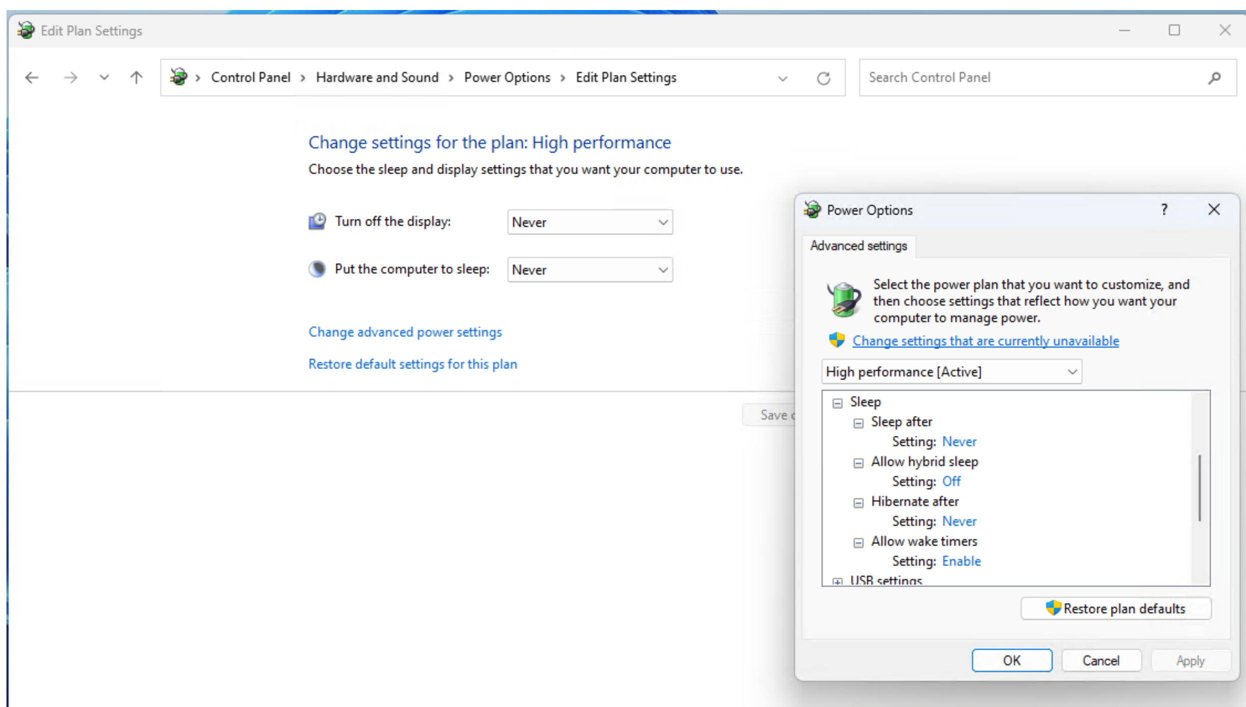
Configuring the target and setting up pre-requisites – Getting Started

Firstly, disable the UEFI TCO Watchdog timeout, and set the CRB Advanced Debug Settings to "Platform Debug Consent" to USB DbC2 timeout, as described in the SourcePoint Getting Started Guide for the AAEON UP Xtreme i11. *This is an essential step to ensuring that the Tiger Lake target will function properly with run-control. Otherwise, the target will reset itself asynchronously, and the Intel Trace Hub won't work, disrupting your debugging session.*

We'll also need to prevent Windows from changing power states from disrupting run-control prematurely, and VMX and VBS need to be disabled.

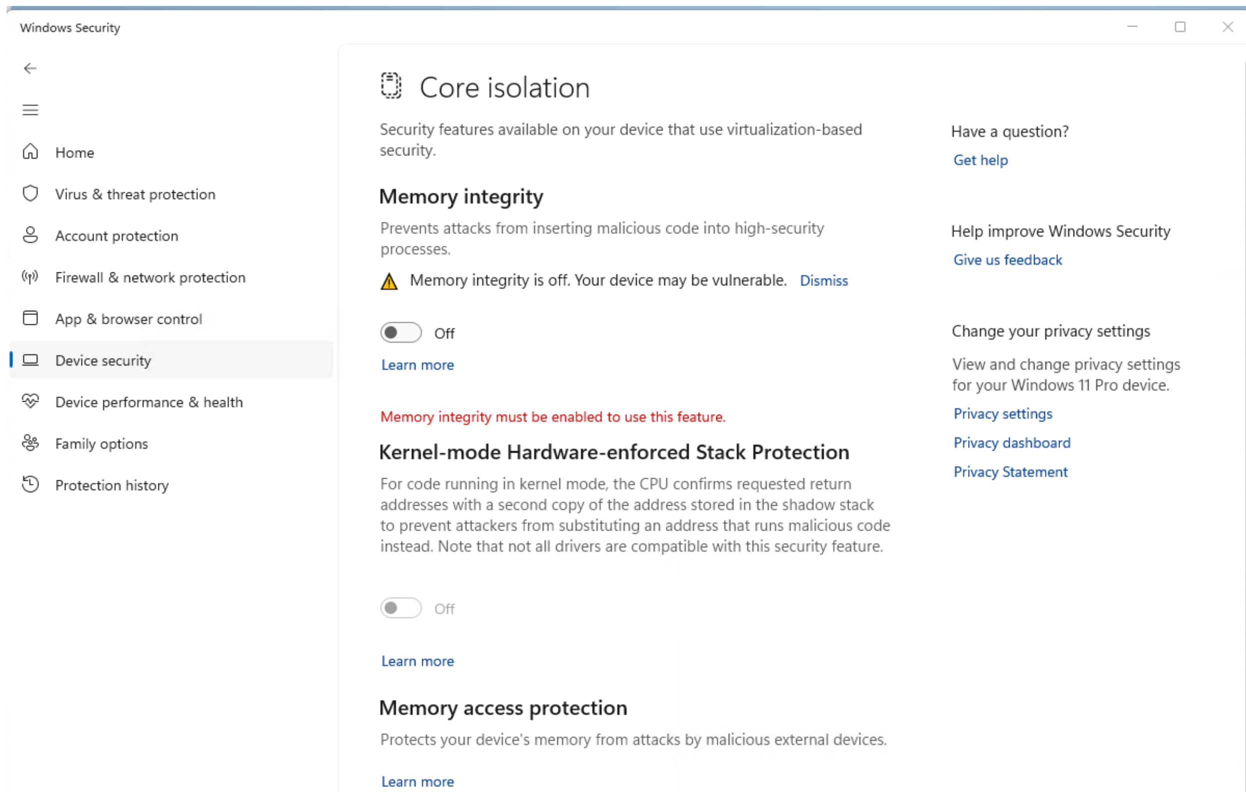
These steps are highly recommended (as of the time of writing) to have a successful initial debugging session, especially for newcomers to SourcePoint WinDbg.

To adjust the power settings in Windows, open the Control Panel > Hardware and Sound > Power Options > Change plan settings > Change advanced power settings and set these per the below (use High performance dropdown). It also helps to set "Turn off the display" and "Put the computer to sleep" both to "Never":



©2024 ASSET InterTech, Inc.

For Windows VBS, go into Windows Security > Device security > Core isolation details, and ensure that Memory Integrity is off:



For VMX, boot the Tiger Lake board to BIOS settings menu (pressing the F7 key when restarting), enter the Advanced BIOS Setup (by entering the password upassw0rd) and follow the menu path `CRB Setup > CRB Advanced > CPU Configuration` and change “Intel (VMX) Virtualization Technology” to **Disabled**. Save and exit (pressing F4) and the target will reset.

Power Tip: Go to `CRB Setup > CRB Advanced > Platform Settings > VTIO` and make sure it is set to **Disabled**. This is the default in the AAEON Tiger Lake Debug BIOS, but it’s worthwhile checking.

One last thing: To avoid the WinDbg error message “Unable to read debugger data block header” that indicates kernel debugging is not enabled, execute the command:

```
>bcdedit /debug on
```

on the target from an Administrator command prompt, then reboot the target.

Note that this is not absolutely necessary if you're solely going to be using SourcePoint (with no WinDbg connection) for your debugging.

Now you're ready to set up a debugging session.

How to Establish a SourcePoint WinDbg Session

NOTE: With SourcePoint WinDbg, there is no need for the kdnet Ethernet connection, as all the traffic is over EXDI and the specialty USB cable.

Four steps are needed to begin a debugging session with SourcePoint WinDbg:

1. Connect SourcePoint to the target
2. Run the StartWinDbg macro
3. Issue a Break from WinDbg
4. Load symbols with the LoadCurrent macro.

Step 1: Connect SourcePoint to the target

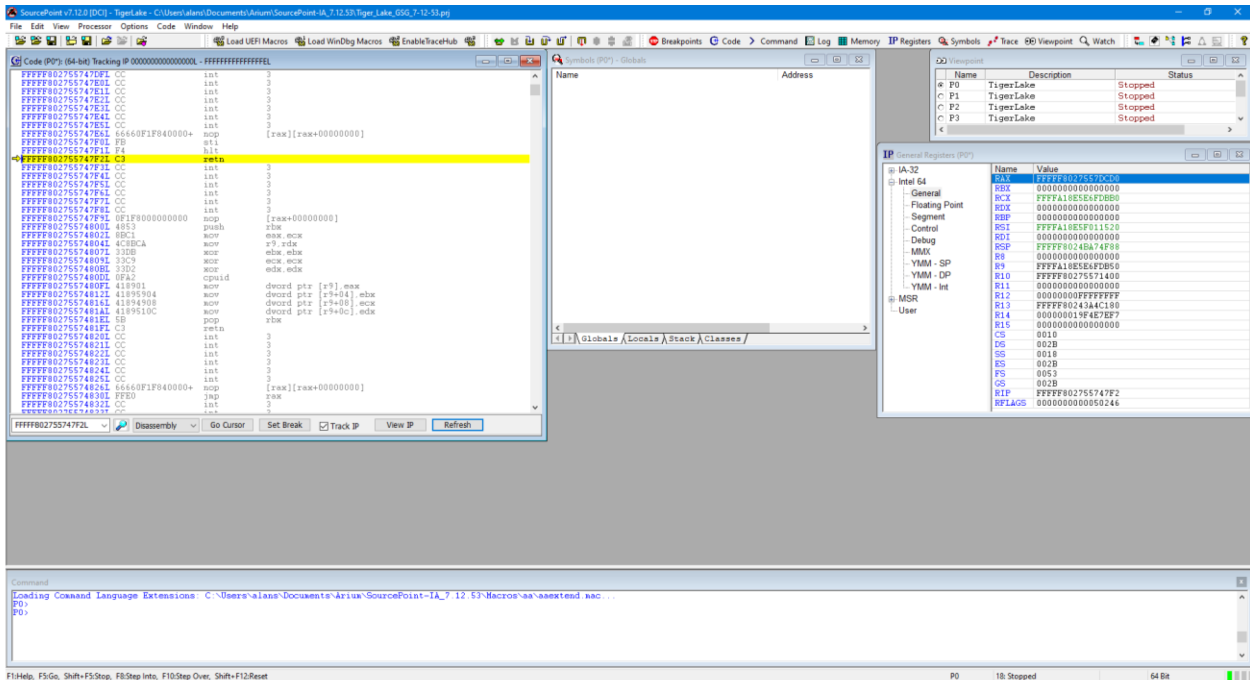
Boot the target to Windows. Log into the Windows desktop.

Follow the steps as described in the Getting Started with SourcePoint section of the Getting Started Guide for the AAEON UP Xtreme i11.

Halt the target by hitting the Stop button in the SourcePoint Icon Toolbar at the top:

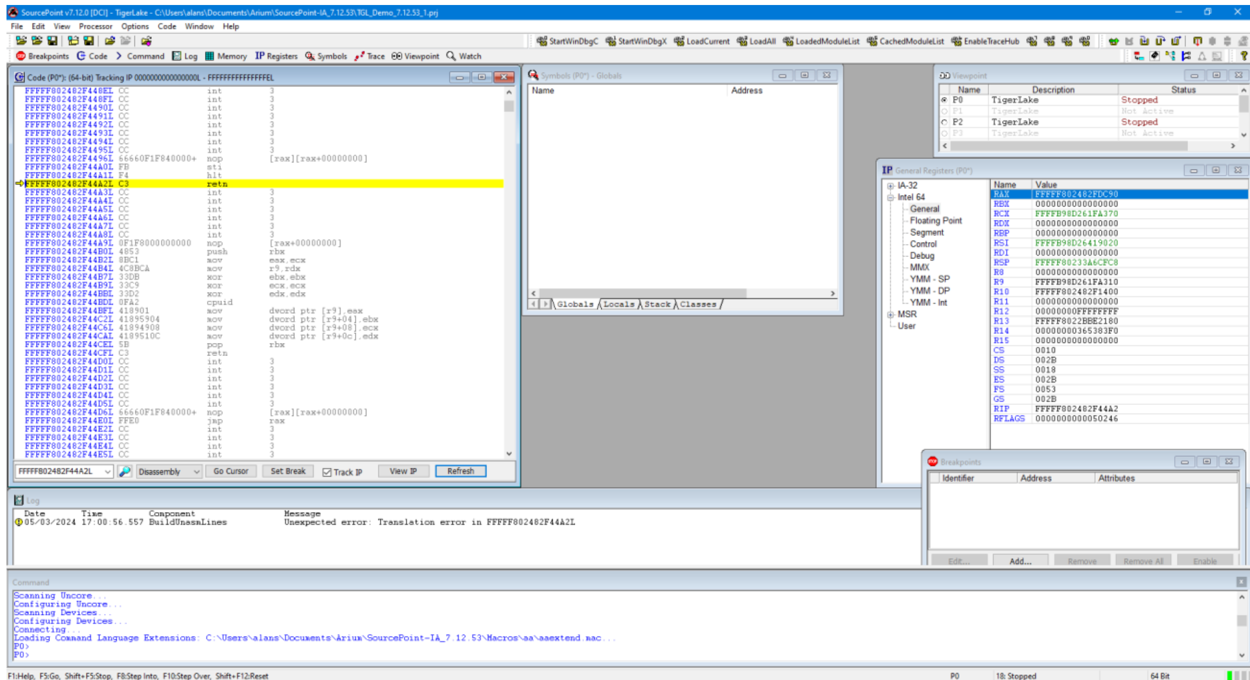
You will have to hit the Refresh button to see code displayed in the Code window. This transitions the Code view out of Safe Mode.

Your screen should then look something like this:



Now, you have a choice to set up your environment for Windows or UEFI debugging in this session.

Click on the `Load WinDbg Macros` button at the top of the screen. This will enable a number of new “Windows debugging friendly” macros available for later use. The screen should look like this:



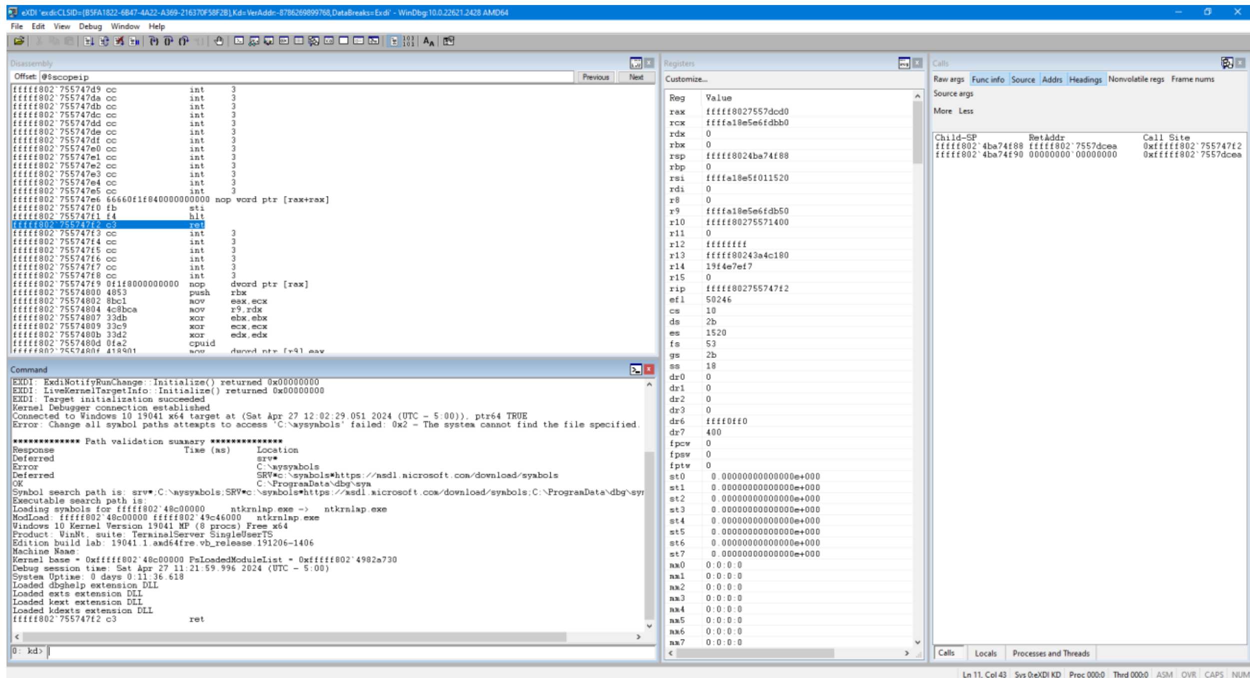
You now have extra buttons showing up, including:

StartWinDbgC	This initiates a debug session with WinDbg Classic
StartWinDbgX	This initiates a debug session with WinDbgX
LoadCurrent	This load symbols into SourcePoint at the current RIP
LoadAll	Loads all current context symbols into SourcePoint
LoadedModuleList	Displays all loaded modules
CachedModuleList	Shows the module list that is currently cached

Step 2: Start WinDbg via a SourcePoint macro

Next, it is time to run the SourcePoint macro that launches WinDbg and establishes the EXDI connection. For simplicity, click on either StartWinDbgC (Classic) or the StartWinDbgX macro button at the top of the screen. After about 30 seconds, WinDbg will open:

©2024 ASSET InterTech, Inc.



