SourcePoint WinDbg

Getting Started Guide for the

AAEON UP Xtreme i11

Revision 1.1

Contents

Revision History

Revision Number	Description	Date
1.0	Original document.	December 6, 2023
1.1	Update for beta release SourcePoint 7.12.52. Support for HV/VBS debug.	March 31, 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 highly recommended background reading.

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:

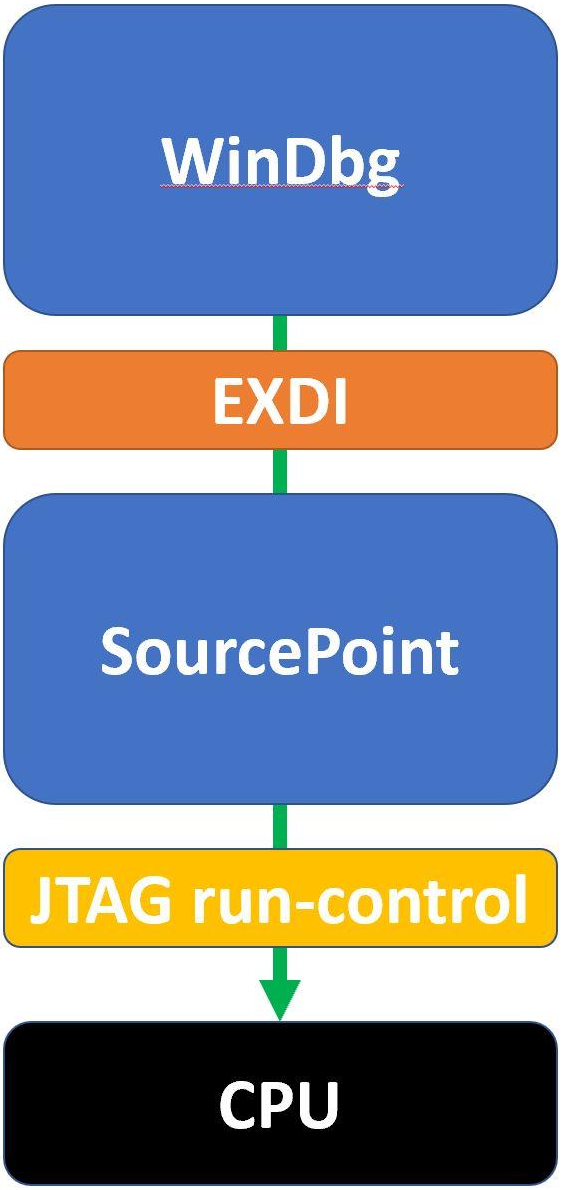
JTAG Debug using DCI on the AAEON UP Xtreme Whiskey Lake board

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 and Intel Trace, 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.

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.

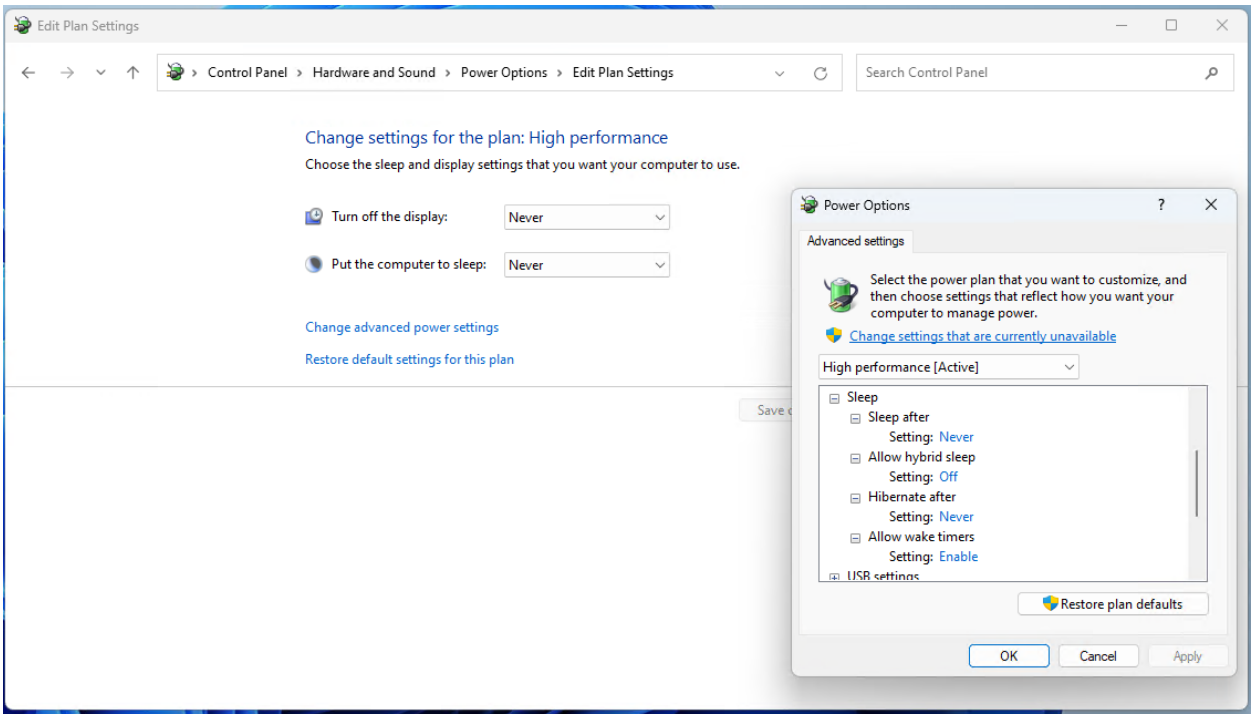
Configuring the target and setting up pre-requisites – Getting Started

Firstly, disable the UEFI TCO Watchdog timeout as described in the SourcePoint Getting Started Guide for the AAEON UP Xtreme i11.

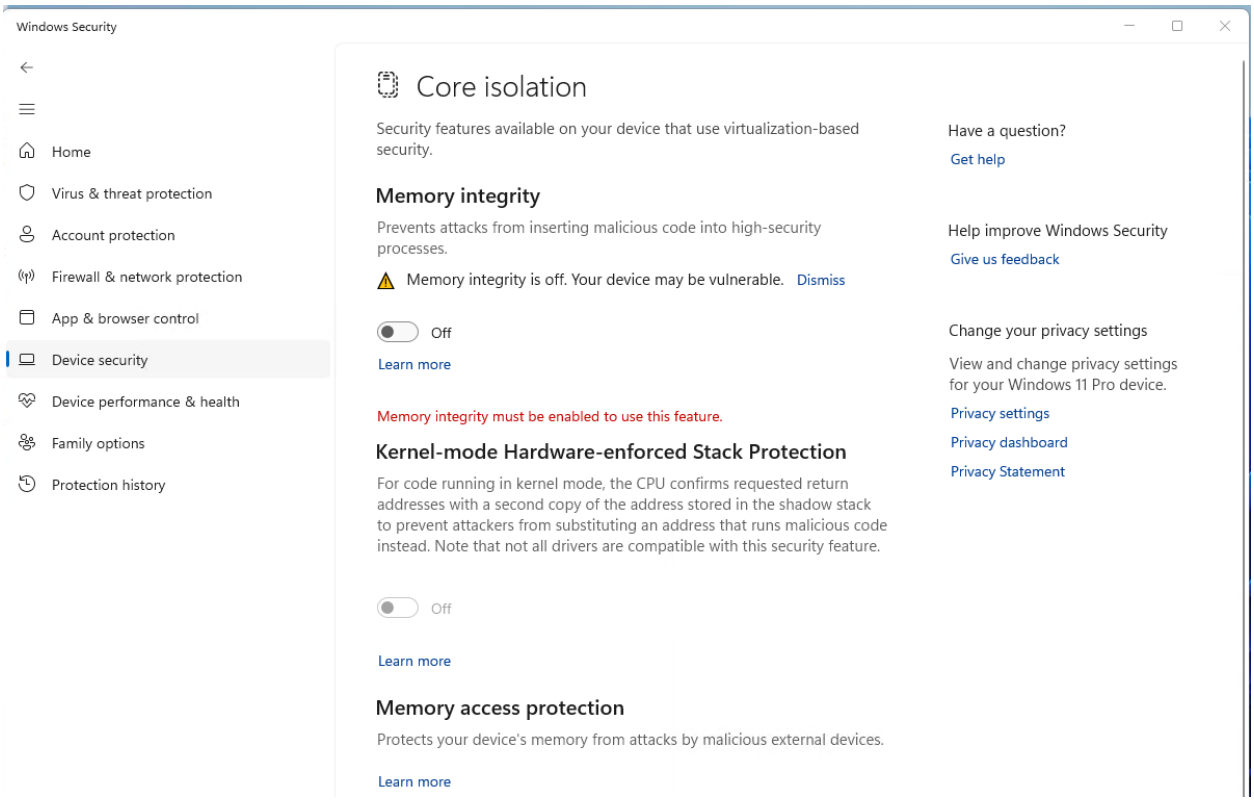
We'll also need to prevent Windows from changing power states from disrupting run-control prematurely, and VMX and VBS need to be disabled.

These steps are highly recommended (as of the time of writing) to have a successful initial debugging session, especially for newcomers to SourcePoint WinDbg. More advanced users can jump directly to having VBS/Hyper-V enabled.

To adjust the power settings in Windows, open the Control Panel > Hardware and Sounds > Power Options > Edit Plan Settings and set these per the below:



For Windows VBS, go into Windows Security > Device Security > Core Isolation and turn Memory Integrity 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 and restart.

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 the kernel debugging is not enabled, execute the command:

```
>bcdedit /debug on
```

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

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.

