SourcePoint

Getting Started Guide for the

AAEON UP Xtreme i11

Revision 2.4

Contents

REVISION HISTORY

Revision Number	Description	Date
1.0	Original document, describes v0000 board support	November 28, 2021
2.0	Added content for new support of v0001 (with the Type-C connector removed) AAEON UP Xtreme i11 board	May 30, 2022
2.1	Updated for WinDbg support and other perfective changes	December 3, 2023
2.2	Update for beta release SourcePoint 7.12.52	March 31, 2024
2.3	Production release documentation for 7.12.53.	May 5, 2024
2.4	Production release for 7.12.59. Consolidating SourcePoint UEFI and SourcePoint WinDbg documentation.	July 13, 2024

WELCOME!

Thank you very much for your SourcePoint purchase! We appreciate you acquiring our best-in-class debugger, and hope you enjoy using it. We strive to deliver the most powerful, easy-to-use and polished product as possible. So, please feel free to share your feedback directly at our support site at https://www.asset-intertech.com/support/, or via your favorite social media outlet.

As with any new tool, mastering SourcePoint takes an investment in terms of time and effort. JTAG-based debug is a fairly specialized area, and low-level “on the metal” firmware development on x86 platforms is even more so. So, in your use of the tool, you may encounter behavior that seems non-intuitive or even wrong. You may be encountering a tool corner case, a limitation inherent in JTAG or DCI, or even a bug. If so, try a few different options as may be referenced in the Troubleshooting section of this Guide, and if it persists, give us a call. We are happy to support you.

GETTING STARTED WITH SOURCEPOINT

Read this section first! It provides an introduction to the configuration changes you need to make on the AAEON UP Xtreme i11 target, and provides a basic overview on running SourcePoint. Then, you can jump ahead to debugging Windows and Hyper-V.

Boards and Cables

The board covered in this document is the AAEON UP Xtreme i11 board, based upon the Intel Tiger Lake CPU.

The Tiger Lake boards come in four flavors: Celeron, i3, i5 and i7. All boards are suited for Windows debugging. The Celeron board is the least expensive, and supports all the latest Intel debug and trace features such as Intel Processor Trace (Intel PT), Intel Trace Hub, Architectural Event Trace (AET), and others.

WARNING:

Do ***NOT*** plug a regular USB cable into the target and attempt to use DCI. Specialty cables, with VBUS snipped, are required; using a regular USB cable may possibly fry your target, or worse.

The main source to purchase the specialty DCI cable needed for SourcePoint debugging is ASSET InterTech. This target has its Type-C port enabled for DCI. If you have a debug host with a Type-A port, you'll need to purchase the part # ITPDCIAMCM1MU (1.0 meter) cable. The longer 1.8-meter ITPDCIAMCM2MU cable will work as well. If your host has a Type-C port, purchase the ITPDCIC2CD2U1M. Using Type-A/C hubs have been seen to work, but are not warranted. Type A/C adapters have been seen not to work.

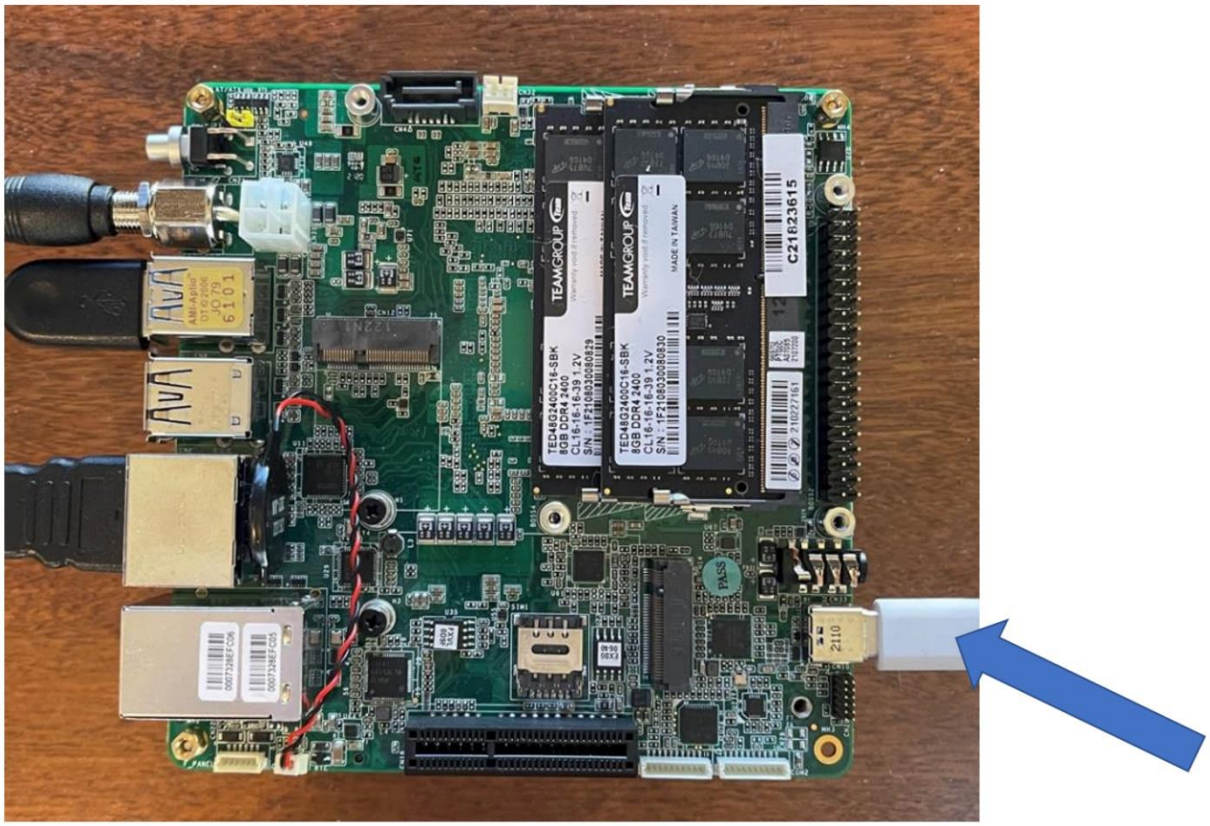
Contact your ASSET representative to obtain the cable needed for your configuration.