Power Tip: You have a choice of launching WinDbg Classic or WinDbgX via the macro buttons at the top. With this release of SourcePoint, ***WinDbg Classic is more stable and higher performance***. See our [Troubleshooting](#) section. Alternatively, at the SourcePoint Command line, you can type in StartWinDbg(true) to start a WinDbg Classic session, or StartWinDbg(false) to start a WinDbgX session.

The target will be halted as part of this process, assuming the VBS override environment variable (_NT_WINDBG_VBS) has not been set:

Environment Variables

User variables for alans

Variable	Value
GPU_FORCE_64BIT_PTR	0
GPU_MAX_ALLOC_PERCENT	100
GPU_MAX_HEAP_SIZE	100
GPU_SINGLE_ALLOC_PERCE...	100
GPU_USE_SYNC_OBJECTS	1
OneDrive	C:\Users\alans\OneDrive - Volaris Group
OneDriveCommercial	C:\Users\alans\OneDrive - Volaris Group

New... Edit... Delete

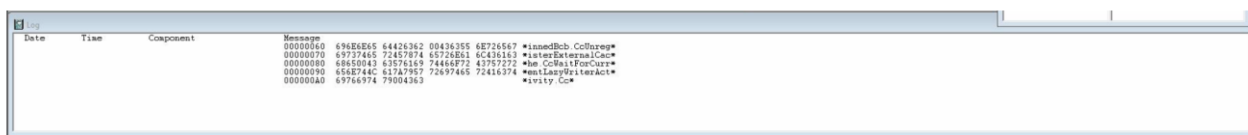
System variables

Variable	Value
_NT_WINDBG_VBS	FALSE
_NT_WINDBG_WORKSPACE	EXDI
ASSET	C:\ScanWorks
ComSpec	C:\WINDOWS\system32\cmd.exe
DALINSTALLDIR	C:\IntelSWTools\system_studio_2019_nda_1945\tools\DAL_1.1942.10...
DriverData	C:\Windows\System32\Drivers\DriverData
DXSDK_DIR	C:\Program Files (x86)\Microsoft DirectX SDK (June 2010)\

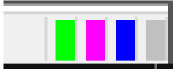
New... Edit... Delete

OK Cancel

SourcePoint will then look for the KdVersionBlock structure, read the kernel memory and retrieve all the symbol information needed to match what WinDbg has (in terms of the Microsoft symbol server, or a local symbol cache). If you have the SourcePoint Log window open, you may see the symbol information being uploaded, but only for WinDbgX:



If you don't have the Log window open, you will nonetheless see the SourcePoint "Dashboard Lights" at the bottom right lighting up as the JTAG-based memory reads are done:



When the symbol load is complete, you will see that WinDbg and SourcePoint break at the same place.

The SourcePoint Code and WinDbg Disassembly window show the same location. Both are typically (but not necessarily) halted on logical processor 0, at a `RET` instruction, as can be seen in the above image.

Step 4: Load symbols with the LoadCurrent macro

Symbols that are visible to WinDbg have to be made visible to SourcePoint as well, if we're going to get the most out of the joint solution.

SourcePoint has the ability to view Windows' symbols on its own, with no connection to WinDbg. To see what this looks like, just launch SourcePoint and load your Project, and follow the following steps:

Ensure that the target is in a Stopped state.

Click on the `LoadCurrent` macro button in the SourcePoint Icon toolbar at the top. After about 10 seconds, the SourcePoint Symbols window will display the module that the current instruction is in:

Interestingly, SourcePoint will display the symbols associated with intelppm.pdb (sometimes). WinDbg does not generally display those symbols.

Expand the Labels within the Symbols window, and then you will see it populated with all functions that are in the current module, for example:

Symbols (P0*) - Globals

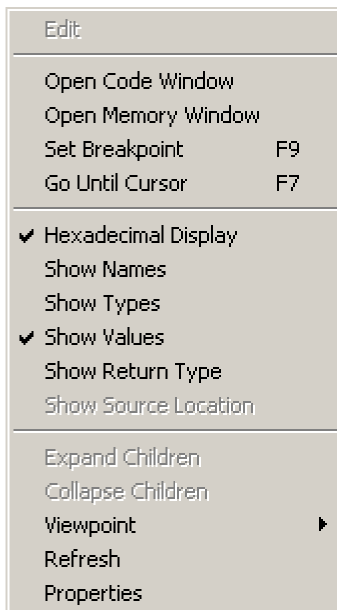
Name	Address
PspStorageGetObject	FFFFF8075F1A7F58L
PspStorageInsertObject	FFFFF8075F0C54C0L
PspStorageMakeSlotReadOnly	FFFFF8075F0C280CL
PspStorageRemoveObject	FFFFF8075F3BBE74L
PspStorageReplaceObject	FFFFF8075F3BBF94L
PspSubtractAccountingValues	FFFFF8075F3B74C8L
PspSysAppIdClaim	FFFFF8075F4779A0L
PspSyscallProviderOptIn	FFFFF8075F3B8FFCL
PspSyscallProviderServiceDispatch	FFFFF8075EE38BD0L
PspSyscallProviderServiceDispatchGeneric	FFFFF8075F3B91A8L
PspSystem32String	FFFFF8075F4771E8L
PspSystemCpuPartitionName	FFFFF8075F59E4E0L
PspSystemDriveString	FFFFF8075F477208L
PspSystemRootString	FFFFF8075F477218L
PspSystemRootSymlinkName	FFFFF8075F4779C0L
PspSystemRootTargetPrefix	FFFFF8075F4771D8L
PspSystemThreadStartup	FFFFF8075ED569A0L
PspTeardownPartition	FFFFF8075F3BA740L
PspTerminateAllProcessesInJobHierarchy	FFFFF8075F0A56F8L
PspTerminateAllThreads	FFFFF8075F1B3830L
PspTerminatePicoProcess	FFFFF8075F3B9CF0L
PspTerminateProcess	FFFFF8075F0A7624L

Globals | Locals | Stack | Classes

Power Tip: If WinDbg accesses symbols outside of intelppm.pdb (which it will during any typical debugging session), you'll need to run another "LoadCurrent.mac" to additionally access these new symbols within SourcePoint.

Power tip: Right-click on a function name within the SourcePoint Symbols window, and you'll see a rich number of capabilities that can be applied to that function, such as setting breakpoints, opening the function's Code window, etc.

All the Windows kernel function name symbols are displayed in the SourcePoint symbols window, under the `Globals` tab. You can right-click in the window to see the function addresses as well as function names. Right-clicking on a function name gives you the context-sensitive options to work with these functions:



Now, it is possible to see the power of the two applications applied together. As an example, go into WinDbg and set a breakpoint on the entry point to the function `MmCreateProcessAddressSpace`:

```
bp nt!MmCreateProcessAddressSpace
```

Then hit Go within WinDbg.

Sometimes the breakpoint is hit right away. You might need to move the target's mouse around, or open a window on the target, before the breakpoint is hit.

You can then see the break in both applications. Do a `LoadCurrent` from within SourcePoint. You can see that the Code windows between the two applications are symmetrical:

Power Tip: between individual "Go" commands and breaks, the context of the code will often change (i.e. the value of CR3 changes). Even though the module name will still appear in the SourcePoint Symbols window, the needed symbols will no longer appear in its Code window. Hit the `LoadCurrent` button again to re-display the symbols.

Alternatively, take the following steps to ensure the Code window context is updated upon each break:

Under the File menu, select `Macro > Configure Macros…`

Click on the `Event Macros` tab.

Select Event: `Breakpoint (any)`

Then browse in the main folder, and select `Events.mac`.

This will slow down breakpoints ever so slightly, but it will ensure the code context is refreshed without manual intervention.

Power Tip: Once the PDB file is identified, SourcePoint will search for the symbol file in WinDbg's stored Symbol path, and then if not found, its Cache path. The symbol path in most WinDbg installations is something similar to:

```
srv*C:\Symbols*http://msdl.microsoft.com/download/symbols
```

and **SourcePoint has no knowledge of HTTP access**, so it will extrapolate only the C:\Symbols portion, and next go to, and include, the cache path.

When symbols are loaded solely with SourcePoint (WinDbg is not launched), SourcePoint will refer to the WinDbg path local to your debugging PC. Type:

```
sympath
```

in the SourcePoint Command window, and you'll see where SourcePoint will look. For example:

```
c:\symbols;C:\ProgramData\dbg\sym
```

You can explicitly set the symbol path to be that of WinDbg Classic's, by typing in the Command line:

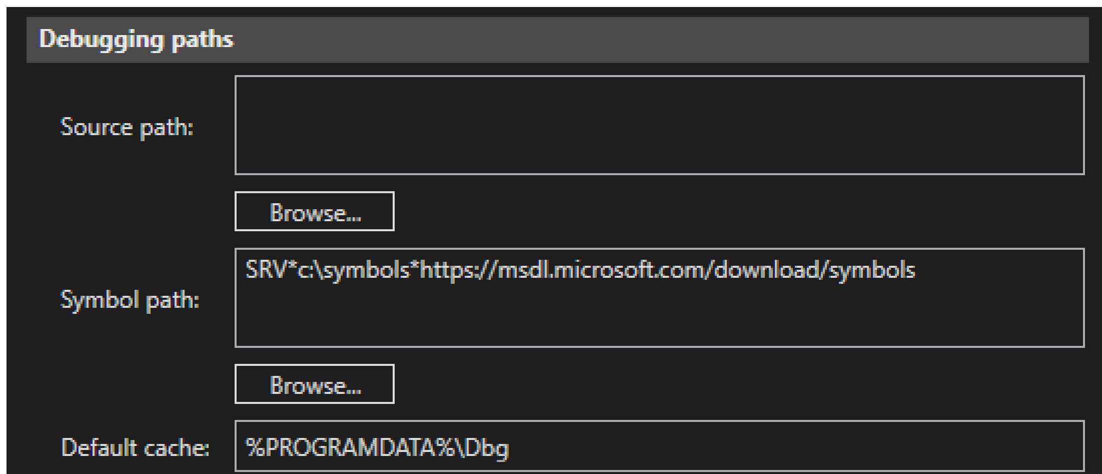
```
windbgc
```

Alternatively, typing:

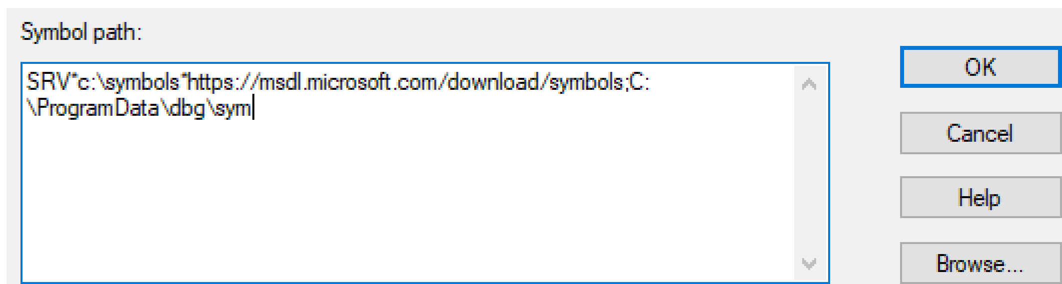
```
windbgx
```

sets up the symbol path as defined within WinDbgX.

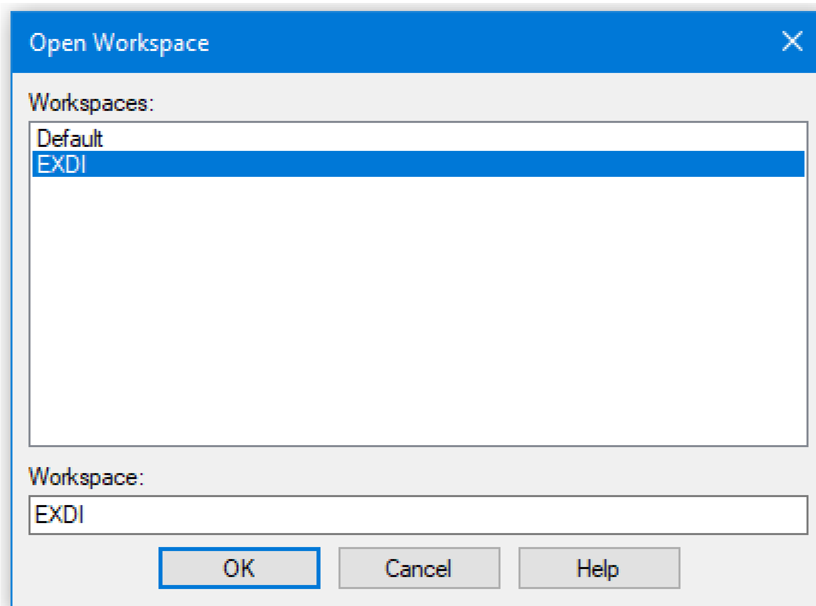
Symbol and cache paths are a little tricky with WinDbg Classic. WinDbgX allows for an explicit description of the cache path:



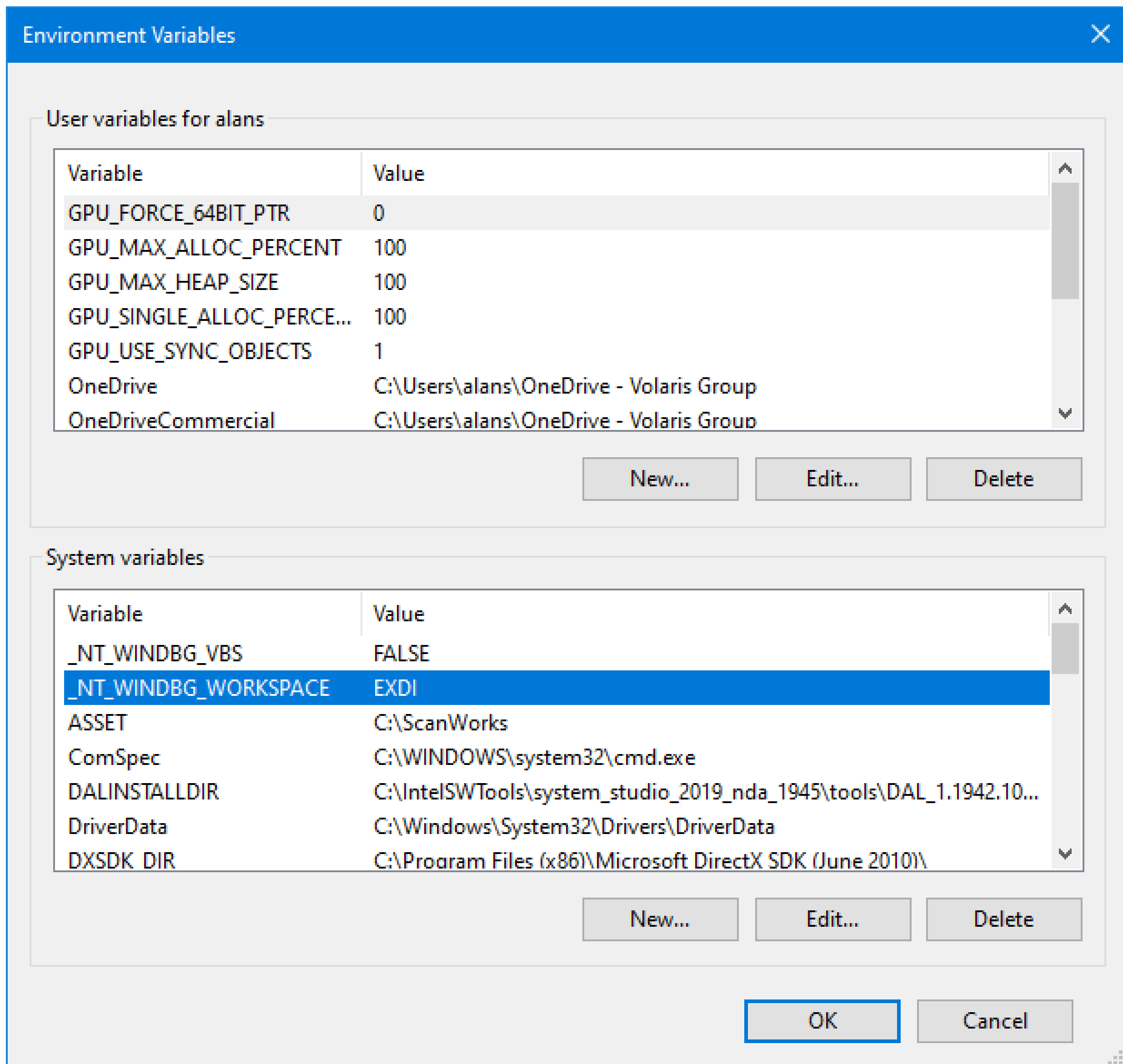
Whereas WinDbg Classic does not; you have to embed it in the Symbol path, and that is squirrelled away in the registry:



It is important to be aware of Workspaces. SourcePoint will use the settings detected for default Workspaces (Default, AMD64 etc), but best practice is to create a separate Workspace; in this case, we name it EXDI:



Be sure to load the Workspace after you launch WinDbg, and use the SourcePoint `windbgc` and `sympath` commands to ensure the path is latched in. Once a Workspace is decided on, you can force the specific Workspace by setting an environment variable (_NT_WINDBG_WORKSPACE):



Editor's Note: WinDbg tends to store an extraneous "\sym" in the cache path that needs to be worked around. You'll see that SourcePoint handles that properly.

Power Tip: If you're using SourcePoint by itself, it may be helpful to store as many symbol files locally as possible. Use this following command (on the target PC) to download them, and then copy them over to your cache path on your host debug PC:

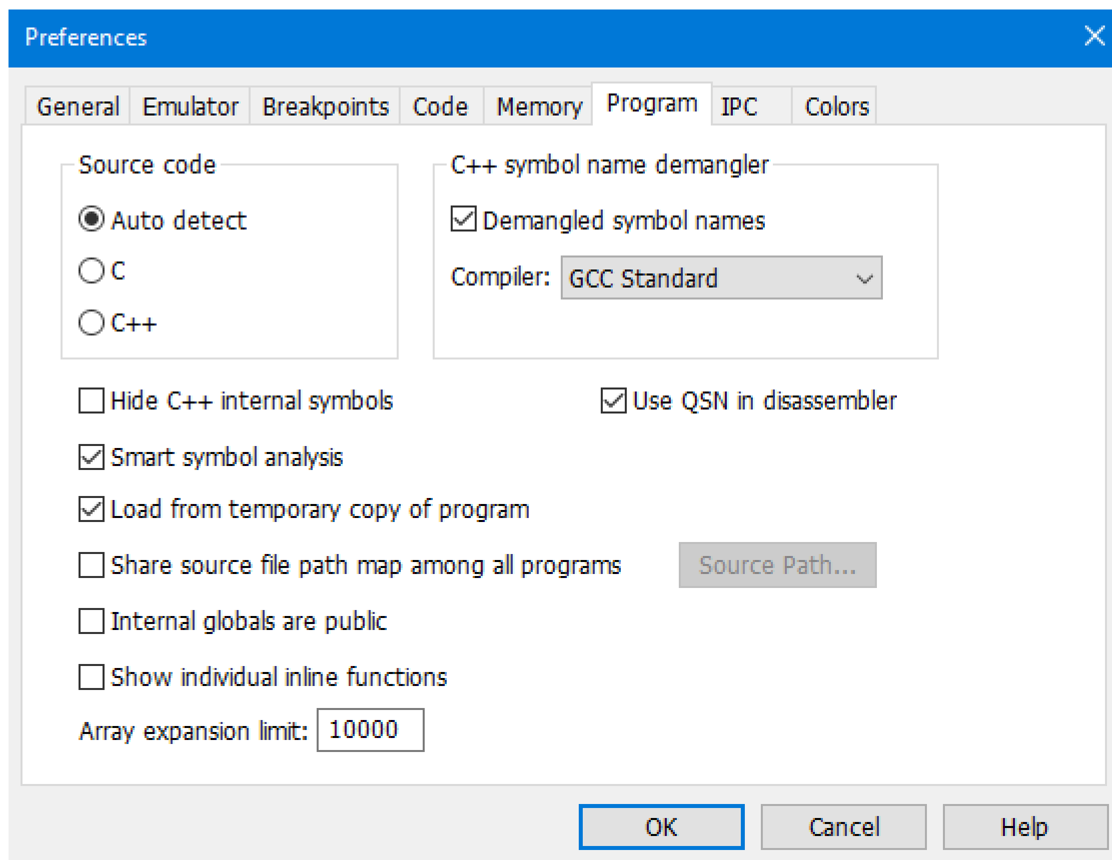
```
"C:\Program Files (x86)\Windows Kits\10\Debuggers\x86\symchk.exe" /r
c:\windows /s
SRV*c:\symbols\*http://msdl.microsoft.com/download/symbols
```

It's a good idea to disable Ethernet on the target for your debugging, to avoid Windows Update changing the modules and their GUIDs, requiring a reload to update the cached symbol files.

Getting SourcePoint to display module names as well as function names

WinDbg displays the fully qualified symbol name, including the module name, in its windows, as in `nt!MmCreateProcessAddressSpace`. SourcePoint truncates them by default to solely the function name, as in `MmCreateProcessAddressSpace`.

The module name prefix can be displayed by enabling SourcePoint's Qualified Symbol Name (QSN) format. In the Options menu, select Preferences, and click on "Use QSN in disassembler".



The Code window display will now look something like this:

```
Code (P0*): (64-bit) Tracking IP 0000000000000000L - FFFFFFFFFFFFFFFEL
FFFFF80274682E07L B948000000         mov      ecx,00000048
FFFFF80274682E0CL 0FB6D0             movzx    edx,al
FFFFF80274682E0FL 418895FA000000     mov      byte ptr [r13+000000fa],dl
FFFFF80274682E16L 8BC2               mov      eax,edx
FFFFF80274682E18L 48C1EA20           shr      rdx,20
FFFFF80274682E1CL 0F30               wrmsr
FFFFF80274682E1EL 4180A5F8000000FE   and      byte ptr [r13+000000f8],fe
FFFFF80274682E26L 41BA01000000       mov      r10d,00000001
FFFFF80274682E2CL 44387C2450         cmp      byte ptr [rsp+50],r15b
FFFFF80274682E31L 7476               je       ::ntkrnlmp.PpmIdleExecuteTransition+11b9
FFFFF80274682E33L 410FB6859A7E0000   movzx    eax,byte ptr [r13+00007e9a]
FFFFF80274682E3BL 44887C2450         mov      byte ptr [rsp+50],r15b
FFFFF80274682E40L 84C0               test     al,al
FFFFF80274682E42L 7465               je       ::ntkrnlmp.PpmIdleExecuteTransition+11b9
FFFFF80274682E44L 65488B04252000+    mov      rax,qword ptr gs:[0000000000000020]
FFFFF80274682E4DL 4C8D05ACD1D7FF     lea      r8,qword ptr [fffff80274400000]
FFFFF80274682E54L 418BDA             mov      ebx,r10d
FFFFF80274682E57L 8B4824             mov      ecx,dword ptr [rax+24]
FFFFF80274682E5AL 4488B89A7E0000     mov      byte ptr [rax+00007e9a],r15b
FFFFF80274682E61L 418B9488D024D000   mov      edx,dword ptr [r8][rcx*4+00d024d0]
FFFFF80274682E69L 8BCA               mov      ecx,edx
FFFFF80274682E6BL 8BC2               mov      eax,edx
FFFFF80274682E6DL 83E13F             and      ecx,0000003f
FFFFF80274682E70L 48D3E3             sal      rbx,cl
FFFFF80274682E73L 48F7D3             not      rbx
FFFFF80274682E1CL   Disassembly   Go Cursor   Set Break   Track IP   View IP   Refresh
```

Power Tip: Note that SourcePoint's syntax is slightly different from WinDbg's:

```
WinDbg:            ntkrnlmp!PpmIdleExecuteTransition+11b9
SourcePoint:       ::ntkrnlmp.PpmIdleExecuteTransition+11b9
```

Do a Project Save to save these settings into your Project, so they'll automatically load for your next session.

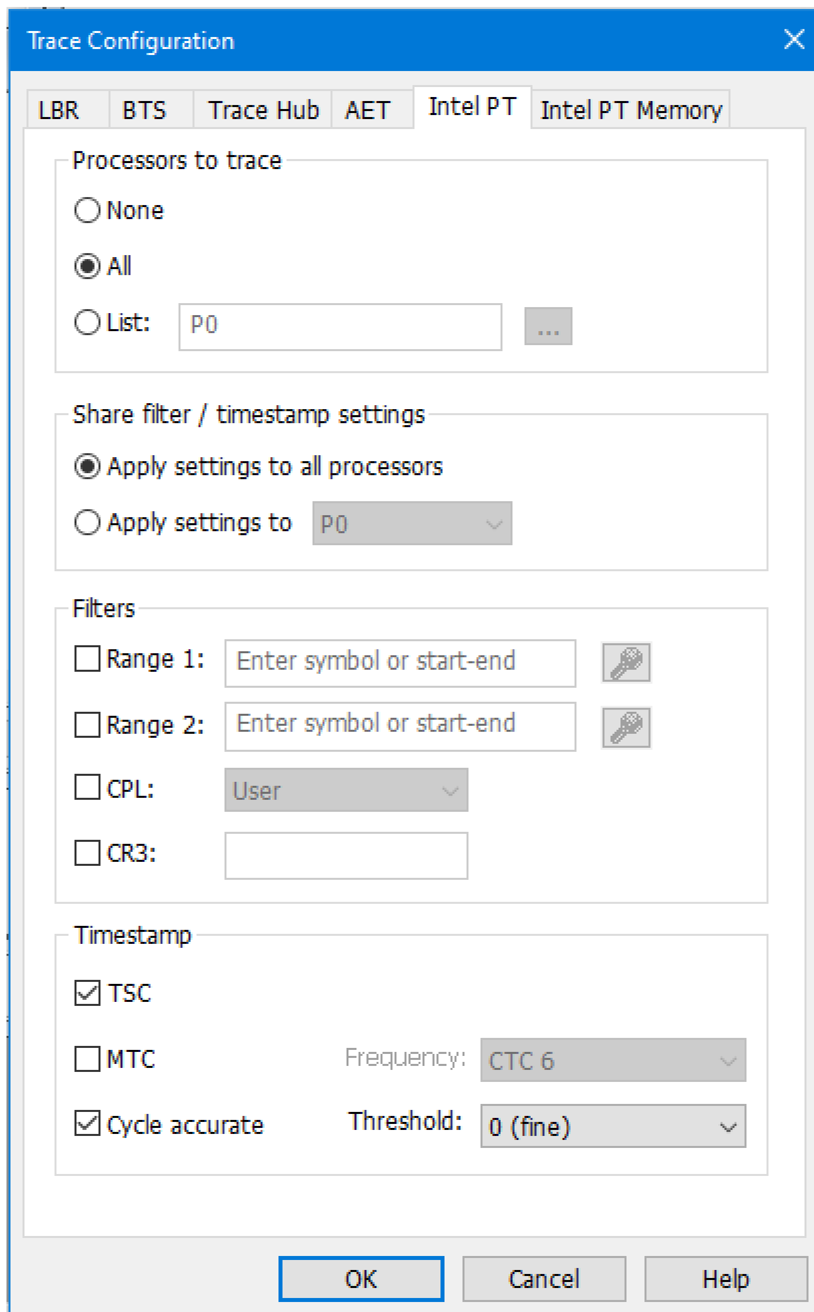
In an upcoming release, we'll make QSN the default in the disassembly for Windows debugging.

Using Intel Processor Trace

Once using run-control is mastered, it is worthwhile testing out some of the SourcePoint advanced trace features, such as Intel PT.

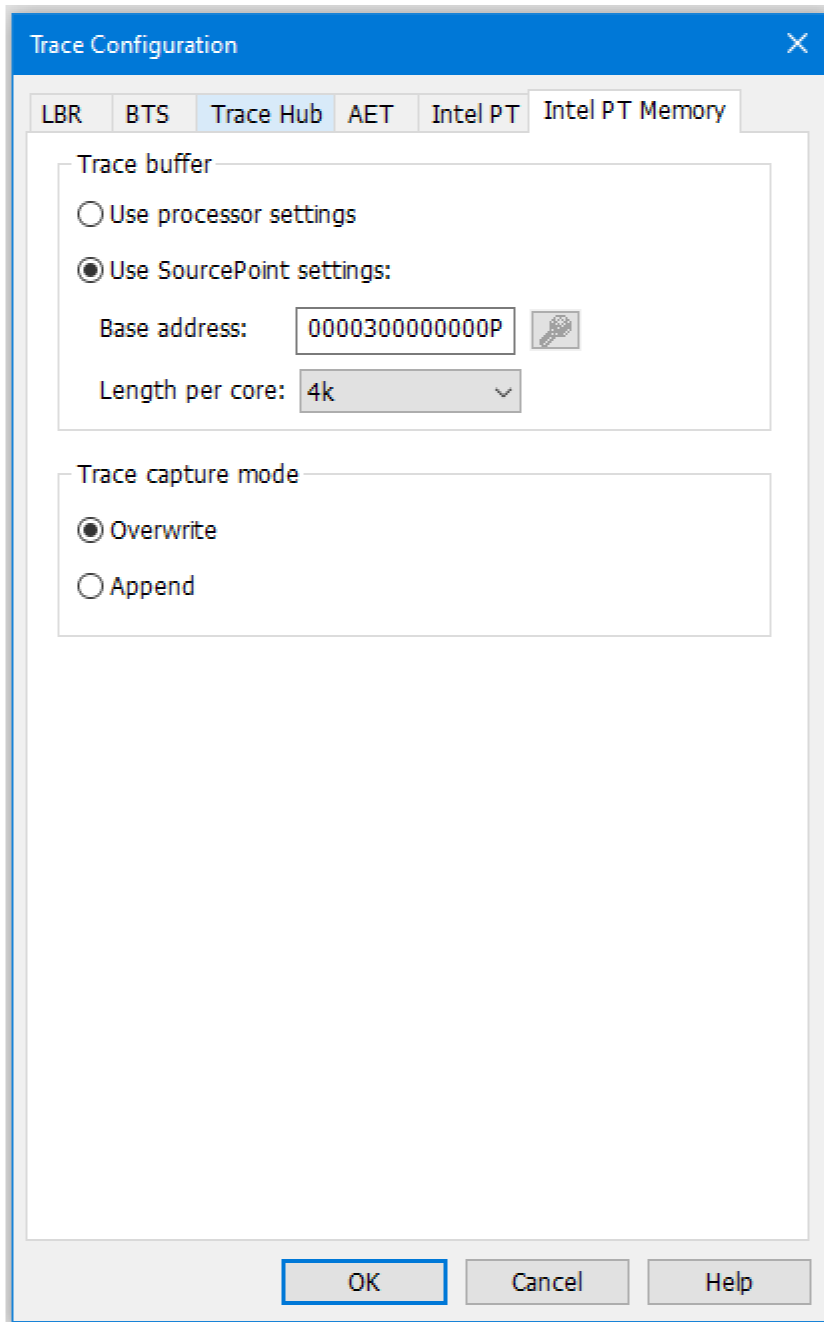
First, ensure that the target is in a Stopped state. If not, issue a Break from within WinDbg.