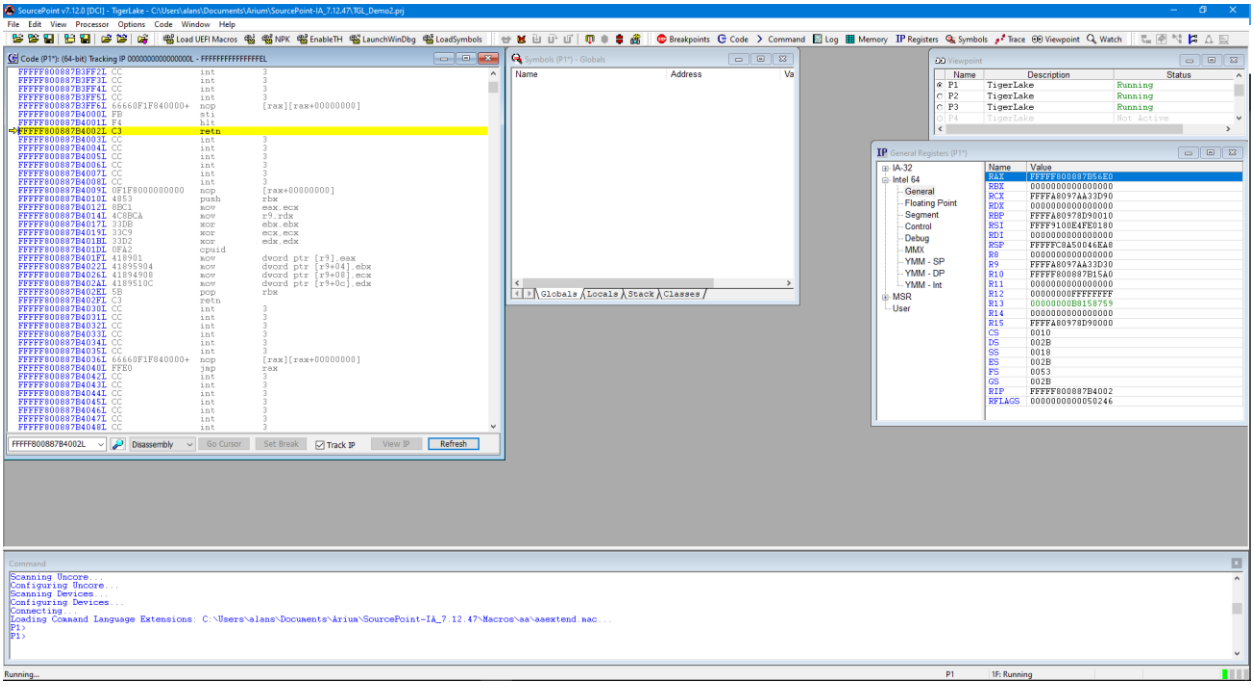
Four steps are needed to begin a debugging session with SourcePoint WinDbg:

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. Your screen should then look something like this:



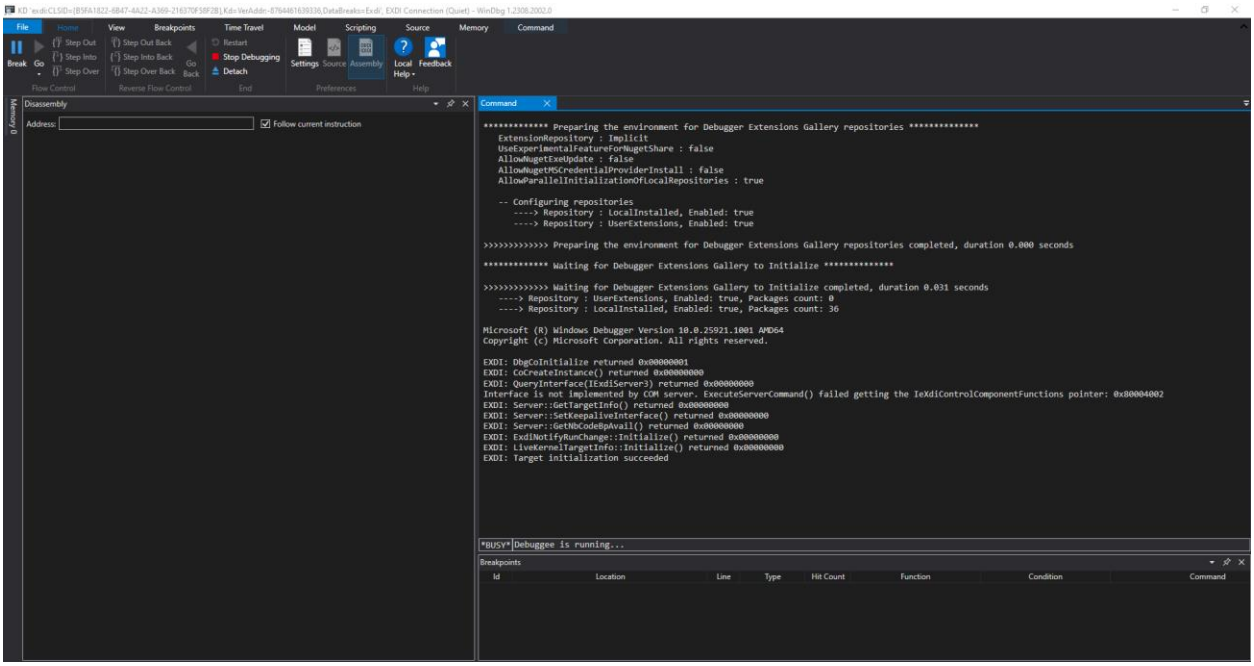
Click on the `Load WinDbg Macros` button at the top of the screen. This will enable a number of new “Windows debugging friendly” macros available for later use.

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

You may have to hit the Refresh button to see code displayed in the Code window.

Step 2: Run the StartWinDbg macro

Next, it is time to run the StartWinDbg.mac SourcePoint macro, that launches WinDbg and establishes the EXDI connection. Go to the File menu, select `Macro > Load Macro…` and select `C:\Users\<your name>\Documents\Arium\SourcePoint-IA_7.12.XX\Macros\WinDbg\StartWinDbgX.mac`. Alternatively, and more simply, click on the StartWinDbgX macro button at the top of the screen. After about 10 seconds, WinDbg will open:



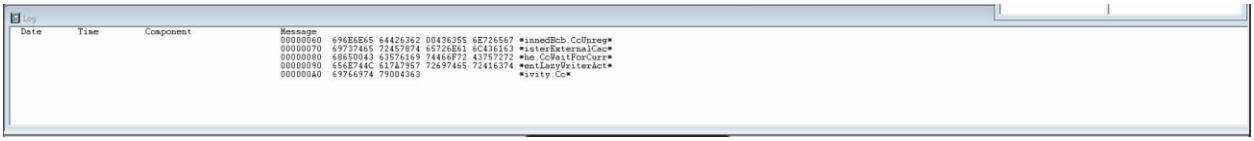
Power Tip: You can assign a button in the SourcePoint Icon Toolbar at the top to perform a one-click equivalent to the above operation, or to run any other macro. Refer to the SourcePoint User Guide or online help for how to do this.

Power Tip: You have a choice of launching WinDbg Classic or WinDbgX (aka Preview) via the macro buttons at the top. With this release of SourcePoint, WinDbg Classic is more stable. See our Troubleshooting section. Alternatively, at the SourcePoint Command line, you can type in “windbgclassic” (an alias for invoking

SetCustomWinDbgVariables("Classic", true), or windbgx (an alias for invoking SetCustomWinDbgVariable("Classic", false).

Step 3: Issue a Break from WinDbg

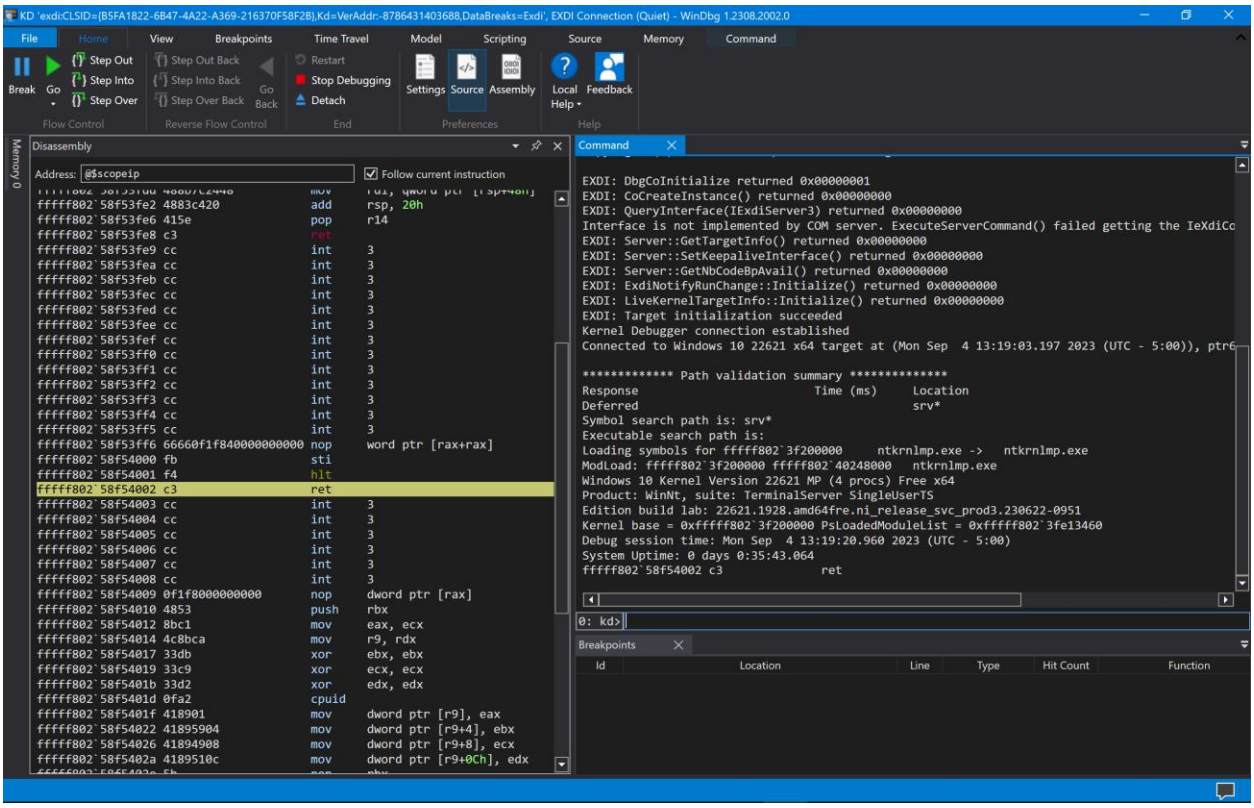
Now, hit the Break key within WinDbg. It will take ~ 30-50 seconds for SourcePoint to read the kernel memory and retrieve all the symbol information needed to match what WinDbg has (in terms of the Microsoft symbol server, or a local symbol cache). If you have the SourcePoint Log window open, you can see the symbol information being uploaded to WinDbg:

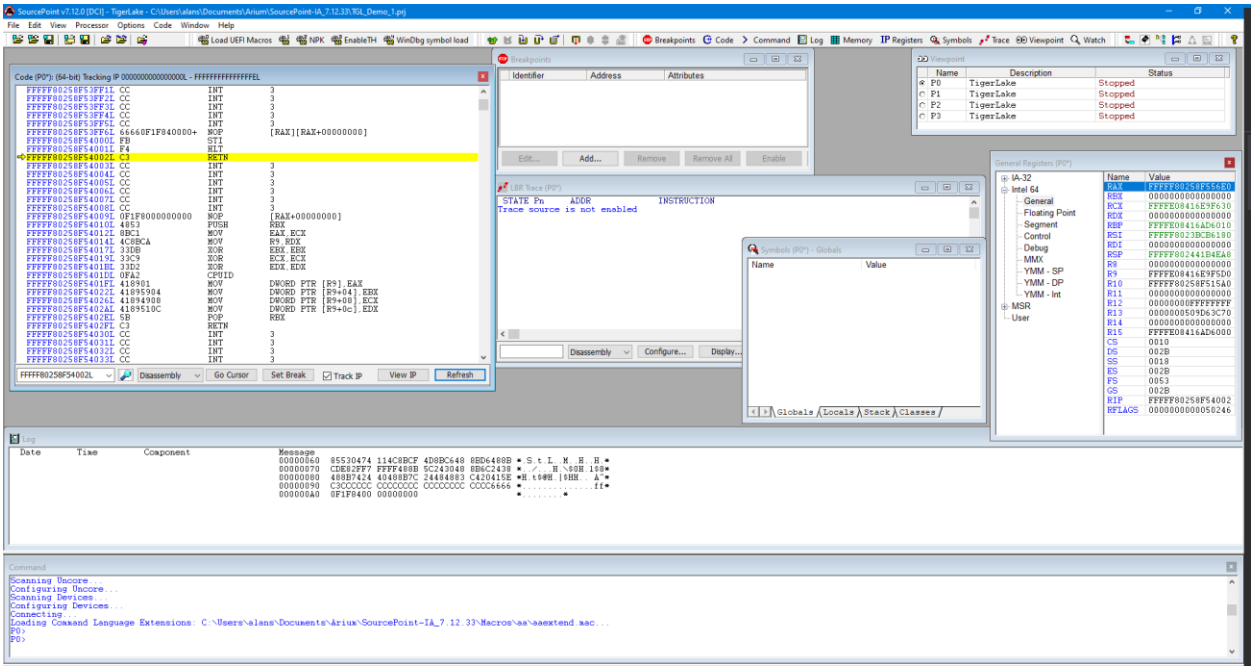


If you don't have the Log window open, you will nonetheless see the SourcePoint "Dashboard Lights" at the bottom right lighting up as the JTAG-based memory reads are done:

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 halted on logical processor 0, at a `RET` instruction:


KD 'exdi:CLSID={B5FA1822-6B47-4A22-A369-216370F58F2B},Kd=VerAddr:-8786431403688,DataBreaks=Exdi', EXDI Connection (Quiet) - WinDbg 1.2308.2002.0
File
Home
View
Breakpoints
Time Travel
Model
Scripting
Source
Memory
Command
Step Out
Step Into
Step Over
Step Out Back
Step Into Back
Step Over Back
Restart
Stop Debugging
Detach
Settings
Source
Assembly
Local Help
Feedback
Flow Control
Reverse Flow Control
End
Preferences
Help
Disassembly
Address: @$scopeip
Follow current instruction
fffff802`58f53fe2 4883c420 add rsp, 20h
fffff802`58f53fe6 415e pop r14
fffff802`58f53fe8 c3 ret
fffff802`58f53fe9 cc int 3
fffff802`58f53fea cc int 3
fffff802`58f53feb cc int 3
fffff802`58f53fec cc int 3
fffff802`58f53fed cc int 3
fffff802`58f53fee cc int 3
fffff802`58f53fef cc int 3
fffff802`58f53ff0 cc int 3
fffff802`58f53ff1 cc int 3
fffff802`58f53ff2 cc int 3
fffff802`58f53ff3 cc int 3
fffff802`58f53ff4 cc int 3
fffff802`58f53ff5 cc int 3
fffff802`58f53ff6 66660f1f840000000000 nop word ptr [rax+rax]
fffff802`58f54000 fb sti
fffff802`58f54001 f4 hlt
fffff802`58f54002 c3 ret
fffff802`58f54003 cc int 3
fffff802`58f54004 cc int 3
fffff802`58f54005 cc int 3
fffff802`58f54006 cc int 3
fffff802`58f54007 cc int 3
fffff802`58f54008 cc int 3
fffff802`58f54009 0f1f8000000000 nop dword ptr [rax]
fffff802`58f54010 4853 push rbx
fffff802`58f54012 8bc1 mov eax, ecx
fffff802`58f54014 4c8bca mov r9, rdx
fffff802`58f54017 33db xor ebx, ebx
fffff802`58f54019 33c9 xor ecx, ecx
fffff802`58f5401b 33d2 xor edx, edx
fffff802`58f5401d 0fa2 cpuid
fffff802`58f5401f 418901 mov dword ptr [r9], eax
fffff802`58f54022 41895904 mov dword ptr [r9+4], ebx
fffff802`58f54026 41894908 mov dword ptr [r9+8], ecx
fffff802`58f5402a 4189510c mov dword ptr [r9+0Ch], edx
Command
EXDI: DbgCoInitialize returned 0x00000001
EXDI: CoCreateInstance() returned 0x00000000
EXDI: QueryInterface(IExdiServer3) returned 0x00000000
Interface is not implemented by COM server. ExecuteServerCommand() failed getting the IeXdiCo
EXDI: Server::GetTargetInfo() returned 0x00000000
EXDI: Server::SetKeepaliveInterface() returned 0x00000000
EXDI: Server::GetNbCodeBpAvail() returned 0x00000000
EXDI: ExdiNotifyRunChange::Initialize() returned 0x00000000
EXDI: LiveKernelTargetInfo::Initialize() returned 0x00000000
EXDI: Target initialization succeeded
Kernel Debugger connection established
Connected to Windows 10 22621 x64 target at (Mon Sep 4 13:19:03.197 2023 (UTC - 5:00)), ptr6
************ Path validation summary **************
Response Time (ms) Location
Deferred srv*
Symbol search path is: srv*
Executable search path is:
Loading symbols for fffff802`3f200000 ntkrnlmp.exe -> ntkrnlmp.exe
ModLoad: fffff802`3f200000 fffff802`40248000 ntkrnlmp.exe
Windows 10 Kernel Version 22621 MP (4 procs) Free x64
Product: WinNt, suite: TerminalServer SingleUserTS
Edition build lab: 22621.1928.amd64fre.ni_release_svc_prod3.230622-0951
Kernel base = 0xfffff802`3f200000 PsLoadedModuleList = 0xfffff802`3fe13460
Debug session time: Mon Sep 4 13:19:20.960 2023 (UTC - 5:00)
System Uptime: 0 days 0:35:43.064
fffff802`58f54002 c3 ret
0: kd>
Breakpoints
Id
Location
Line
Type
Hit Count
Function

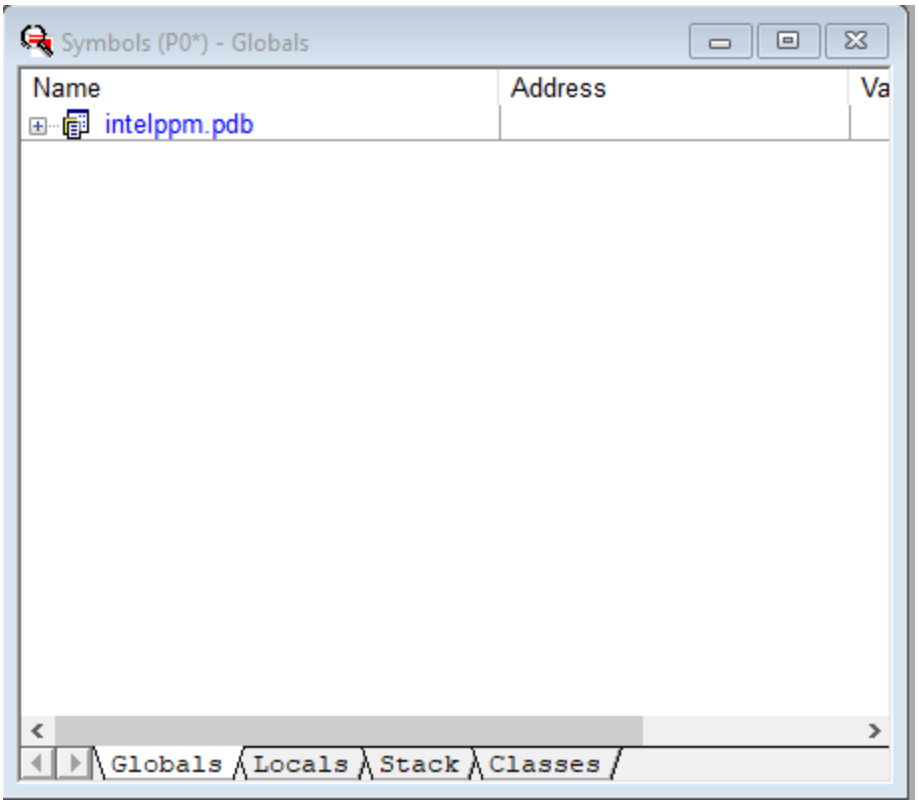

Code (P0*): (64-bit) Tracking IP
Breakpoints
Identifier
Address
Attributes
Viewpoint
Name
Description
Status
P0 TigerLake Stopped
P1 TigerLake Stopped
P2 TigerLake Stopped
P3 TigerLake Stopped
LBR Trace (P0*)
STATE Fn ADDR INSTRUCTION
Trace source is not enabled
Symbols (P0*) - Globals
Name
Value
General Registers (P0*)
IA-32
Intel 64
General
Floating Point
Segment
Control
Debug
MMX
YMM - SP
YMM - DP
YMM - Int
MSR
User
Log
Date
Time
Component
Message
Command
Scanning Uncore...
Configuring Uncore...
Scanning Devices...
Configuring Devices...
Connecting...
Loading Command Language Extensions: C:\Users\alans\Documents\Arium\SourcePoint-IA_7.12.33\Macros\aa\aaextend.mac...
P0>
P0>

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. Follow the following steps:

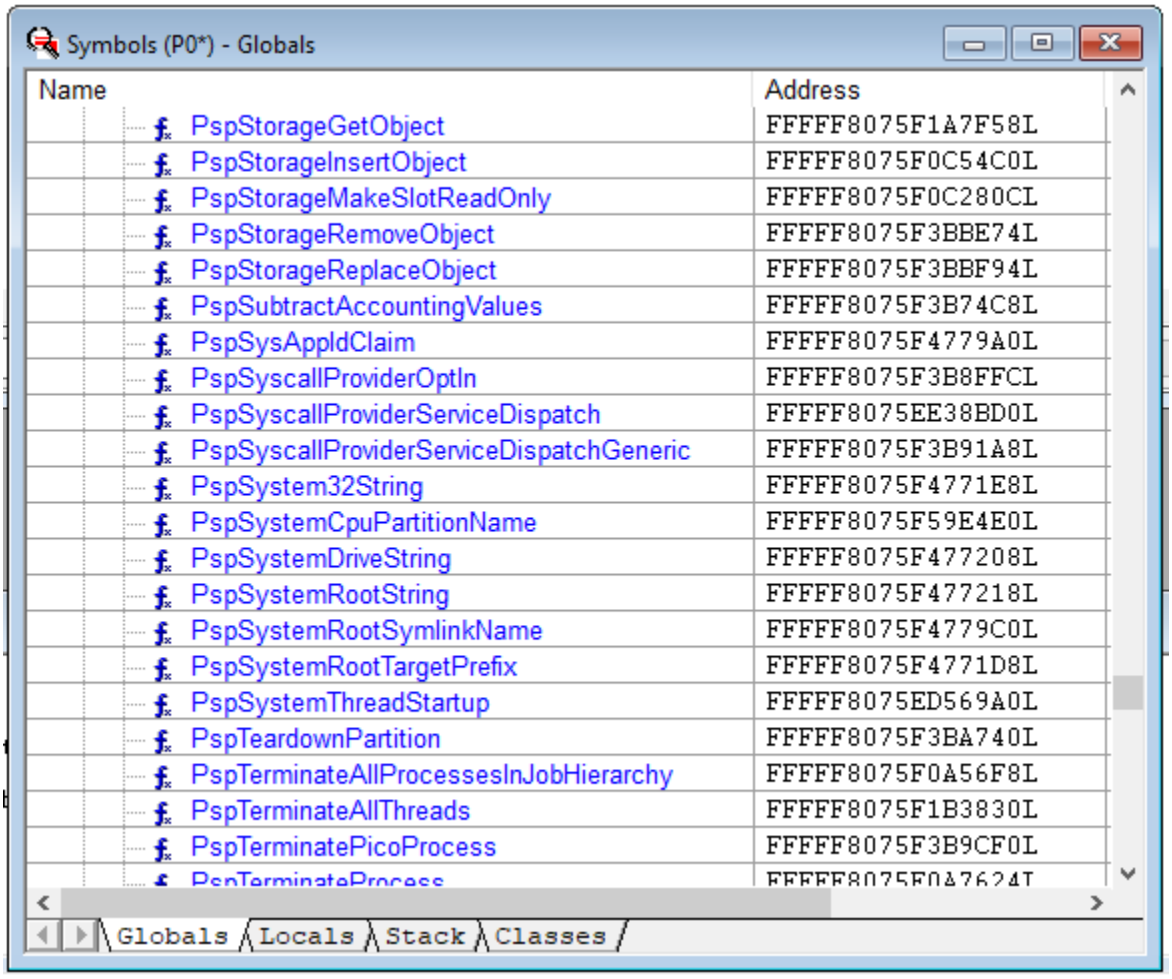
Ensure that the target is in a Stopped state. Hit Break within WinDbg if necessary.

Load the symbols by going into SourcePoint, under the File menu, select `Macro > Load Macro…` and select `C:\Users\<your computer>\Documents\Arium\SourcePoint-IA_7.12.XX\Macros\WinDbg\Load Current.mac`. 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 display those symbols.

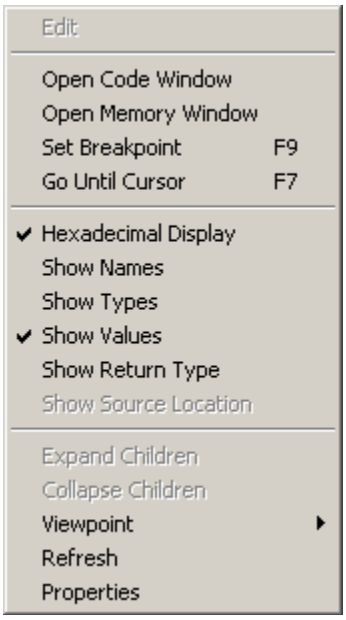
Expand the Labels within the Symbols window, and then you will see it populated with all functions that are in the current module, for example:



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:



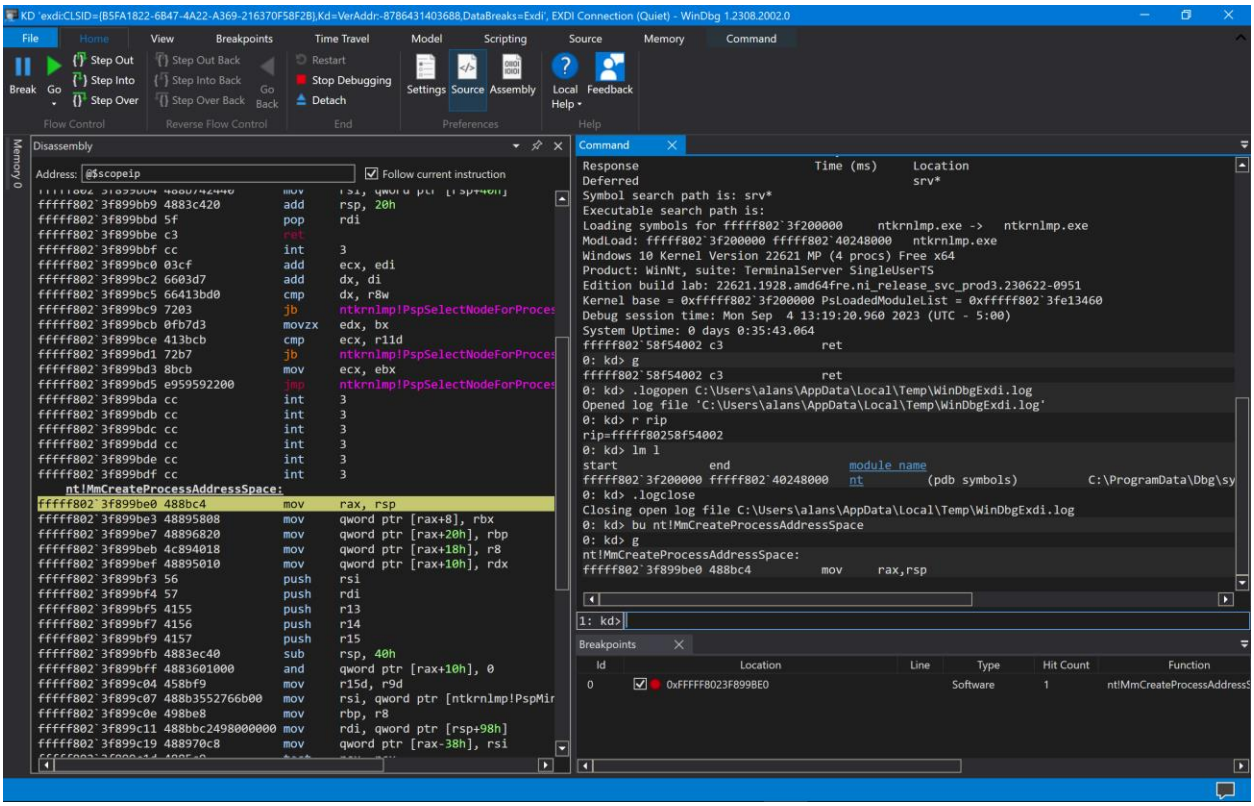
Now, it is possible to see the power of the two applications applied together. As an example, go into WinDbg and set a breakpoint on the entry point to the function `MmCreateProcessAddressSpace`:

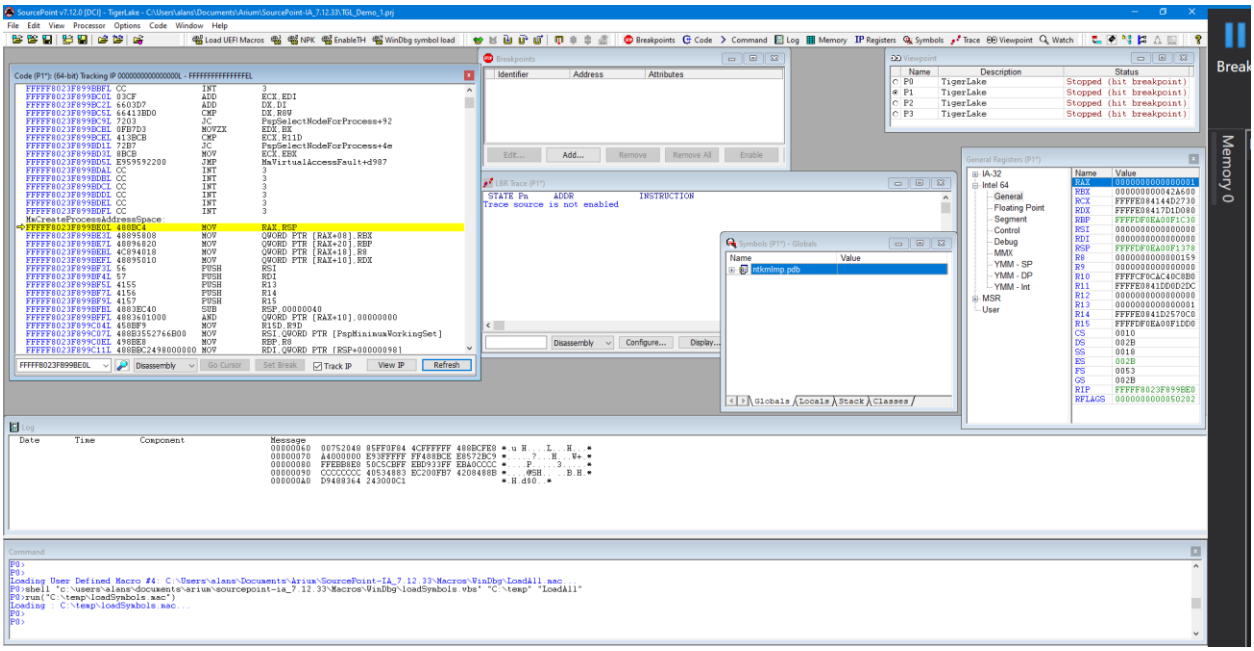
```
bp nt!MmCreateProcessAddressSpace
```

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. They 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. Symbol path in most WinDbg installations is srv*, which SourcePoint has not knowledge of, so it will next go to the cache path. WinDbg tends to store an extraneous "\sym" in the cache path that needs to be worked around. Use the following commands as needed:

```
WinDbgCachePath                 // displays the Cache path
WinDbgSymPath                   // displays the Symbol path
WinDbgCachePath                 // sets the cache path manually; i.e.
WinDbgCachePath = "C:\\symbols"
```

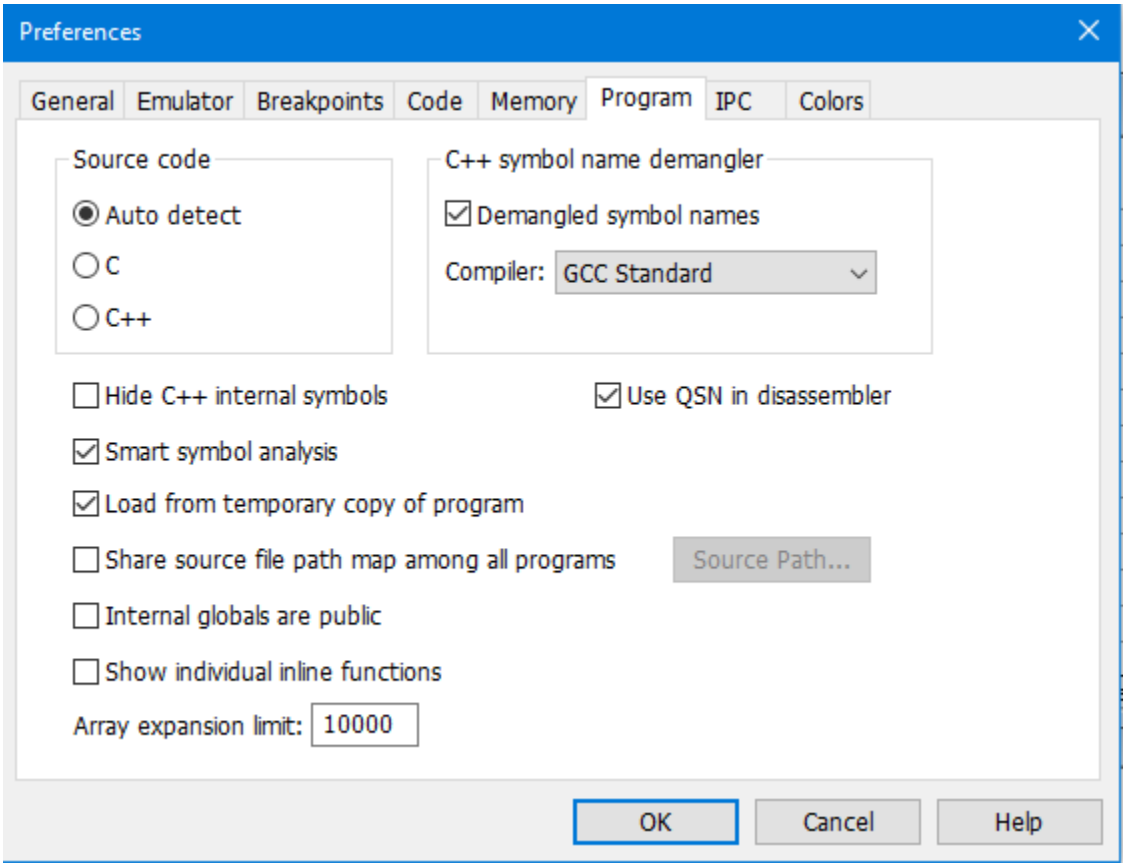
Power Tip: You must be in Host mode to use the vmcs macro functions.

Power Tip: You must do WinDbgCachePath after the first LoadCurrent, or it does not stick for the session. To be fixed.

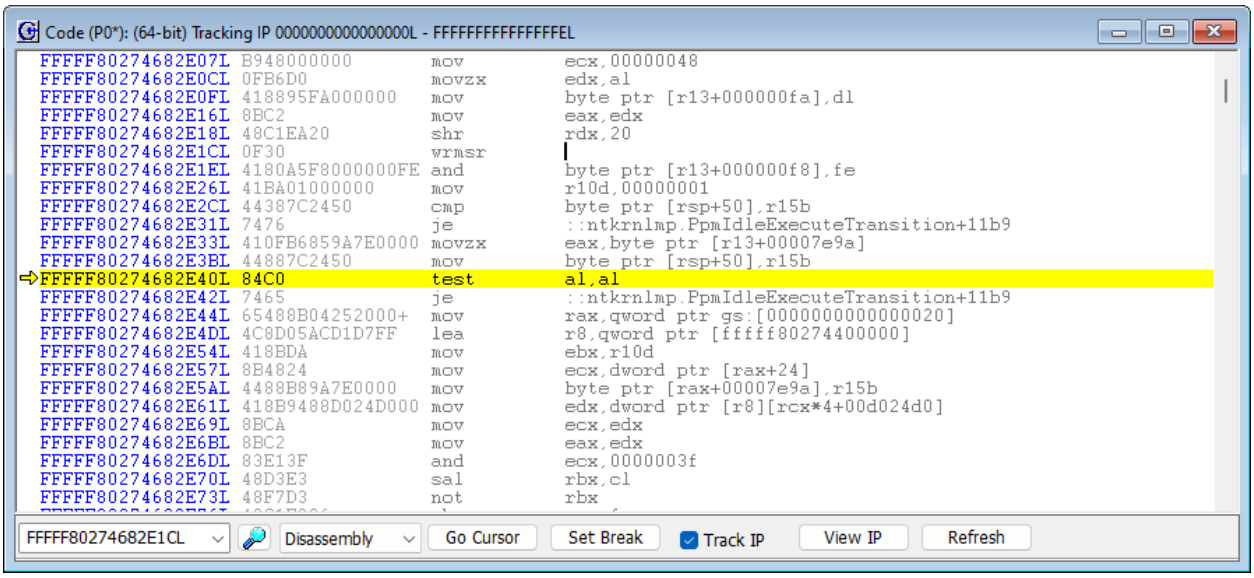
Getting SourcePoint to display module names as well as function names

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:



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

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

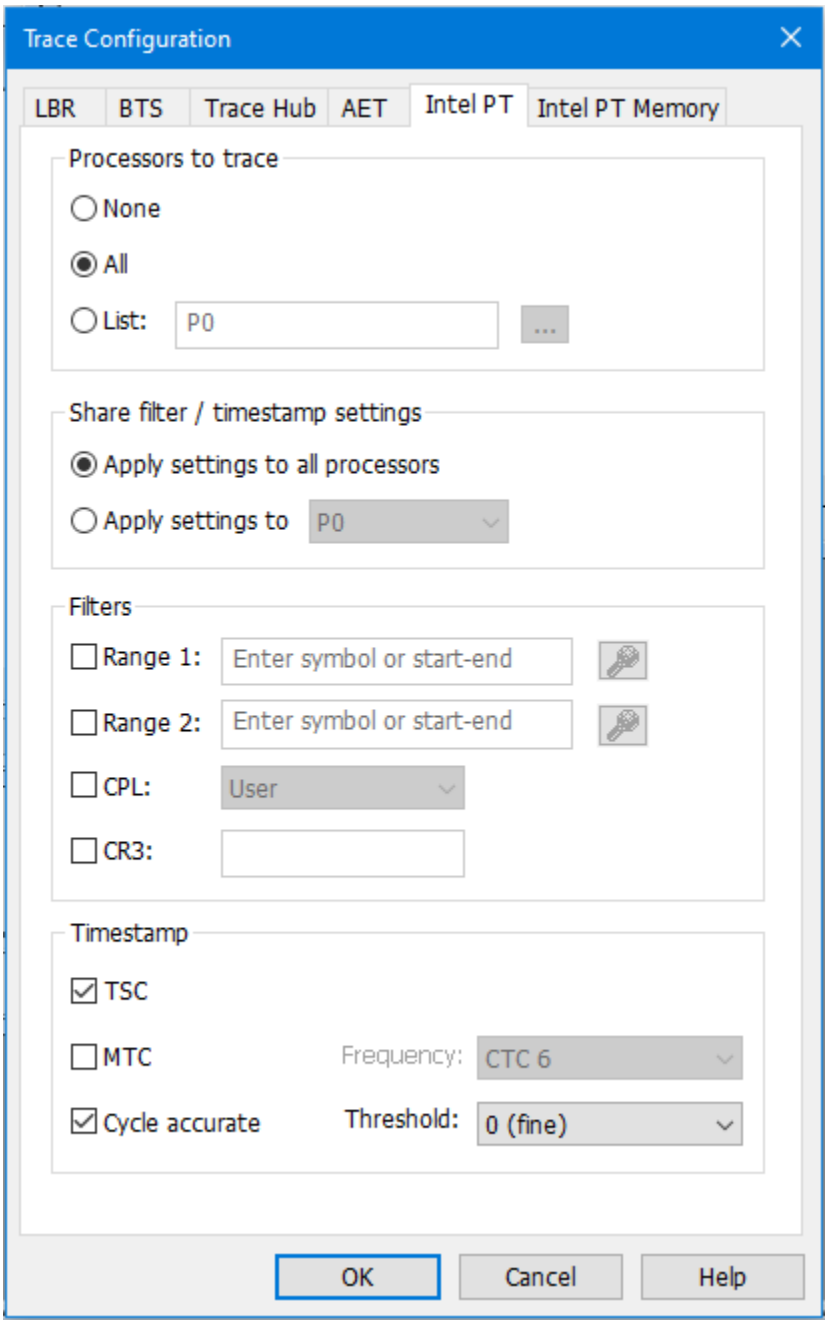
Do a Project Save to save these settings into your Project, so they'll automatically load for your next session.

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.

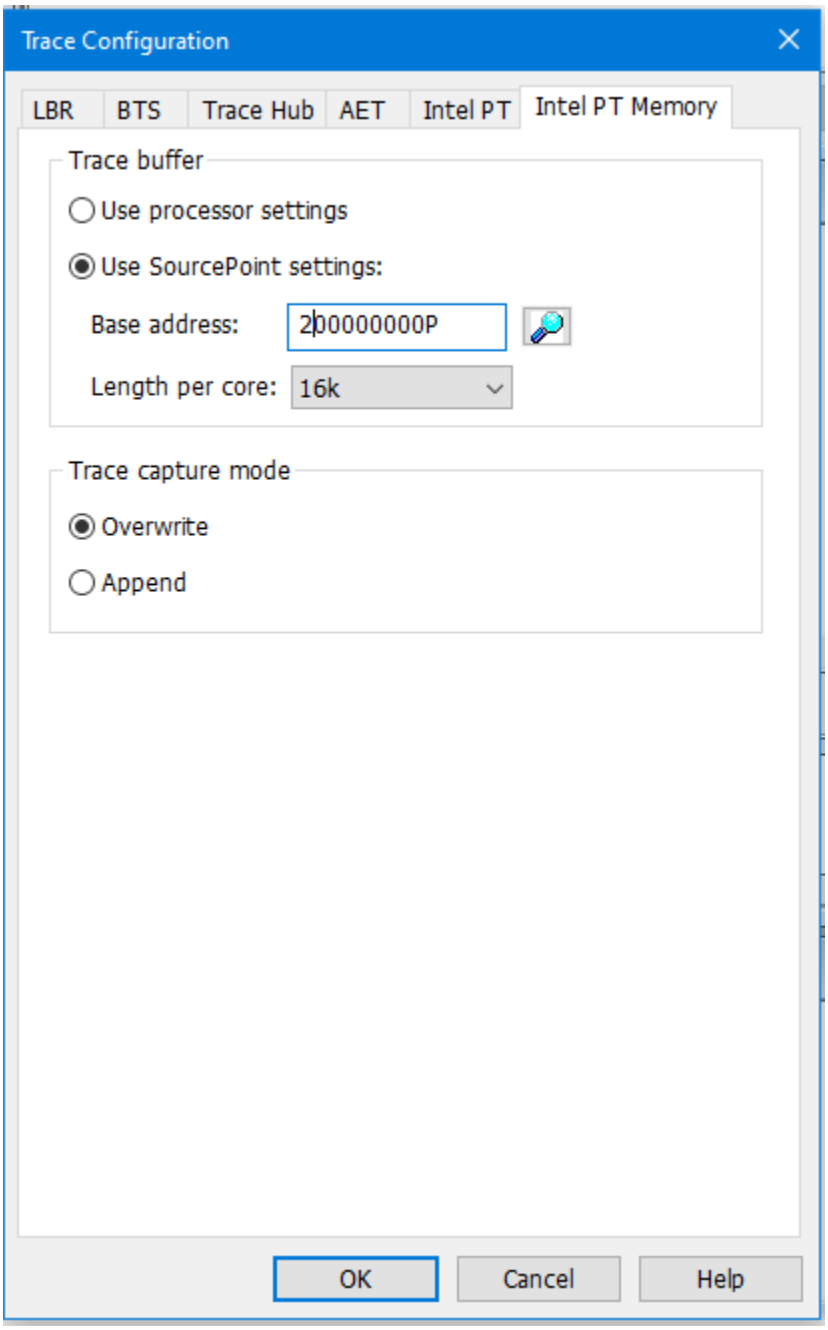
First, ensure that the target is in a Stopped state. If not, issue a Break from within WinDbg.