Then, within SourcePoint, open up a Trace window, click on the Configure, and then click on the Intel PT tab at the top:



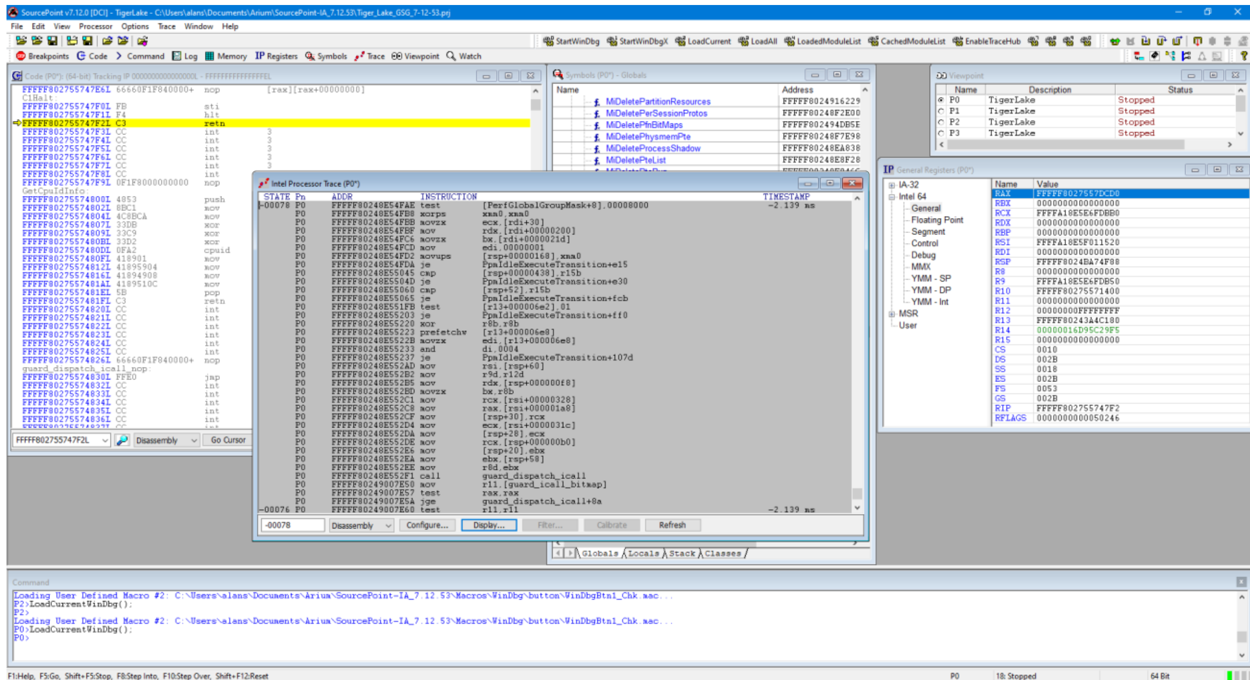
Click on "All" Processors to Trace, or select a processor from the list. Ensure both `TSC` and `Cycle accurate` are enabled.

Then click on the Intel PT Memory tab, and use a spare memory area to store the trace data:



NOTE: "Use processor settings" can be selected if the BIOS has been set up with this. For the UP Xtreme i11 board, this is not the default.

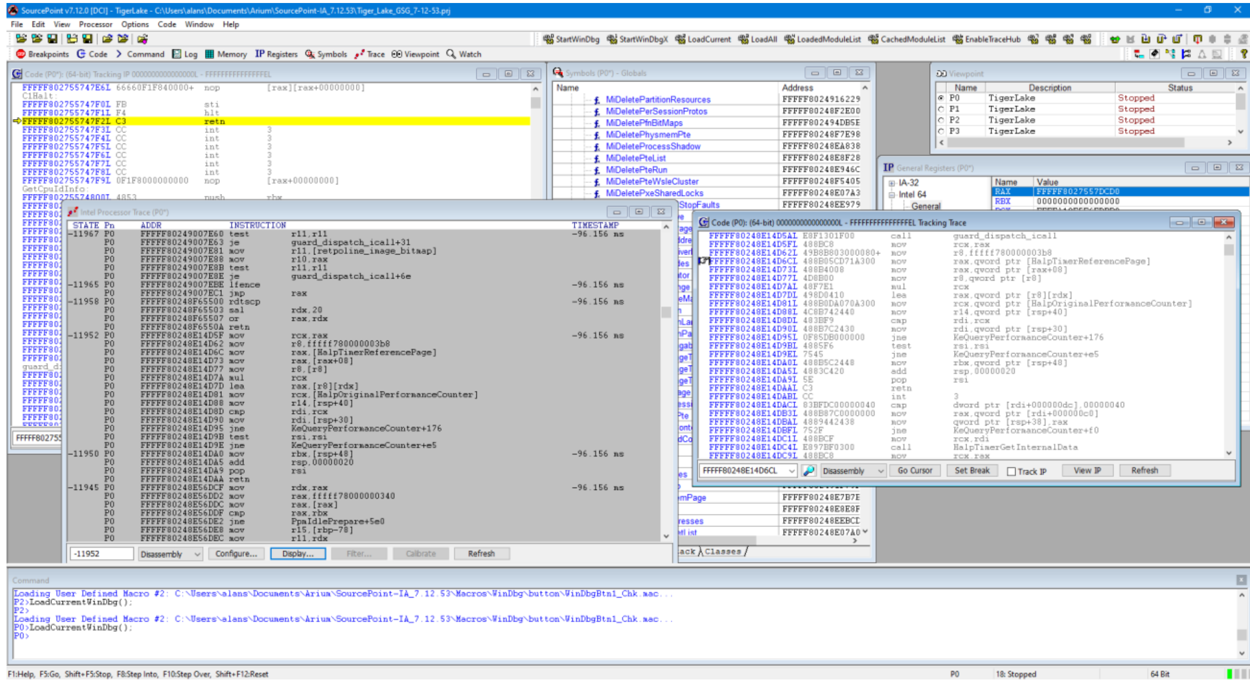
Then hit Go from within WinDbg, and then hit Break, and you will see something like the below in SourcePoint:



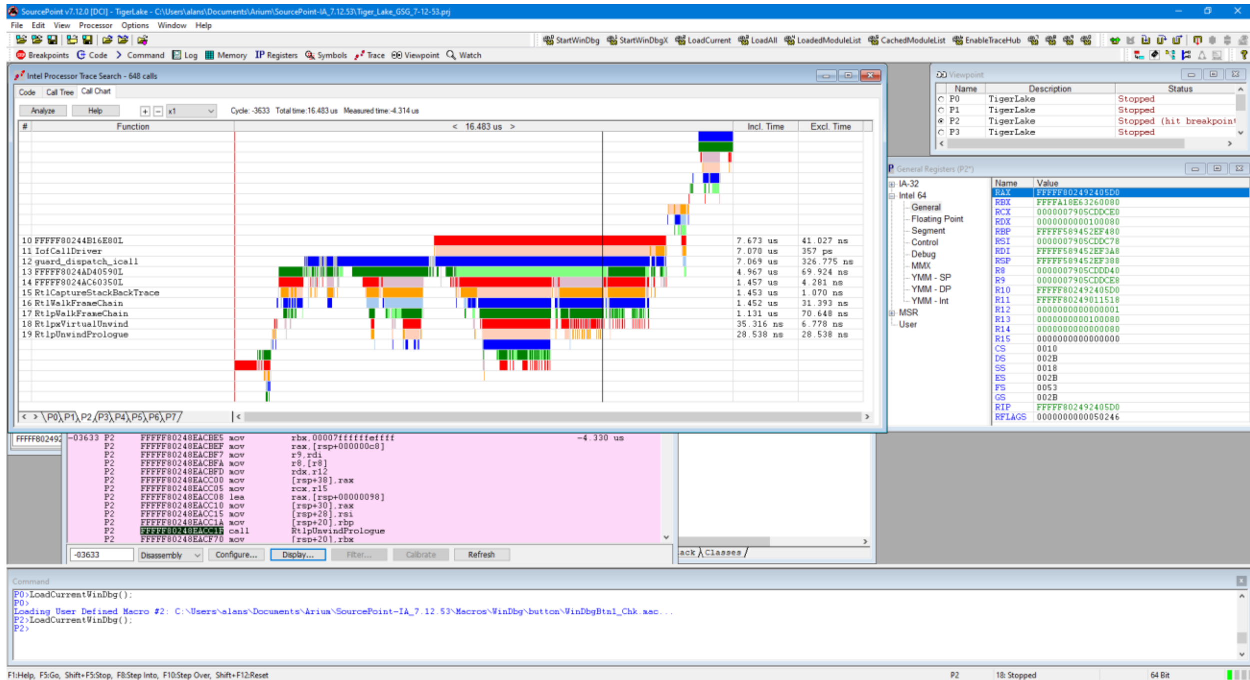
Feel free to resize the Intel Processor Trace window, and make it Floating, to see the trace data.

Click on the `Display` button within the Intel PT window, and be sure to click the appropriate buttons to ensure you see the symbols. These would include Object Code, Symbols, Pseudo-ops, Instruction Lines, Data Lines, and Labels Lines in the Disassembly section.

You can click the cursor at any code line within the Intel Processor Trace window, and right-click to open up a Tracking Code window that shows you the code and symbols (if available) for that line of code. You'll see the below when you open up the Tracking Code window at an arbitrary line of the traceback:



To see a visual display of the trace data, right-click within the Intel Processor Trace window, click on `Trace Search…`, click on the `Call Chart` tab, and hit `Analyze`. You'll see something like this:



Move the time arrow by clicking on a section of code, or use your arrow keys. Expand the view of a particular area of code with the mouse wheel, or using the Expand (starts at x1) drop-down or +/- buttons at the top.

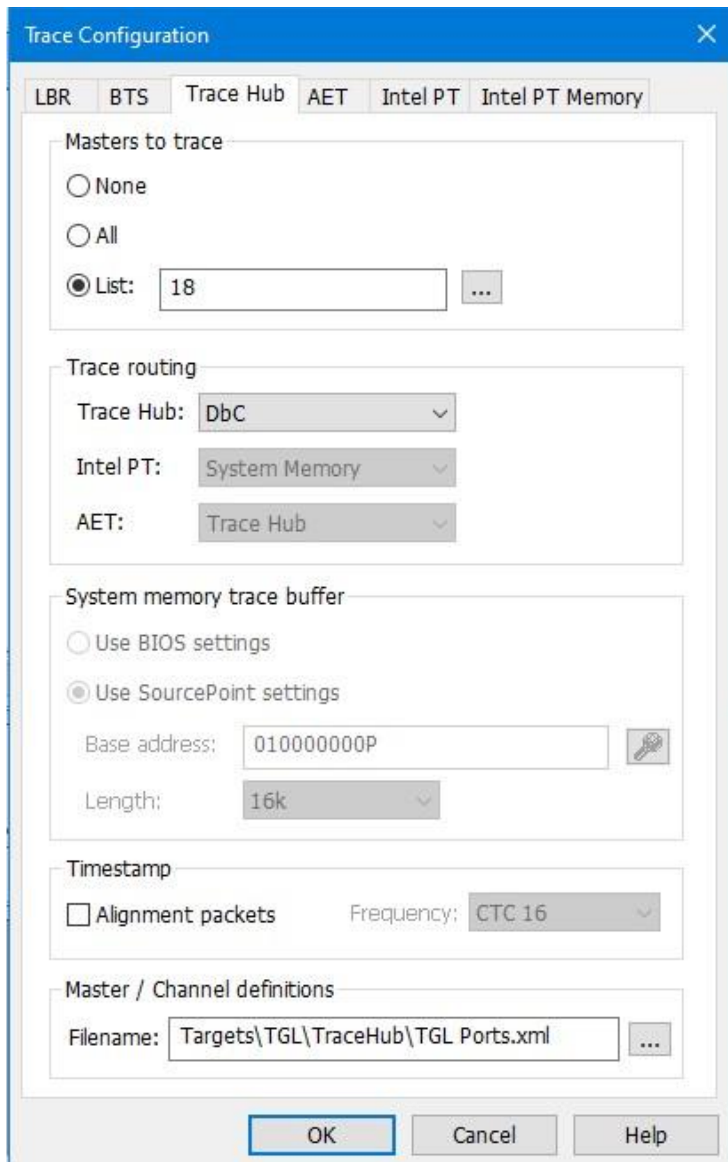
Event Trace

First Step: Configuring the Intel Trace Hub

Event tracing on the TGL platform is accomplished by the Intel Trace Hub. Fortunately, using DCI, events supported by the Intel Trace Hub can be streamed directly out of the system, well before Windows boots, with no need for system memory to be available.

Boot to the UEFI shell. This is accomplished by powering on the target, and pressing the F7 key until you come to the password entry screen. Note the Power Tip above that references the newer Celeron boards, and the workaround necessary to get the target to power up.

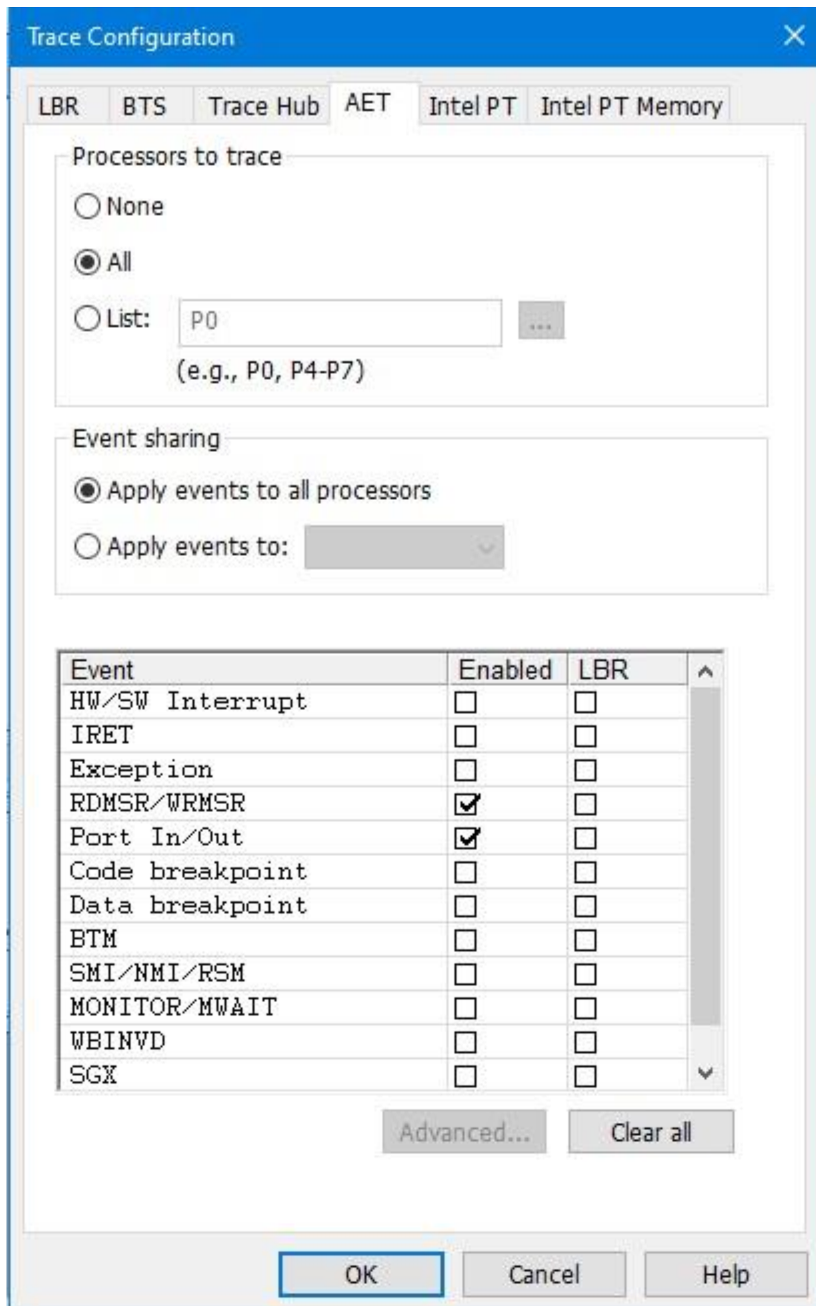
Click on the `Trace` button in the toolbar at the top, to open the Trace window; then click on the `Configure…` button; then click on the `Trace Hub` tab. Set the settings as below:



Second Step: Set up Architectural Event Trace

Now, it's time to tell the Trace Hub what you want to trace.

Once the Trace Hub has been enabled for the features you need, click on the `AET` tab, select `All` as `Processors to trace`, and select `RDMSR/WRMSR` and `Port In/Out` as events to trace:



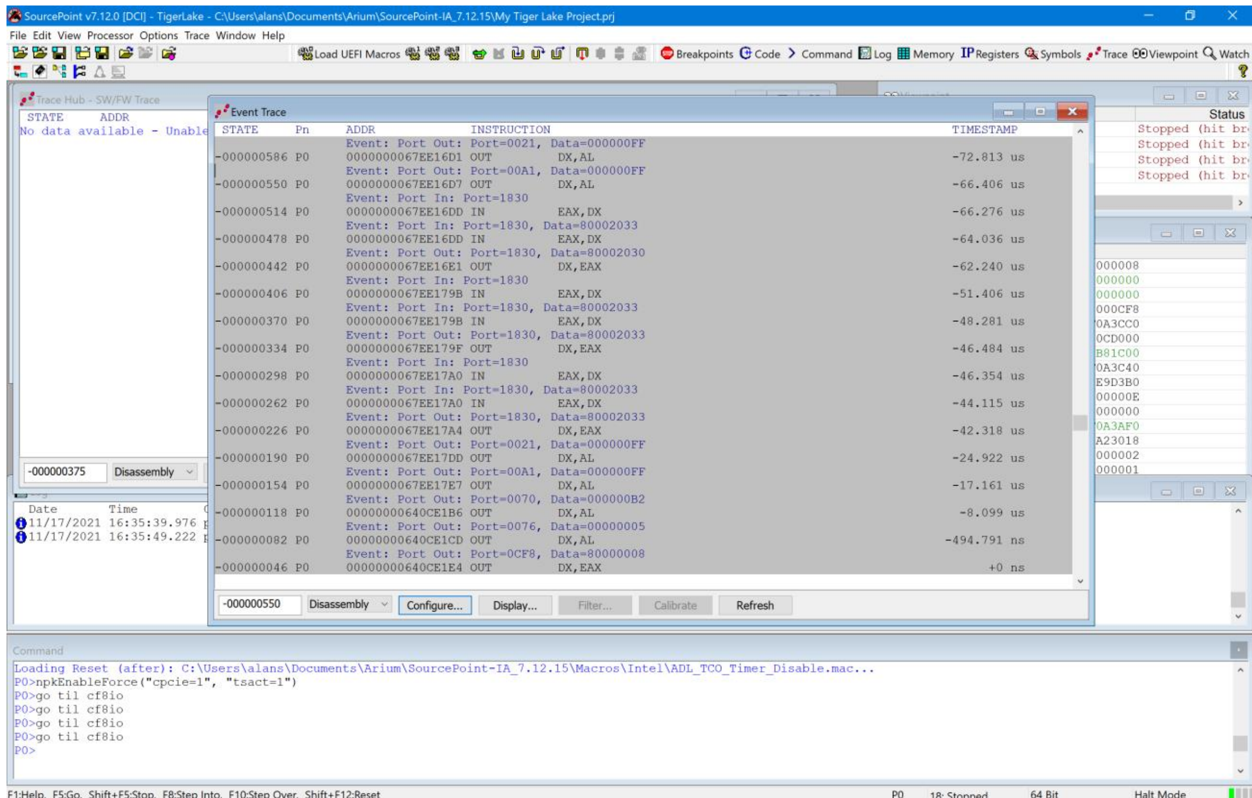
Now, you can simply do a Go/Stop to capture the event trace data. Below shows the use the `Command` window to simulate a break on any read/write of, say, port x'CF8', the

PCI CONFIG_ADDRESS. This is conveniently done by issuing at the `Command` window `P0>` prompt:

```
go til cf8io
```

This will run the target until the next IN or OUT to CF8.

After issuing the command, you'll see something like this:



Scrolling up a little, you'll see a mix of `Port In/Out` and `RDMSR/WRMSR`. All timestamped.

Power tip: The Last Branch Record (LBR) stack associated with each event can be captured as well. This is a very powerful debugging utility, especially when troubleshooting code execution leading up to events before system memory is initialized and Intel Processor Trace is available.

©2024 ASSET InterTech, Inc.

Trace Configuration

LBR | BTS | Trace Hub | AET | Intel PT | Intel PT Memory

Processors to trace

- ( ) None
- ( ) All
- (•) List: p0 ...

(e.g., P0, P4-P7)

Event sharing

- (•) Apply events to all processors
- ( ) Apply events to:

Event	Enabled	LBR
HW/SW Interrupt	☐	☐
IRET	☐	☐
Exception	☐	☐
RDMSR/WRMSR	☑	☑
Port In/Out	☑	☑
Code breakpoint	☐	☐
Data breakpoint	☐	☐
BTM	☐	☐
SMI/NMI/RSM	☐	☐
MONITOR/MWAIT	☐	☐
WBINVD	☐	☐
SGX	☐	☐

Advanced... | Clear all

OK | Cancel | Help

©2024 ASSET InterTech, Inc.

Getting Started with Hyper-V/VBS Debug

Debugging with Hyper-V and Virtualization-Based Security (VBS) is supported in the SourcePoint 7.12.53 release.

In SourcePoint 7.12.53, the main features introduced for HV/Secure Kernel debug are:

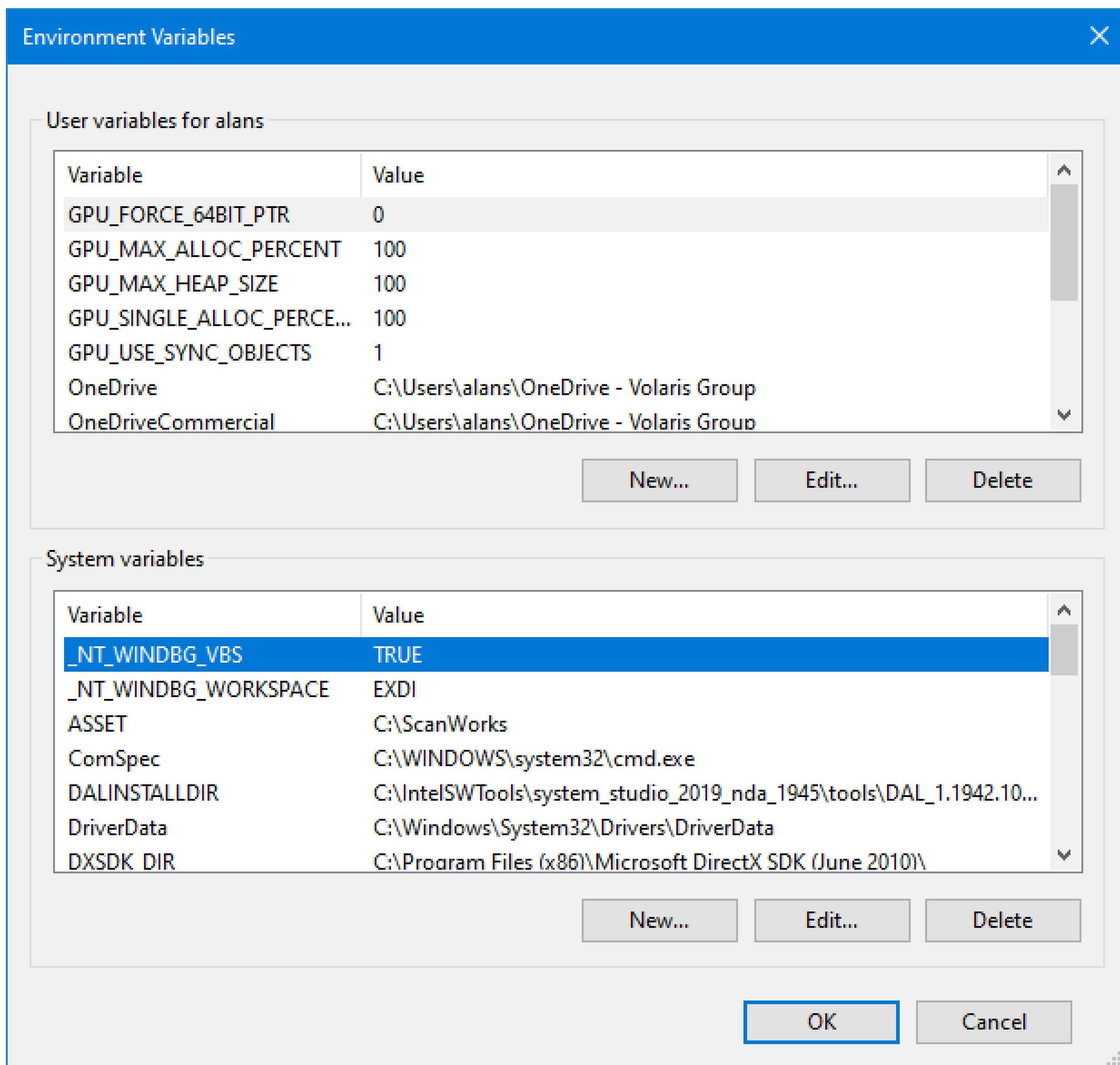
- VM Launch, VM Resume, VM Exit breakpoints
- vmcs macro (usable in Host mode) with functions:
    - vmread(encoding) // to read a specific VMCS field
    - vmwrite(encoding, value) // to write a specific value to a VMCS field
    - dump() // dumps the VMCS
    - reason() // displays the VM Exit reason
    - ipt() // turns off “conceal” bits and enables Intel PT

Power Tip: You must be in Host mode to use the vmcs macro functions.

Within SourcePoint 7.12.54, the vmcs macro (which uses in-line assembly) will be deprecated, and direct probe mode-based access to the VMCS in both Host and Guest mode will be provided.

Let’s get started. We’ll walk through a subset of the capabilities. There are a multitude of areas to explore here.

Firstly, change the Environment Variable _NT_WINDBG_VBS to TRUE, so that SourcePoint doesn’t go scanning for the KdVersionBlock (it can’t find it in a hypervisor-enabled target anyways):



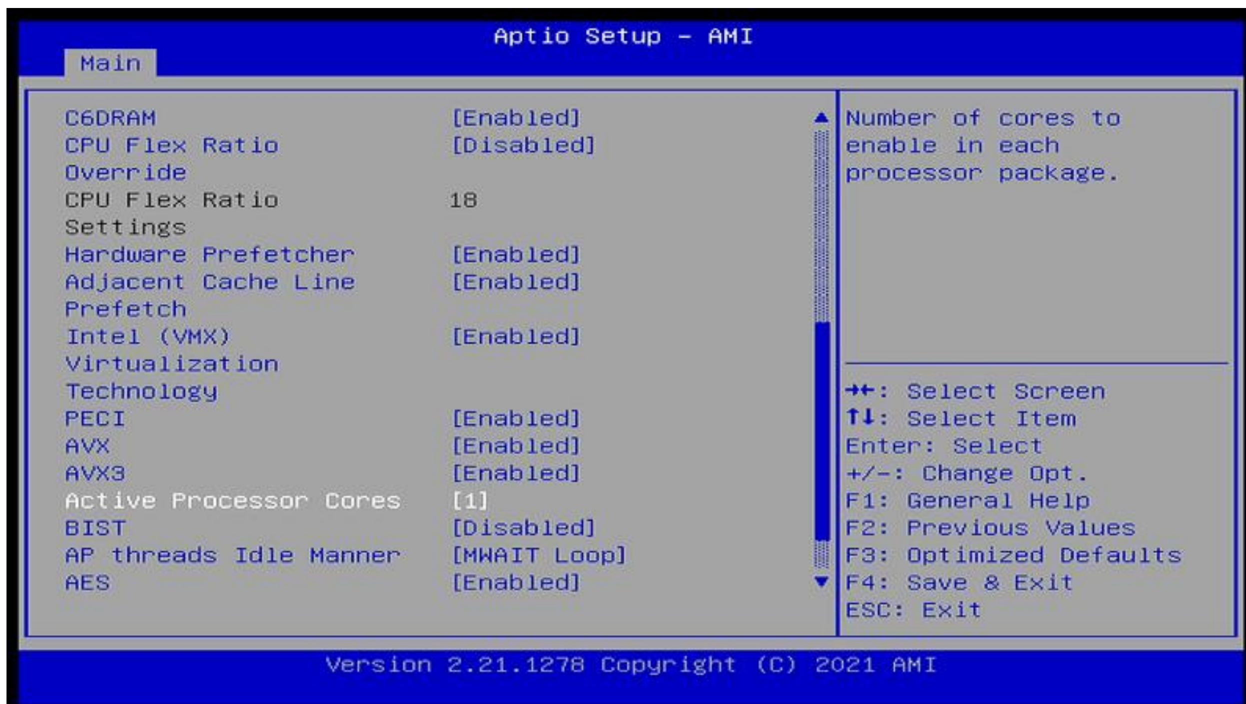
It is so much easier to see VM transitions if only one processor is made active on the target. There are several ways to do this. One is to issue this command on the target, from an Administrator CMD prompt:

```
>bcdedit /set numprocs 1
```

And then reset the target.

Another means is via the advanced BIOS settings (with the upassw0rd password), go to CRB Setup > CRB Advanced > CPU Configuration, disable Hyper-Threading (if you're

on a target that supports it – the Celeron board does not), and set Active Processor Cores to 1:



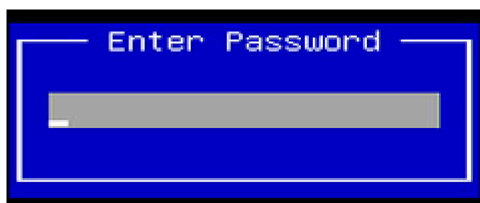
Disable the synthetic watchdog on the target to ensure that the target does not autonomously reset itself in probe mode:

```
>bcdedit /set {default} loadoptions "systemwatchdogpolicy=disabled"
```

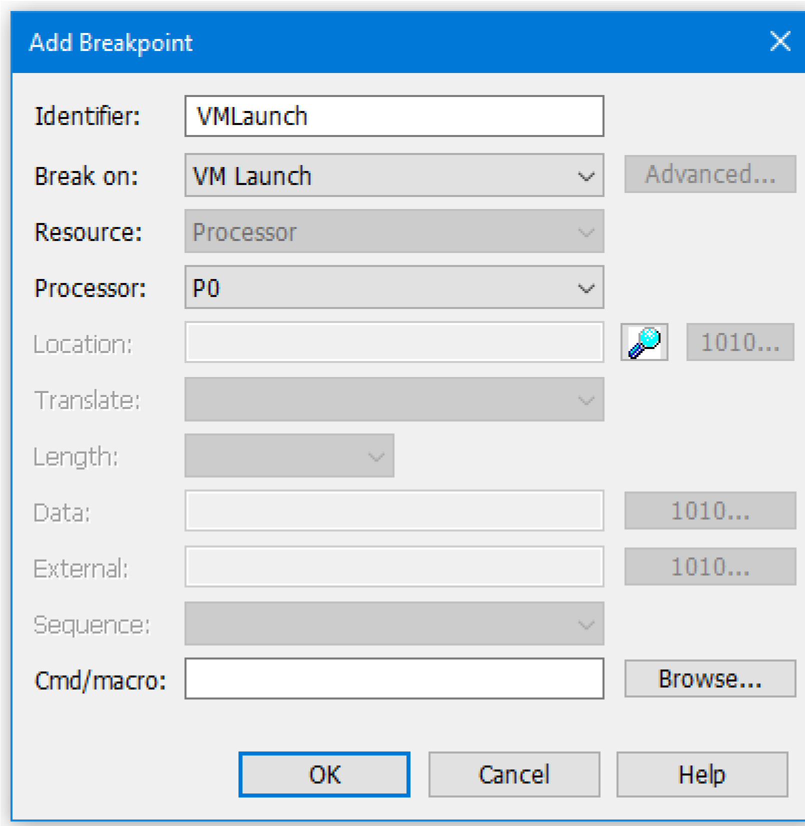
Power Tip: After crashing the target, which you will do periodically when you go “off the fairway” with VMM debug, you will need to power-cycle, and Windows will launch Automatic Repair to attempt to restore itself. Then you have to Restart again. There is no need for this, so to save time, you can disable recovery boot from an Administrator CMD window:

```
>bcdedit /set recoveryenabled No >NUL
>bcdedit /set bootstatuspolicy ignoreallfailures >NUL
```