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:

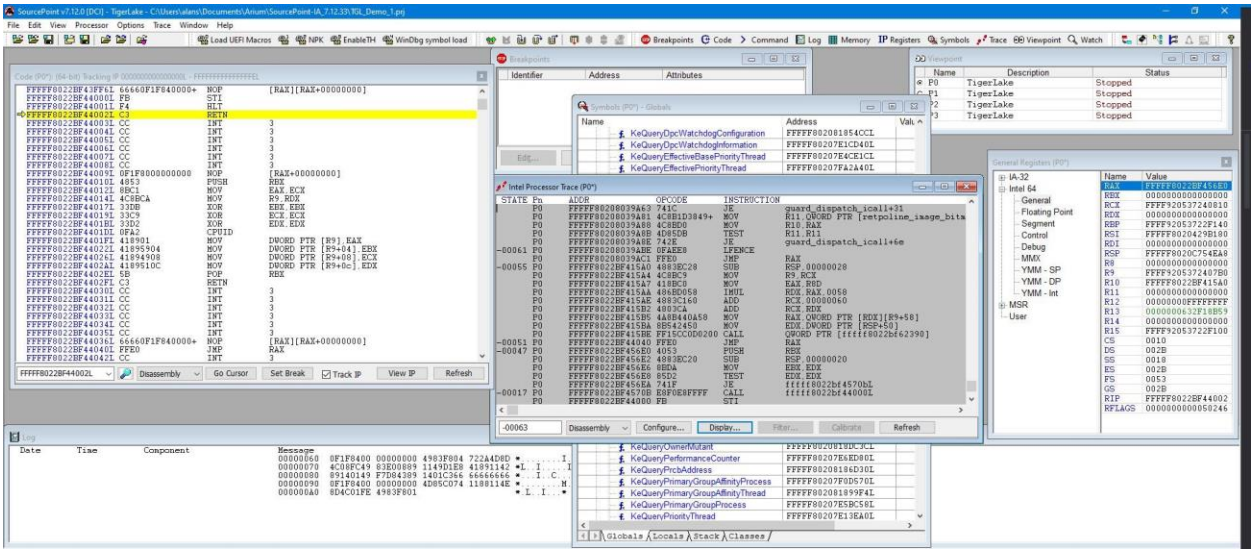


NOTE: “Use processor settings” can be selected if the BIOS has been set up with this. For the UP Xtreme i11 board, this is not the default.

Click OK. The Intel Processor Trace window will, after a few seconds, ***sometimes*** refresh with some data. This first set of execution trace is not valid. It’s a feature of SourcePoint that enables a JTAG “hotplug” dump of processor instructions if the

emulator had been unplugged and then plugged back in again. Only subsequent Go and Stop will yield valid trace data.

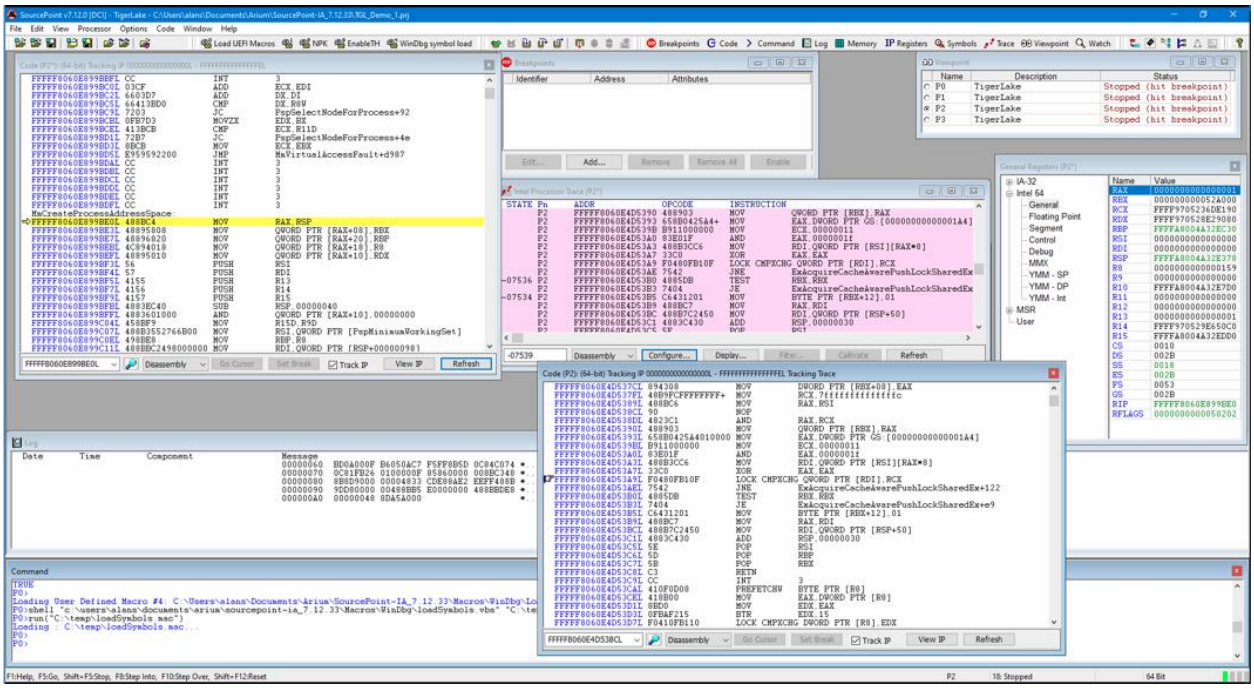
Then hit Go from within WinDbg, and then hit Break, and you will see something like the below in SourcePoint:



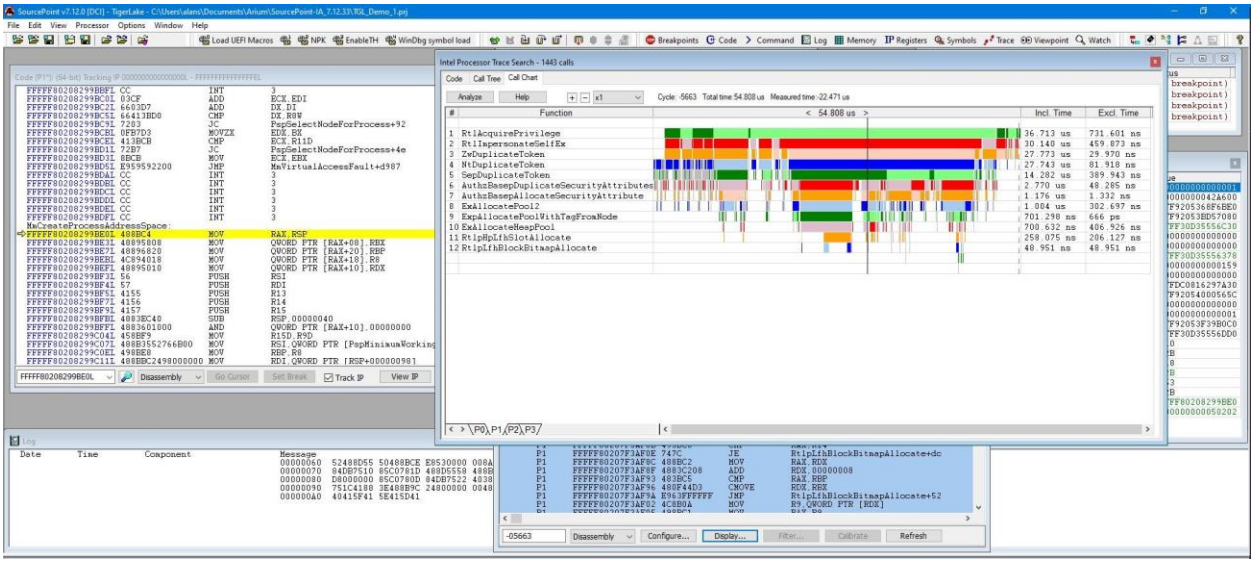
Feel free to resize the Intel Processor Trace window, and make it Floating, to see the trace data.

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; and Source Lines in the Source code section.