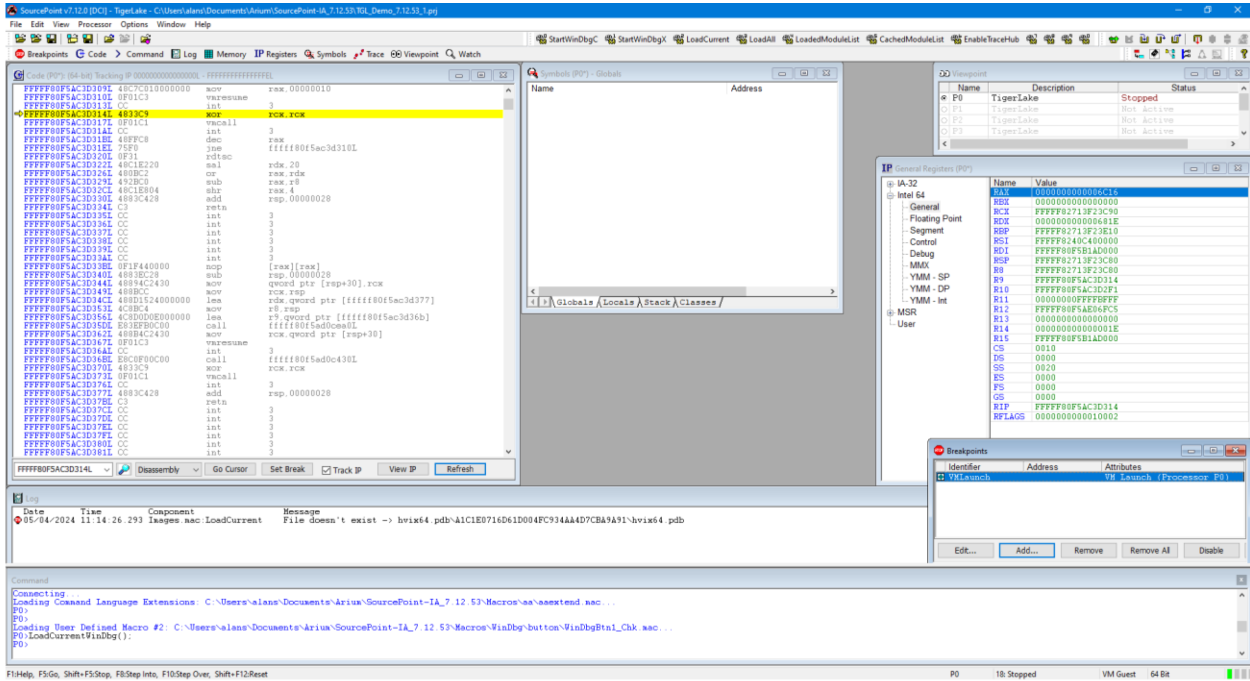
Boot the target to the UEFI shell. This is accomplished by power-cycling the target and holding down the F7 key until you see the BIOS login prompt:



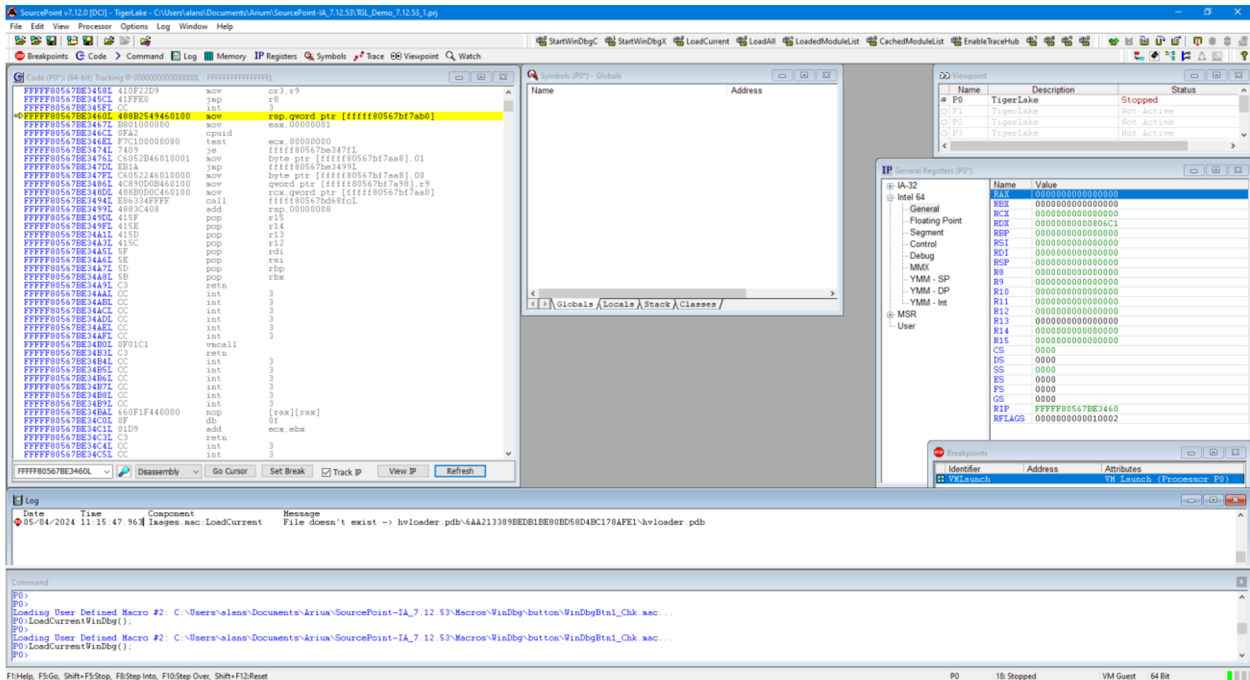
Launch SourcePoint, connect to the target, issue a Stop, Refresh the Code window, and set a VM Launch breakpoint:



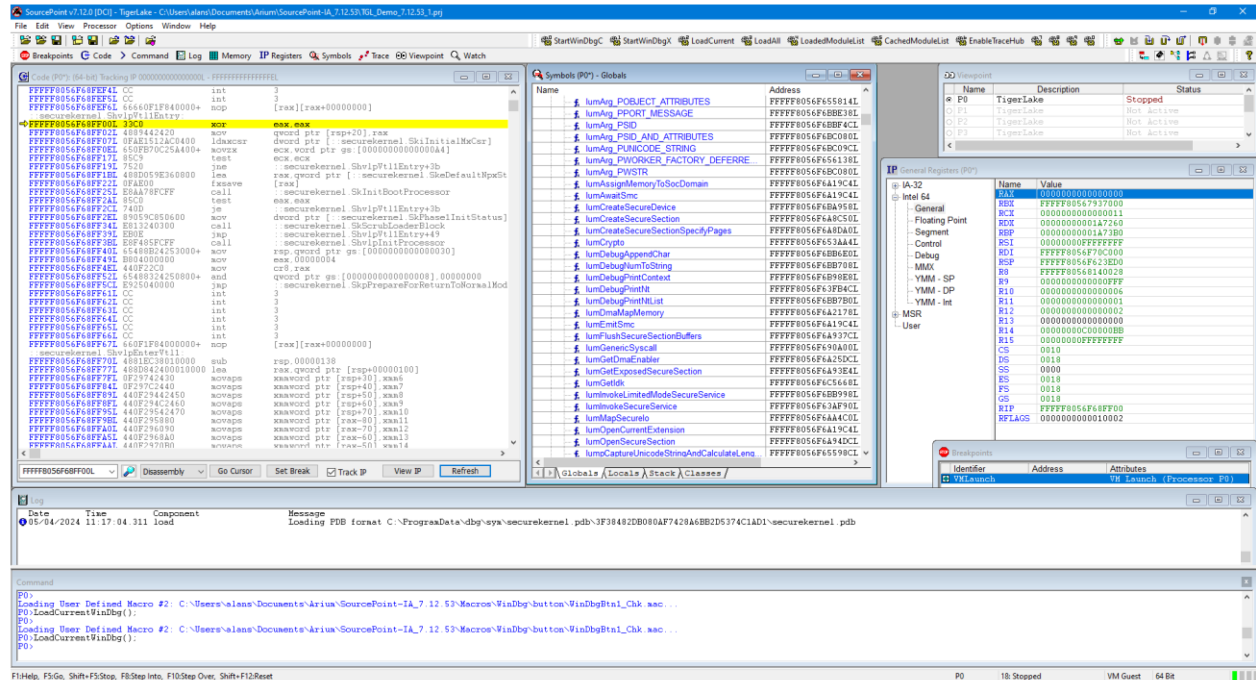
Hit Go. Be sure to hit Enter a couple of times to start the Windows boot process. You will break at the first VM launch, and land in the hypervisor, in VM Guest mode. Click on the LoadCurrent macro button, and you will see we're in hvix64, for which there are no symbols:



Hit Go, and then LoadCurrent a second time. You're in hvloader, again in VM Guest mode (not surprisingly, since we used a VM Launch breakpoint):



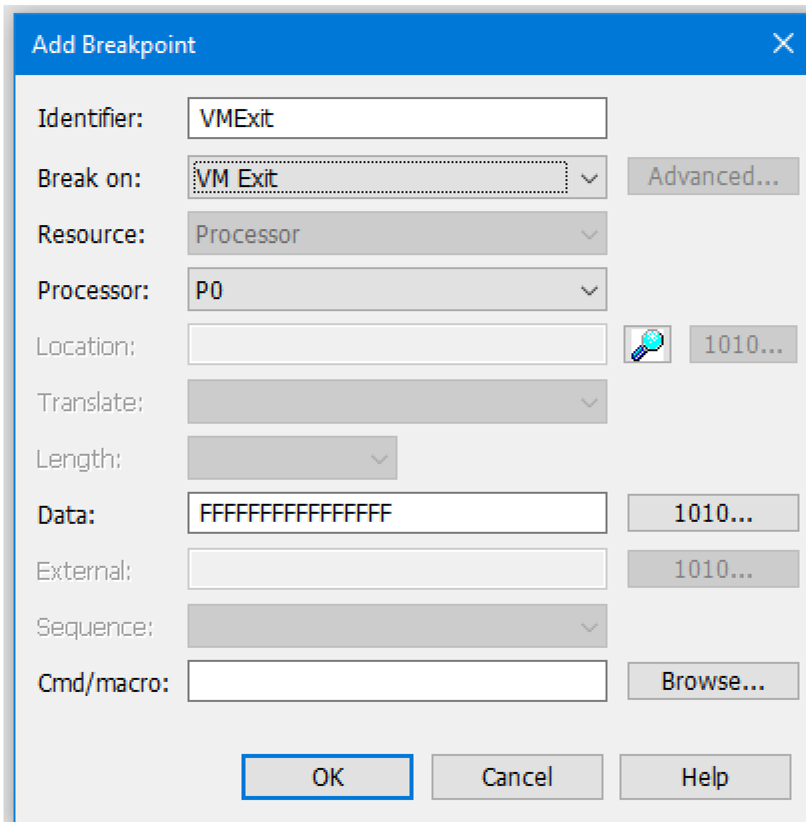
Hit Go a third time, encountering the third VM Launch breakpoint, with the target again in Guest mode. Hit LoadCurrent again, and provided you have the securekernel.pdb file cached, you will see the symbols for the Secure Kernel:



All of the Secure Kernel functions are available for debug. Enjoy!

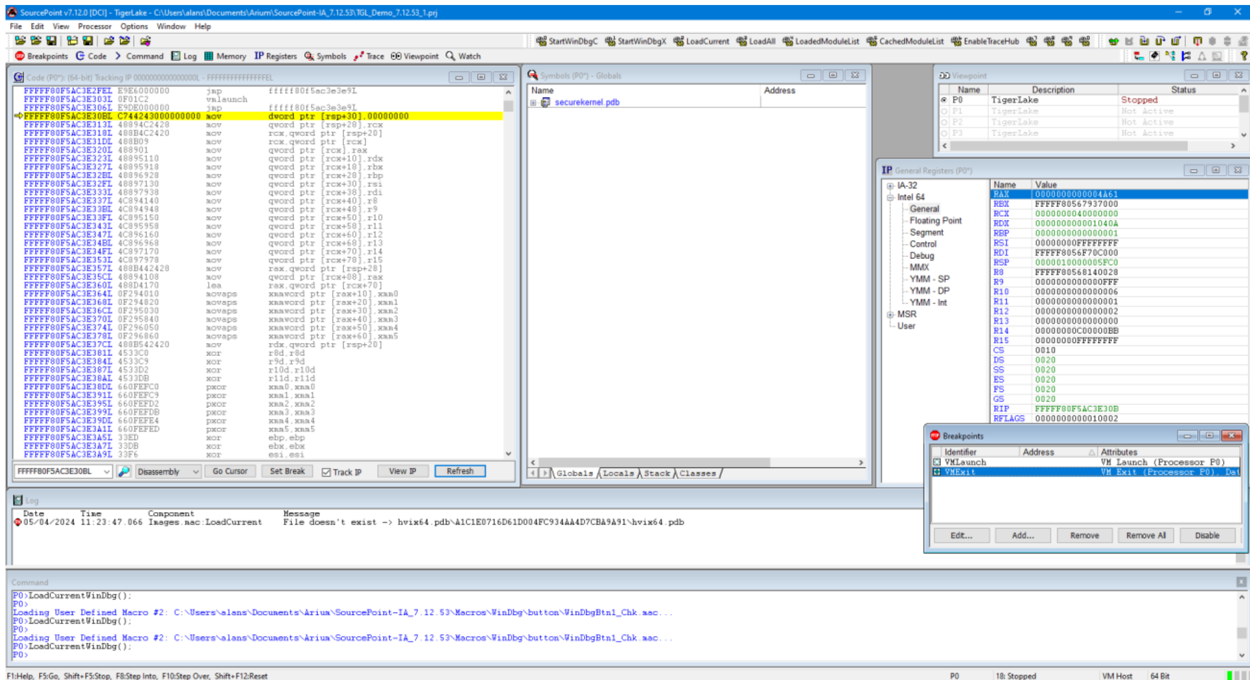
For a more advanced topic, let's look at using VM Exit breakpoints to capture Guest to Host transitions, and use Intel PT to see code flow for dynamic analysis.

Turn off the VM Launch breakpoint, and add a VM Exit breakpoint:



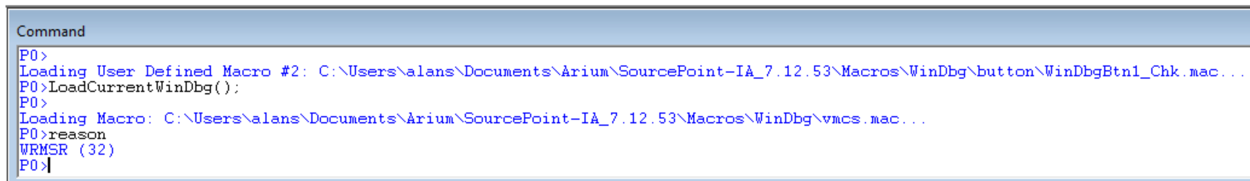
Clicking on the 1010… to the right of Data, shows that we can trigger on any single or combination of VM Exit reasons, as detailed in the Intel Software Developer's Manual, Volume 3A, Appendix C, VMX Basic Exit Reasons. For now, let's just leave them as all F's.

This puts us back into hvix64, and this time we're in VM Host mode:



Now, let’s load the vmcs macro. Go to the File > Macro > Load Macro… and select the vmcs.mac, and hit Open.

Type the `reason` command in the Command window:



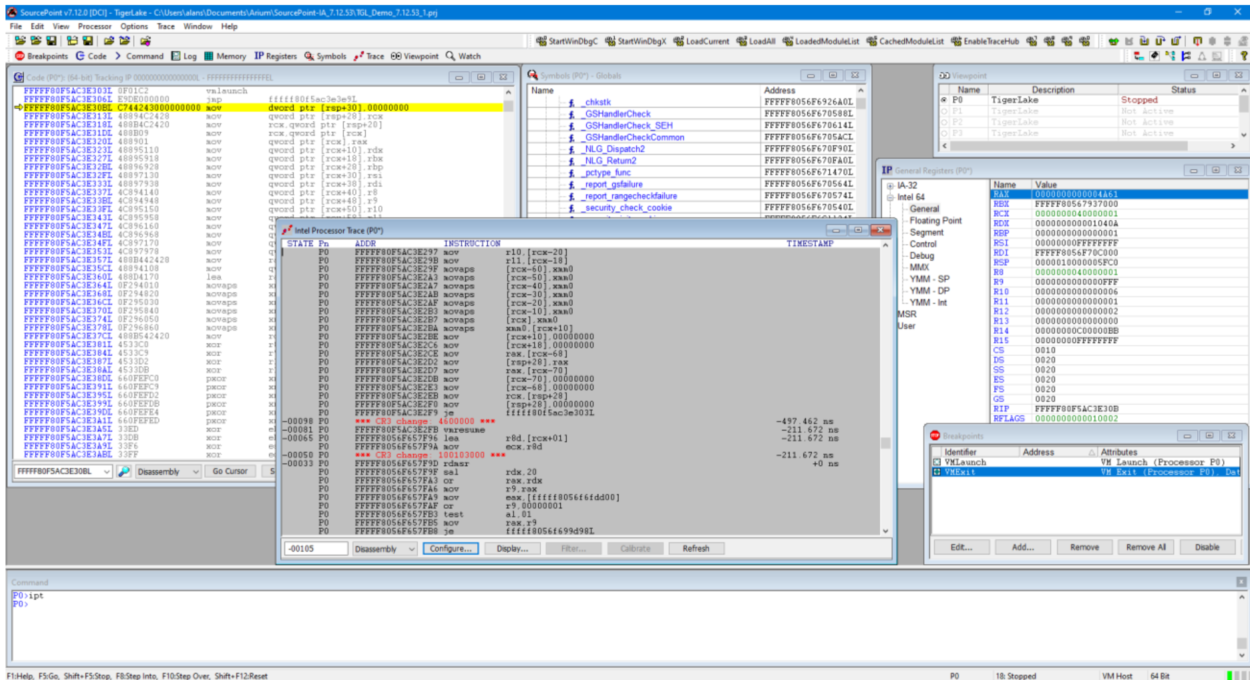
In this instance you’ll see the VM Exit Reason was a WRMSR (Basic Exit Reason #32).

Use the dump command to look at select fields within the VMCS, with one example below:

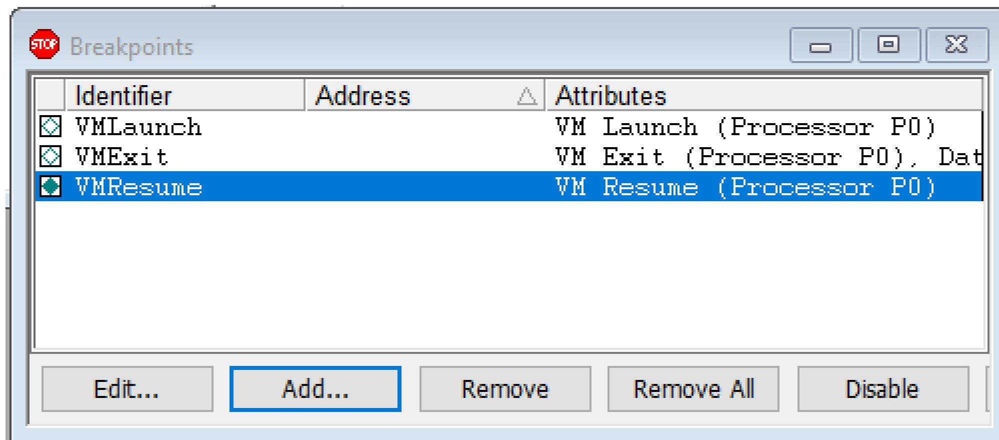
```
P0>dump
Guest-state:
  RIP: FFFFF8056F657F94
  CR3: 0000000004600000
  IA32_DEBUG_CTL: 0000000000000000
  IA32_RTIT_CTL: 0000000000000000
  IA32_LBR_CTL: FFFFF80567937000
Host-state:
```

```
  Exception bitmap: 00060002
  I/O bitmap (0000-7fff) address: 0000000101403000
  I/O bitmap (8000-ffff) address: 0000000101404000
  MSR bitmap address: 000000010DC45000
  EPT pointer: 00000001102EF01E
  VPID: 0002
VM-execution:
  Pin-based: 0000003F
    B0: External-interrupt exiting: TRUE
  Processor-based primary: B6A06DFA
    B23: Move DR causes VM-exit: TRUE
    B24: Unconditional I/O exiting: FALSE
    B25: Use I/O bitmaps: TRUE
    B27: Monitor trap flag: FALSE
    B28: Use MSR bitmaps: TRUE
  Processor-based secondary: 001813AB
    B01: EPT enabled: TRUE
    B05: VPID enabled: TRUE
    B14: VMCS Shadowing: FALSE
    B19: Hide NR bit in Intel PT PIPs: TRUE
    B24: Intel PT uses Guest physical: FALSE
VM-entry:
  Primary: 000213FF
    B02: Load IA32_DEBUGCTL: TRUE
    B17: Conceal VM-entry from Intel PT: TRUE
    B18: Load IA32_RTIT_CTL: FALSE
    B21: Load Guest IA32_LBR_CTL: FALSE
  MSR load count: 00000000
VM-exit:
  Primary: 0103EFFF
    B02: Save IA32_DEBUGCTL: TRUE
    B24: Conceal VM-exit from Intel PT: TRUE
    B25: Clear IA32_RTIT_CTL: FALSE
    B26: Clear IA32_LBR_CTL: FALSE
  Secondary: 67937000
  MSR store count: 00000000
  MSR load count: 00000000
```

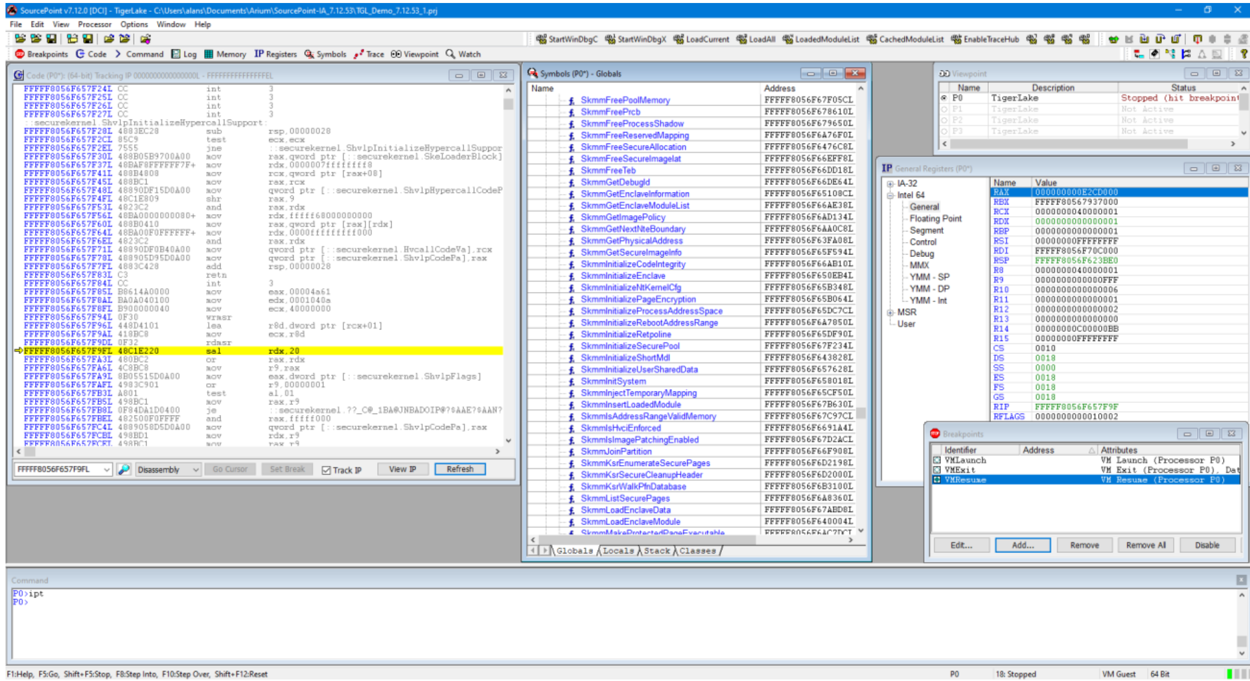
To use Intel PT, use the `ipt` command from the SourcePoint Command window, set up Intel PT as you normally would, and hit Go to break at the next VM Exit.



To trace within the Secure Kernel, with symbols, uncheck the VM Exit breakpoint, and turn off Intel PT **(there's still a bug in tracing VM Resume transitions)** and add a VM Resume breakpoint:



Hit Go, and you'll land back in the Secure Kernel:



Then set Intel PT back up again, uncheck the VM Resume breakpoint, set a VM Exit breakpoint, and we'll see the code traced as we go from Guest mode back to Host mode:

```
Intel Processor Trace (P0*)
STATE Pn    ADDR              INSTRUCTION                                                      TIMESTAMP
-00154 P0   *** VMCS pointer change: 10DC43 ***
            *** Trace enabled ***
-00081 P0   FFFFF8056F657F9F sal      rdx,20                                                   -506.477 ns
       P0   FFFFF8056F657FA3 or       rax,rdx
       P0   FFFFF8056F657FA6 mov      r9,rax
       P0   FFFFF8056F657FA9 mov      eax,[fffff8056f6fdd00]
       P0   FFFFF8056F657FAF or       r9,00000001
       P0   FFFFF8056F657FB3 test     al,01
       P0   FFFFF8056F657FB5 mov      rax,r9
       P0   FFFFF8056F657FB8 je       ::securekernel.??_C@_1BA@JNBADOIP@?$AAE?$AAN?$AAC?$AAL?$AAA?$AAV?$AAE@FNODOBFM@+3e58
-00070 P0   FFFFF8056F657F9F sal      rdx,20                                                   -215.010 ns
       P0   FFFFF8056F657FA3 or       rax,rdx
       P0   FFFFF8056F657FA6 mov      r9,rax
       P0   FFFFF8056F657FA9 mov      eax,[fffff8056f6fdd00]
       P0   FFFFF8056F657FAF or       r9,00000001
       P0   FFFFF8056F657FB3 test     al,01
       P0   FFFFF8056F657FB5 mov      rax,r9
       P0   FFFFF8056F657FB8 je       ::securekernel.??_C@_1BA@JNBADOIP@?$AAE?$AAN?$AAC?$AAL?$AAA?$AAV?$AAE@FNODOBFM@+3e58
-00065 P0   FFFFF8056F657FBE and      rax,fffff000                                             -215.010 ns
       P0   FFFFF8056F657FC4 mov      [fffff8056f6fdd58],rax
       P0   FFFFF8056F657FCB mov      rdx,r9
       P0   FFFFF8056F657FCE mov      rax,r9
       P0   FFFFF8056F657FD1 shr      rdx,20
       P0   FFFFF8056F657FD5 mov      ecx,r8d
-00050 P0   *** CR3 change: 100103000 ***                                                      -215.010 ns
-00033 P0   FFFFF8056F657FD8 wrmsr                                                                   +0 ns
       P0   FFFFF8056F657FDA mov      rcx,[fffff8056f6fdd40]
       P0   FFFFF8056F657FE1 mov      r8d,00000011
       P0   FFFFF8056F657FE7 shr      r9,c
       P0   FFFFF8056F657FEB mov      rdx,r9
       P0   FFFFF8056F657FEE call     fffff8056f65923cL
       P0   FFFFF8056F65923C mov      r11,rcx
       P0   FFFFF8056F65923F mov      eax,r8d
       P0   FFFFF8056F659242 and      al,02
       P0   FFFFF8056F659244 mov      r10d,r8d
       P0   FFFFF8056F659247 neg      al
       P0   FFFFF8056F659249 sbb      r9d,r9d
       P0   FFFFF8056F65924C and      r9d,00000003
       P0   FFFFF8056F659250 inc      r9d
       P0   FFFFF8056F659253 mov      ecx,r9d
       P0   FFFFF8056F659256 or       ecx,00000002
       P0   FFFFF8056F659259 and      r10d,00000010
       P0   FFFFF8056F65925D cmove    ecx,r9d
       P0   FFFFF8056F659261 mov      r9,0000000ffffffff
       P0   FFFFF8056F65926B mov      eax,ecx
       P0   FFFFF8056F65926D or       eax,00000010
       P0   FFFFF8056F659270 and      r8d,00000020
       P0   FFFFF8056F659274 cmove    eax,ecx
       P0   FFFFF8056F659277 and      rdx,r9
       P0   FFFFF8056F65927A sal      rdx,4
       P0   FFFFF8056F65927E lea      rcx,[fffff8056f6dab90]
       P0   FFFFF8056F659285 mov      rcx,[rcx][rax*8]
       P0   FFFFF8056F659289 mov      rax,ffff000000000eff
       P0   FFFFF8056F659293 and      rcx,rax
       P0   FFFFF8056F659296 movzx    ax,[fffff8056f6fda70]
       P0   FFFFF8056F65929D and      eax,00000001
       P0   FFFFF8056F6592A0 or       rax,rdx
       P0   FFFFF8056F6592A3 mov      rdx,7fffffffffffffff
       P0   FFFFF8056F6592AD sal      rax,8
       P0   FFFFF8056F6592B1 or       rcx,rax
       P0   FFFFF8056F6592B4 mov      rax,rcx
       P0   FFFFF8056F6592B7 and      rax,rdx
       P0   FFFFF8056F6592BA test     r10d,r10d
       P0   FFFFF8056F6592BD cmove    rax,rcx

-00033   Disassembly   Configure...   Display...   Filter...   Calibrate   Refresh
```

And yes, we need to demangle some of these function calls. It's on the to-do list.

One last SourcePoint trick: when in the Secure Kernel, with the conceal bits turned off with the `ipt` command, turn Intel PT back on, turn off the Label lines in the Display… Trace Display Settings, but this time use the Append radio button within the Intel PT Memory tab, and single-step away:

ASSET™

Intel Processor Trace (P0*)

```
STATE  Pn     ADDR              INSTRUCTION
-03025 P0     FFFFF8056F657FEB  mov          rdx,r9
-02897 P0     FFFFF8056F657FEE  call         ::securekernel.SkmmMapBootPage
-02769 P0     FFFFF8056F65923C  mov          r11,rcx
-02641 P0     FFFFF8056F65923F  mov          eax,r8d
-02513 P0     FFFFF8056F659242  and          al,02
-02385 P0     FFFFF8056F659244  mov          r10d,r8d
-02257 P0     FFFFF8056F659247  neg          al
-02129 P0     FFFFF8056F659249  sbb          r9d,r9d
-02001 P0     FFFFF8056F65924C  and          r9d,00000003
       P0     FFFFF8056F659250  inc          r9d
       P0     FFFFF8056F659253  mov          ecx,r9d
       P0     FFFFF8056F659256  or           ecx,00000002
       P0     FFFFF8056F659259  and          r10d,00000010
       P0     FFFFF8056F65925D  cmove        ecx,r9d
       P0     FFFFF8056F659261  mov          r9,0000000ffffffff
       P0     FFFFF8056F65926B  mov          eax,ecx
       P0     FFFFF8056F65926D  or           eax,00000010
       P0     FFFFF8056F659270  and          r8d,00000020
       P0     FFFFF8056F659274  cmove        eax,ecx
       P0     FFFFF8056F659277  and          rdx,r9
       P0     FFFFF8056F65927A  sal          rdx,4
       P0     FFFFF8056F65927E  lea          rcx,[::securekernel.SkmiProtectionToPte]
       P0     FFFFF8056F659285  mov          rcx,[rcx][rax*8]
       P0     FFFFF8056F659289  mov          rax,ffff000000000eff
       P0     FFFFF8056F659293  and          rcx,rax
       P0     FFFFF8056F659296  movzx        ax,[::securekernel.SkmiState]
       P0     FFFFF8056F65929D  and          eax,00000001
       P0     FFFFF8056F6592A0  or           rax,rdx
       P0     FFFFF8056F6592A3  mov          rdx,7fffffffffffff
       P0     FFFFF8056F6592AD  sal          rax,8
       P0     FFFFF8056F6592B1  or           rcx,rax
       P0     FFFFF8056F6592B4  mov          rax,rcx
       P0     FFFFF8056F6592B7  and          rax,rdx
       P0     FFFFF8056F6592BA  test         r10d,r10d
       P0     FFFFF8056F6592BD  cmove        rax,rcx
       P0     FFFFF8056F6592C1  mov          rcx,ffff68000000000
       P0     FFFFF8056F6592CB  mov          rdx,rax
       P0     FFFFF8056F6592CE  or           rdx,00000018
       P0     FFFFF8056F6592D2  test         r8d,r8d
       P0     FFFFF8056F6592D5  cmove        rdx,rax
       P0     FFFFF8056F6592D9  shr          r11,9
       P0     FFFFF8056F6592DD  mov          rax,0000007ffffffff8
       P0     FFFFF8056F6592E7  and          r11,rax
       P0     FFFFF8056F6592EA  mov          rax,fffff8482413000
       P0     FFFFF8056F6592F4  add          rcx,r11
       P0     FFFFF8056F6592F7  add          rax,r11
       P0     FFFFF8056F6592FA  mov          [rcx],rdx
       P0     FFFFF8056F6592FD  cmp          rax,000007ff
-01878 P0     FFFFF8056F659303  ja           ::securekernel.SkmmMapBootPage+f5
-01750 P0     FFFFF8056F659331  retn
-01601 P0     FFFFF8056F657FF3  mov          rax,[::securekernel.ShvlpHypercallCodePage]
-01473 P0     FFFFF8056F657FFA  mov          ecx,00000001
-01345 P0     FFFFF8056F657FFF  mov          [::securekernel.HvcallCodeVa],rax
-01217 P0     FFFFF8056F658006  call         ::securekernel.SkeFlushCurrentTb
-01089 P0     FFFFF8056F68EA70  test         ecx,ecx
-00961 P0     FFFFF8056F68EA72  mov          rax,cr4
-00838 P0     FFFFF8056F68EA75  je           ::securekernel.SkeFlushCurrentTb+f
-00705 P0     FFFFF8056F68EA77  test         rax,00000080
-00582 P0     FFFFF8056F68EA7D  jne          ::securekernel.SkeFlushCurrentTb+16
-00449 P0     FFFFF8056F68EA7F  mov          rax,cr3
-00297 P0     FFFFF8056F68EA82  mov          cr3,rax
-00166 P0     FFFFF8056F68EA85  retn
-00017 P0     FFFFF8056F65800B  jmp          ::securekernel.ShvlpInitializeHypercallSupport+57
```

-03025 | Disassembly | Configure... | Display... | Filter... | Calibrate | Refresh

Suggested reading for this section is as follows, with some tips below.