You can click the cursor at any code line within the Intel Processor Trace window, and right-click to open up a Tracking Trace 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 Trace 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:



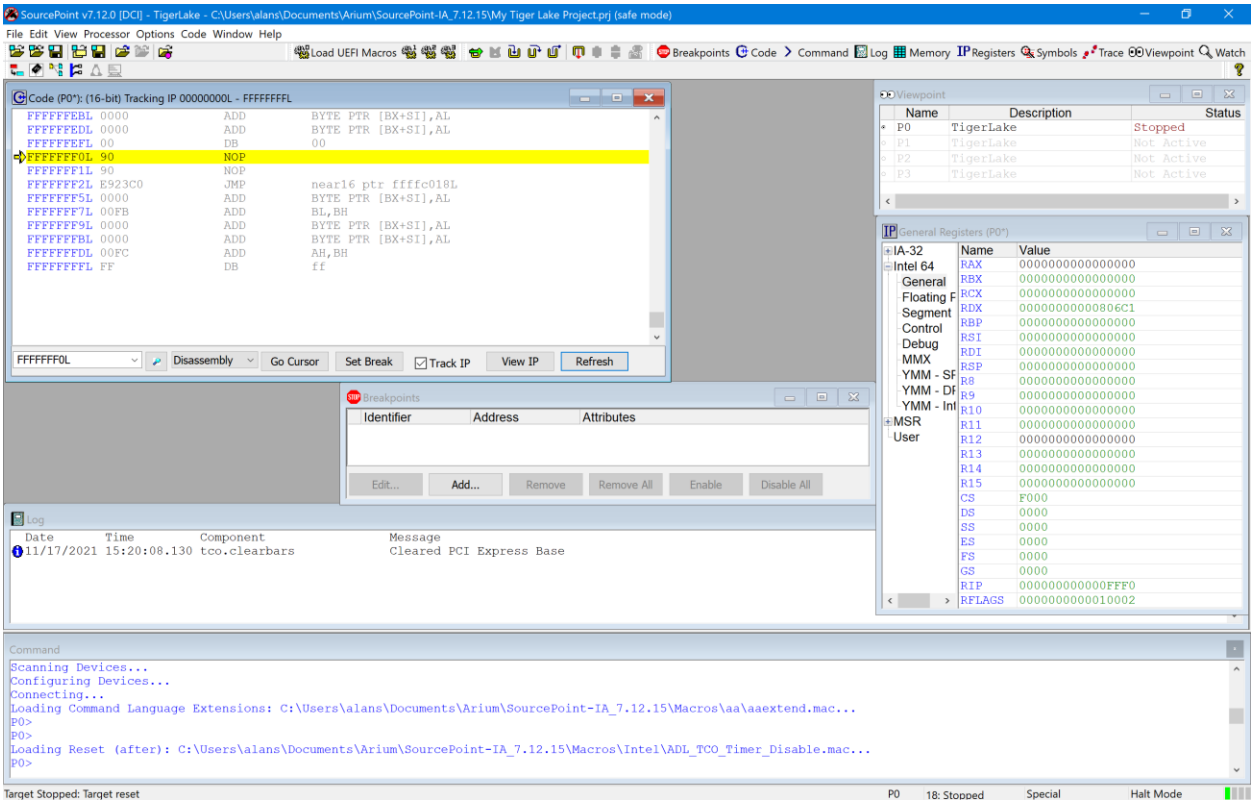
Move the time arrow by clicking on a section of code, or use your arrow keys. Expand the view of a particular area of code with the mouse wheel, or using the Expand (starts at x1) drop-down or +/- buttons at the top.

Event Trace

First Step: Configuring the Intel Trace Hub

Event tracing is accomplished with the Intel Trace Hub (ITH). Fortunately, using DCI, events supported by the ITH can be streamed directly out of system reset. The one limitation that exists is that some events (like Port IN/OUT tracing) happen so frequently at some points of the boot process that they overwhelm the capacity of the USB 2.0 (DbC2) connection and event processing, and thus cause trace buffer overflows – but these should be rare as long as the events collected are relatively close to the debug point of interest.

The first thing to do is to configure the ITH. Reset the target by clicking on the Reset button in the Icon Toolbar at the top of the screen, and it will halt at the reset vector, physical address FFFFFFF0:

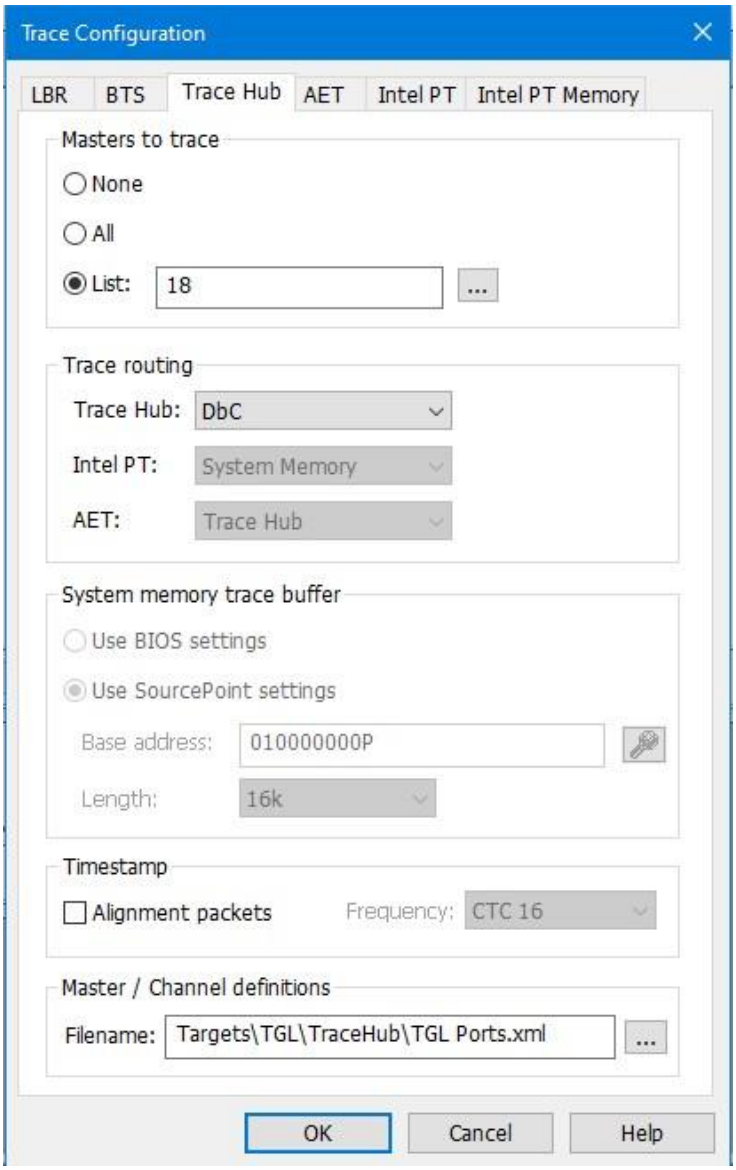


The next step is to run the ITH macros that enable the Trace Hub and hide it from the OS. Fortunately, SourcePoint comes equipped with a macro button built in to make this process easier, At the top of the screen, you'll see a button labeled `NPK` and `EnableTraceHub`. Click on it. Wait about five seconds.

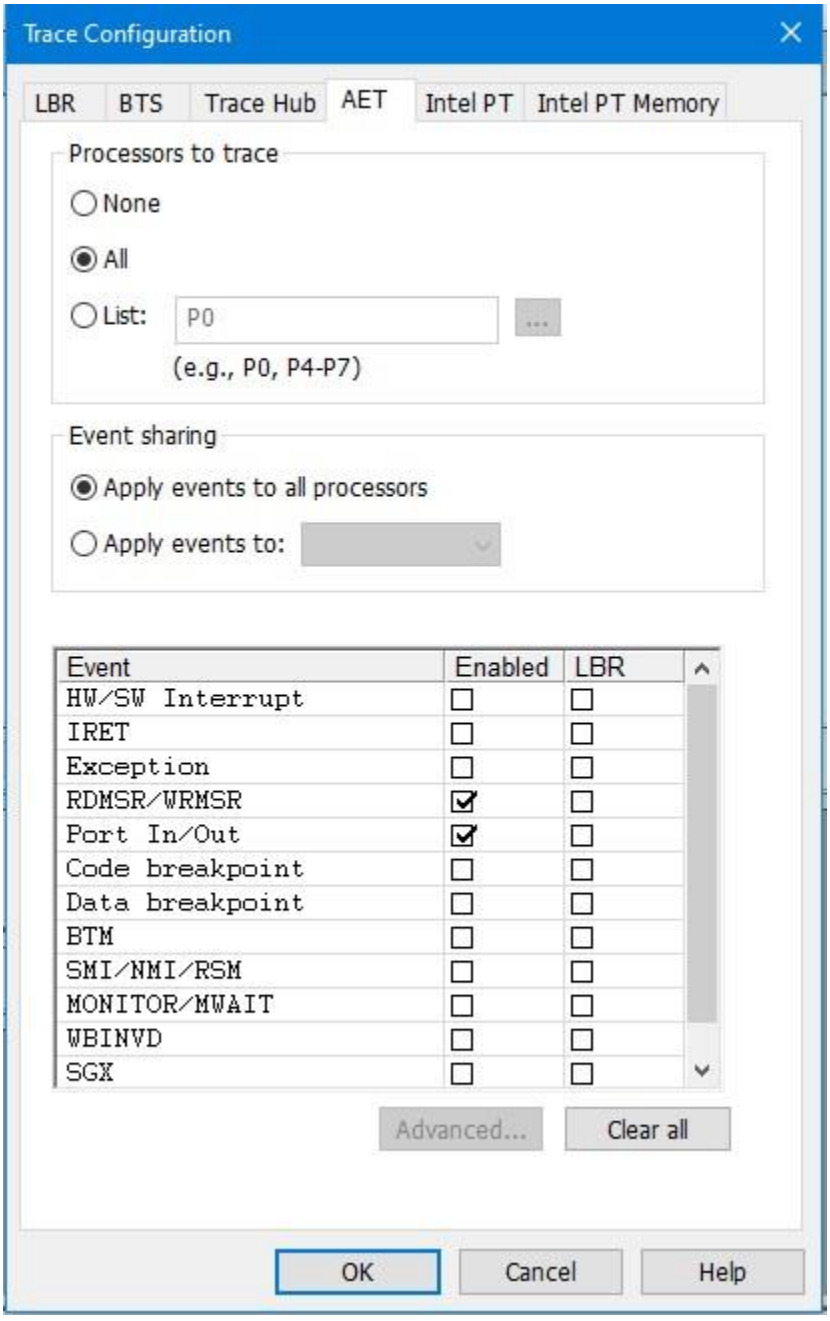
You can then boot up to Windows (or not).