Part 1: JTAG debug of Windows Hyper-V / Secure Kernel with WinDbg and EXDI
This is a basic introduction to enabling HV/SK, and the use of the VM Launch and VM Exit breakpoints.

Part 2: JTAG debug of Windows Hyper-V / Secure Kernel with WinDbg and EXDI
One thing to note is that the symbols for the securekernel are in fact in the public domain, on the Microsoft symbol server. You need to ensure that these are in your cache folder for SourcePoint to see them.

Part 3: JTAG debug of Windows Hyper-V / Secure Kernel with WinDbg and EXDI
This blog covers symbolic debug of the Secure Kernel, with Intel Processor Trace. It highly recommends that the number of active processors is set to ‘1’, in order to easily distinguish transitions with the hypervisor, secure kernel, and NT OS.

Part 4: JTAG debug of Windows Hyper-V / Secure Kernel with WinDbg and EXDI
Under the SourcePoint File menu, click on Macro > Load Macro… and mouse over to C:\Users\<my computer>\Documents\Arium\SourcePoint-IA_7.12.52\Macros\WinDbg and select vmcs.mac. This makes the dump, vmread, vmwrite, reason and ipt commands available. The ipt() function is crucial to ensure that Intel Processor Trace works properly between Host ⇔ Guest transitions.

Part 5: JTAG debug of Windows Hyper-V / Secure Kernel with WinDbg and EXDI
This is a preamble article to using Intel AET to capture RDMSR and WRMSR events, and correlating them against the Windows MSR bitmap. For more advanced users only.

Troubleshooting Tips and Errata

Chances are, you'll run into something strange during your testing. We're the first to admit that JTAG-based run-control and trace are not always deterministic. JTAG is a 30-year hardware protocol, and when something goes astray at a very low level within the chip, SourcePoint tries to (but sometimes doesn't) recover gracefully. There will be times that the board will power cycle on its own. Or the firmware thinks that a thread is running but gets out of sync with the SourcePoint software, which thinks it's halted. Or the DbCStatus.exe ball stays red instead of turning green, while you swear you have a good DbC connection. Sometimes you have no choice but to quit SourcePoint and power cycle the target. That usually clears up the one-of's. But, of course, that means quitting out of WinDbg (preferably first), then quitting out of SourcePoint, power-cycling the target, and then re-establishing the connections from scratch. Tedious.

And, we all know that WinDbg has its quirks as well. And Windows sometimes objects to the presence of JTAG-assisted debuggers. Combine the three, and, well, you're bound to run into some bugs and misbehaviors.

Hopefully you don't run into this too many times. But, on the other hand, if you didn't, we'd have nothing to fix. 😊

In the meantime, here are errata for the UP Xtreme i11, and the steps needed to mitigate where possible.

Windows crashes

If you work with SourcePoint WinDbg long enough, you'll likely crash Windows at some point. Most of the time, Automatic Repair (presuming you have it on) will clean things up, but rarely it won't. In which case you will need to re-install Windows. Really, it's no different from reinstalling Windows in a VM, only more onerous.

Drop us a note on our Support line, or call us, if you can reproduce this.

WinDbg Classic is better than WinDbgX

WinDbgX, in intermittent circumstances, directs SourcePoint to do numerous memory reads at low memory. In which case, if you have the Log window open, will display messages like:

```
Page table is not present
```

```
Page table is not present. Linear address: 0000000000001800L
```

Most of the time, these error messages are just informative. But they do occur much more frequently with WinDbgX than WinDbg Classic.

Pause in Initial Symbol Load

Intermittently, after issuing the first Break in WinDbgX, in the middle of the memory reads associated with the symbol loading, WinDbg stops sending commands to SourcePoint, and the transactions stop. The SourcePoint Dashboard Lights stop flashing, and a look at the Log window shows no traffic.

This issue seems to be very host and target specific. On some, it does not occur at all. In others, we see more frequent failures.

The only option at this point is to quit out of WinDbg and SourcePoint, power cycle the target, and start over. It is currently under investigation.

This issue only manifests itself with WinDbgX. **WinDbg Classic does not have this issue.**

LoadCurrent versus LoadAll

The LoadCurrent macro makes the symbols available within the module at the current instruction pointer visible to SourcePoint. LoadAll will retrieve all symbols for what's in the addressable context. It takes a long time.

COM(32) Surrogate

After a crash, when you restart SourcePoint, once in a blue moon it will misbehave. Run-control will not work properly.

Open Task Manager, and look for a COM(32) Surrogate task. If you see one, kill it.

Viewing the Stack

SourcePoint's Stack display does not work. It's on the to-do list to fix. For now, use WinDbg's stack view.

LoadCurrent intermittently fails in User code

When hitting a breakpoint set in user code, about 50% of the time a LoadCurrent will not successfully display the symbols within the SourcePoint Code window. WinDbg correctly displays the symbols. If you have a SourcePoint Log window open, you may see:

```
File doesn't exist -> \000000000000000000000000000000000\
```

It’s not a critical issue, and we’re working on it.

Breaks are not process-aware

Setting breakpoints in WinDbg to break within a specific process, such as with:

```
bp /p <address> nt!NtReadFile explorer.exe
```

does not work properly. Instead of halting in the instance of nt!NtReadFile associated with explorer.exe, it will halt at the first instance of the shared code, likely in a different process. This is because EXDI does not provide process/thread information down to SourcePoint, unlike the standard WinDbg kdnet interface.

Mangled function names

You may sometimes see a mangled function name, as in:

```
JNE ??_C@_0BH@CBDMLJDN@RtlCreateUnicodeString@
```

SourcePoint does not have a built-in C++ name demangler. It’s on the to-do list.

WinDbg FP register display is not working

WinDbg does not display the floating point registers. SourcePoint displays the registers correctly.

Troubleshooting Tips on Hyper-V/ VBS Enabled Targets

VM Resume breakpoint with Intel PT crashes the target

When transitioning from Host to Guest mode, with Intel PT active, the reads from Guest to Host memory do not succeed. This will crash the target. You will have to quit out of SourcePoint, power-cycle the target, and start over. We are working on this.

Note that if you have not disabled Automatic Repair, any system crash will often require two power cycles of the target.

Hardware breakpoints don’t work well in the Secure Kernel

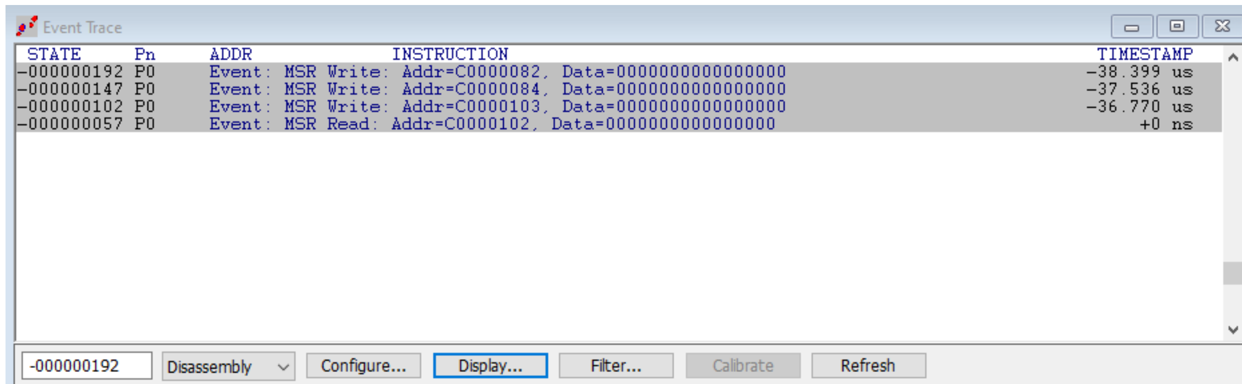
There are a few issues here, including:

(1) BP indicators in the Code view come and go, which occurs when the current CR3 differs from CR3 when the BP was set.
(2) BP set via WinDbg remains set in SourcePoint after the break.
(3) The SourcePoint cause command (which displays why a breakpoint was hit) does not work. The DR6 bit is not getting set to indicate why the BP was hit.

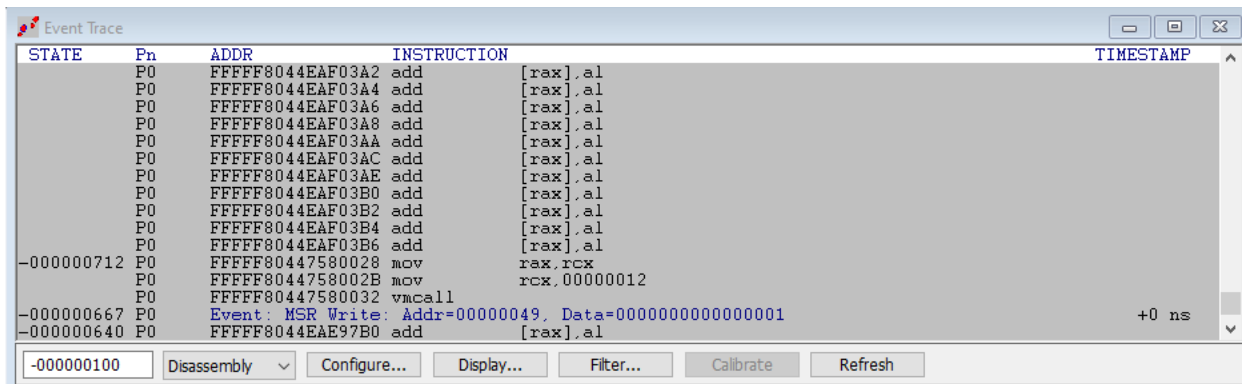
These are all to be fixed in the upcoming release.

AET only partially functional

Intel’s design for AET is only partially functional, with no knowledge of hypervisors and CR3 changes, unlike Intel PT. So, in some cases, you don’t see the actual disassembly in the Event trace window, but just the event itself:



Use LBR where applicable to perhaps get some meaningful code insight, keeping in mind that LBR is an old instruction trace technology, and just uses MSRs to track to/from addresses, so it is not CR3-aware either):



Support for VM Exit Reasons > 63

In the VM Exit breakpoint window, you can break on any single or multiple Basic Exit Reasons, from 0 to 63. As of the time of this writing, there are a total of 78 of them:

64 XRSTORS

65 PCONFIG

66 SPP-related event

67 UMWAIT

68 TPAUSE

69 LOADIWKEY

70 ENCLV

72 ENQCMD PASID translation failure

73 ENQCMDS PASID translation failure

74 Bus lock

75 Instruction timeout

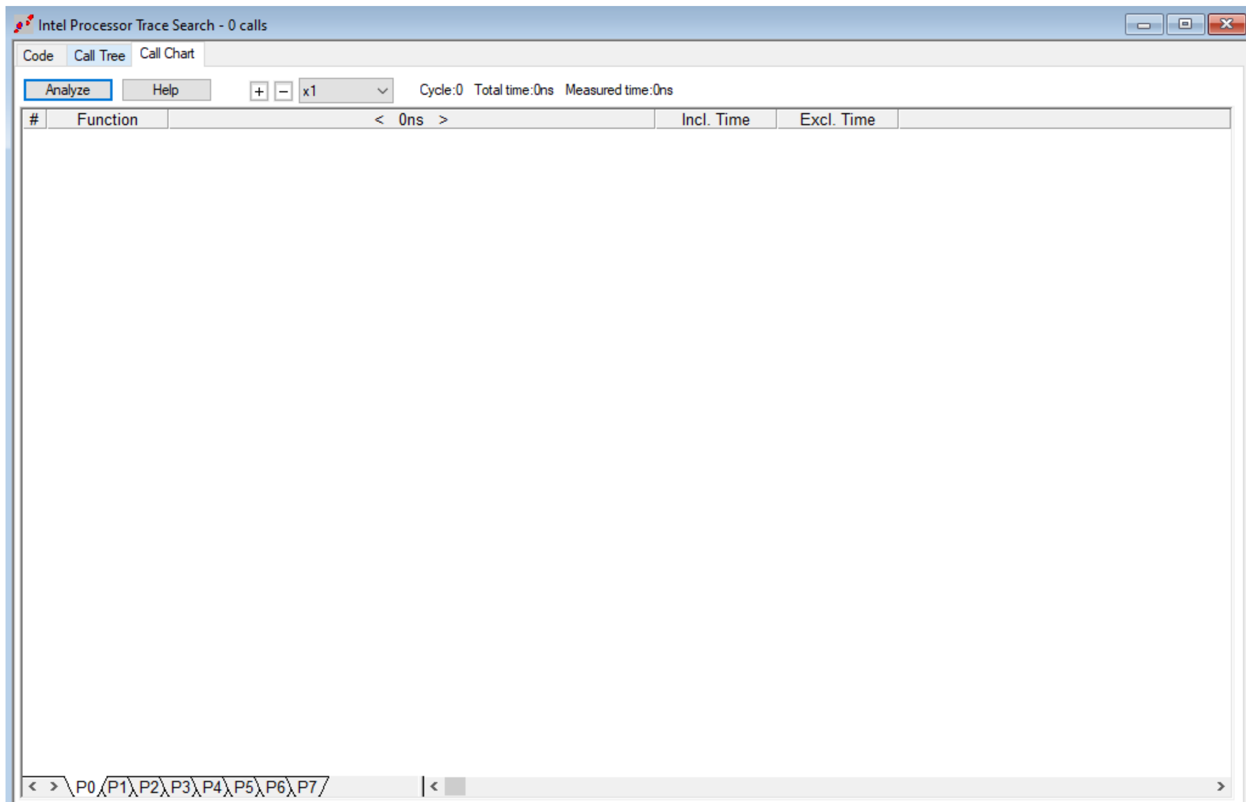
76 SEAMCALL

77 TDCALL

It's a bit of a kludge to include the exit reasons beyond 63, but we're working on it. It will be in the next release.

Intel PT Call Chart does not work reliably

When using Intel PT for tracing code, for example, from Guest to Host transitions, you won't get the Call Chart with the pretty colors to appear; pressing the Analyze button just yields a blank display:



Although this feature works well with Hyper-V disabled, as SourcePoint is “aware” of function entries and exits, this is much more complex with VMM behavior.

Conclusion

Thank you for getting this far! We hope that you have enjoyed the ride, and are using the power of SourcePoint WinDbg successfully in your debugging and learning journeys. There are many new things to discover in the Windows kernel enabled by this technology.

Feel free to browse the SourcePoint Academy at https://www.asset-intertech.com/sourcepoint-academy/ for helpful reference guides, help material and "how to" videos.

If you ever have any questions, please call, email or open a Support Case here: https://www.asset-intertech.com/support/. We'll be glad to help!