Second Step: Set up Architectural Event Trace

Now, it's time to tell the Trace Hub what you want to trace. 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 for the Tiger Lake platform:



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:

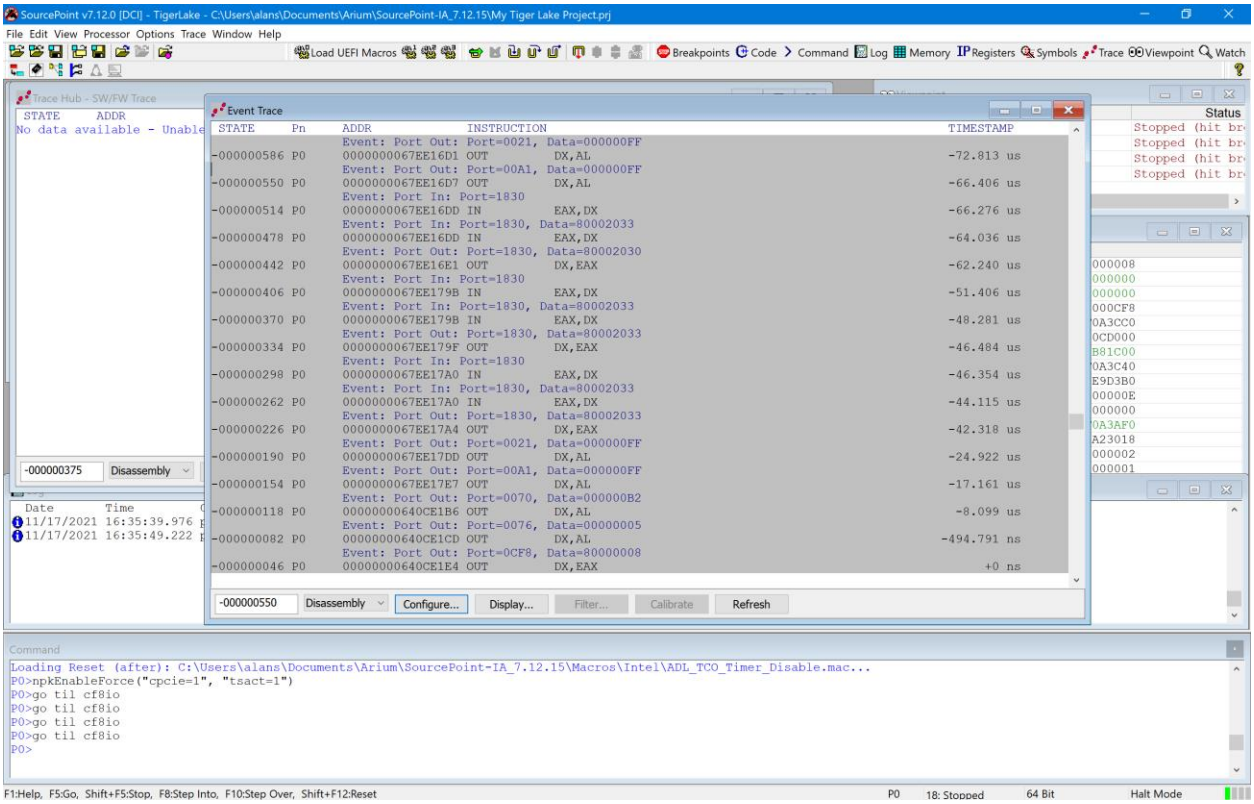


Now, you can simply do a Go/Stop to capture the event trace data. Below shows the use the `Command` window to simulate a break on any read/write of, say, port x'CF8', the PCI CONFIG_ADDRESS. This is conveniently done by issuing at the `Command` window `P0>` prompt:

```
go til cf8io
```

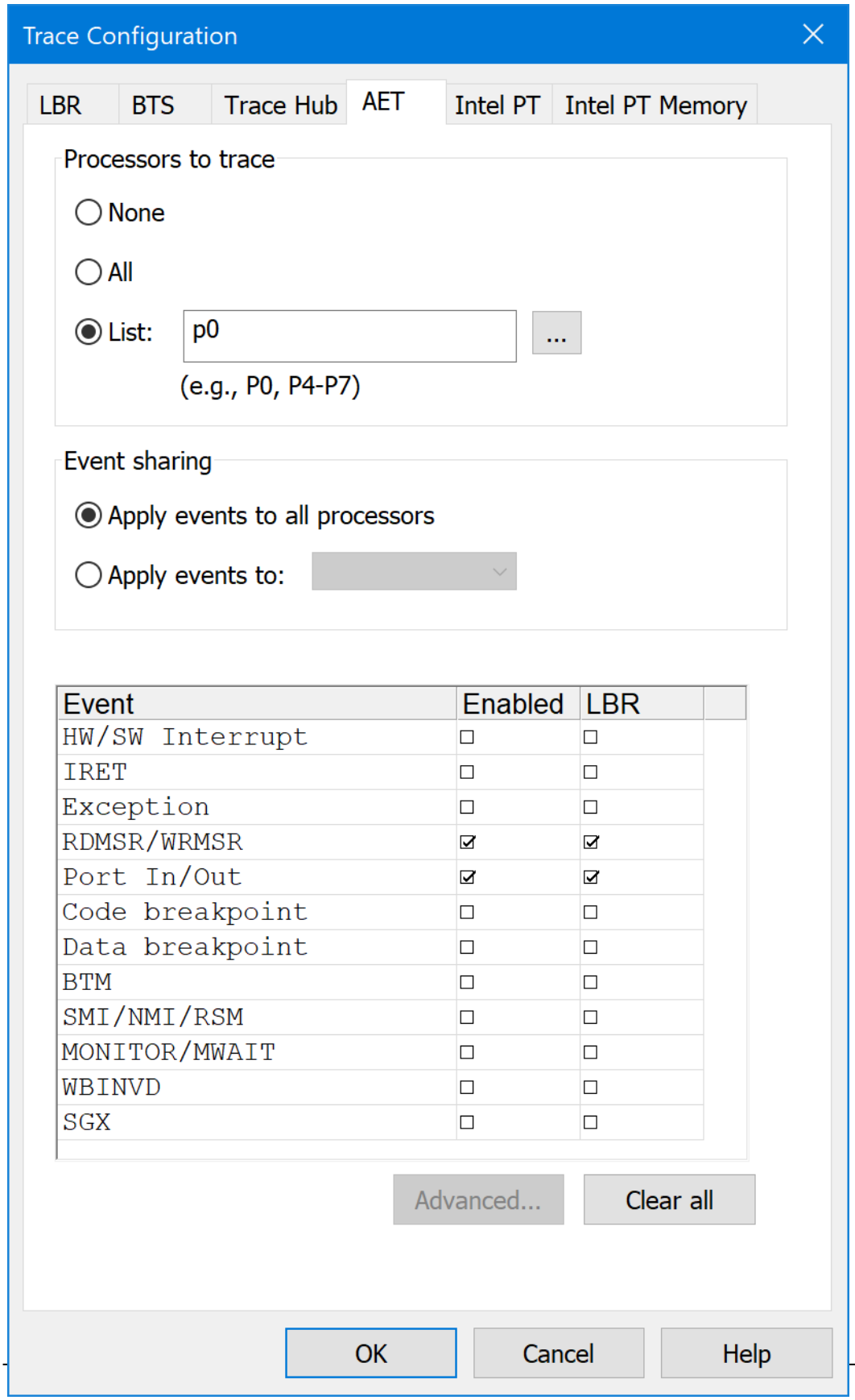
This will run the target until the next IN or OUT to CF8.

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.


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

Getting Started with Hyper-V/VBS Debug

Debugging with Hyper-V and Virtualization-Based Security (VBS) is beta for this SourcePoint 7.12.52 release.

Pre-requisite reading for this section is as follows, with some tips below. This content will be incorporated in this document upon the production release:

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.
Very important note: the command to disable the synthetic watchdog on the target is:
`>bcdedit /set {default} loadoptions "systemwatchdogpolicy=disabled"`

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 preamable article to using Intel AET to capture RDMSR and WRMSR events, and correlating them against the Windows MSR bitmap. For more advanced users only.

SourcePoint 7.12.52 has added breakpoint support for VM Resume.

Power Tip: Don't do LoadCurrent from User Space. The macro will scan forever. To be fixed.

Troubleshooting Tips

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.

Mangled function names

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

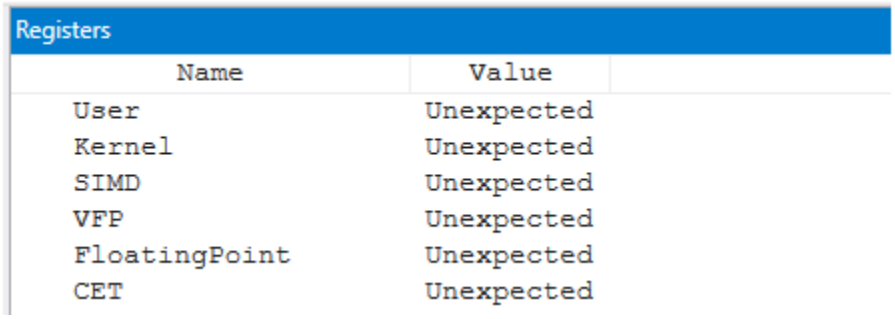
```
JNE ??_C@_0BH@CBDMLJDN@RtlCreateUnicodeString@
```

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

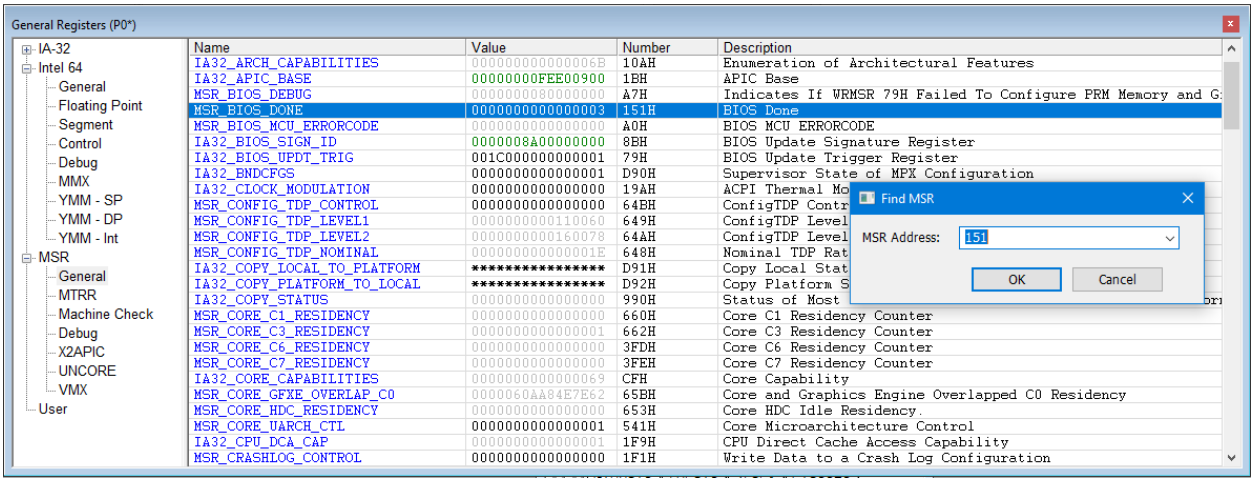
WinDbg Register window slows things down badly

Having the Registers window open within WinDbg slows things down quite a bit. Updating the WinDbg Registers view causes EXDI to transact hundreds of memory reads. This can cause problems. In particular, it has been seen to cause failures of the symbol load from WinDbg to SourcePoint. LoadCurrent.mac will fail quietly.

If it remains open, you may at some point see the below within WinDbg:



Close it out (presuming that you had it open), and consider using the SourcePoint Registers window instead. You can see all the GPRs, Control Registers, Debug Registers, MSRs, VMX registers, and many more. And context-sensitive help (right-click) provides the selective ability to find a particular MSR, open a Code or Memory window, and other features.



WinDbg FP register display is not working

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

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.

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.

Windows crashes

If you work with SourcePoint WinDbg long enough, you'll likely crash Windows at some point. Sometimes Automatic Repair will clean things up, sometimes 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.

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.

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!