BIOS Settings

The AAEON UP Xtreme i11 boards come equipped with an AMI Aptio BIOS that is based upon typical Intel Customer Reference Board (CRB) BIOS.

Luckily, the platform comes with all the necessary hardware hooks and firmware straps to support Intel Direct Connect Interface (DCI) out of the box.

The USB Type C port on the board is the port of interest:



There are three BIOS settings to change for the board to work with SourcePoint and DCI:

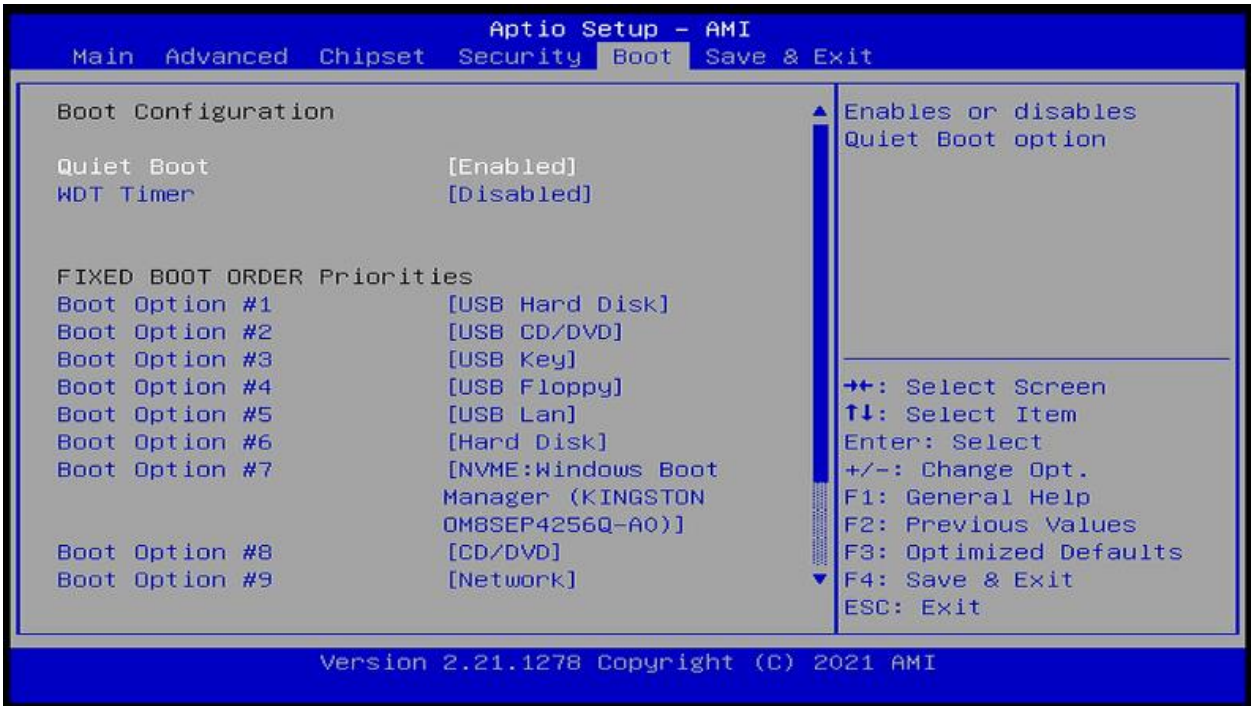
There are three settings in the default AAEON BIOS that need to be changed to successfully JTAG debug the UP Xtreme i11 target:

1. Disable the WDT timer
2. Disable the TCO timer
3. Set the `Platform Debug Consent` to Enabled (USB2 DbC).

Go into the BIOS Boot menu (this is accomplished by rebooting the board, at the same time holding down the F7 key, and you'll be promoted for the password for the "CRB Advanced" BIOS menu. Note that CRB is Customer Reference Board, an Intel reference:

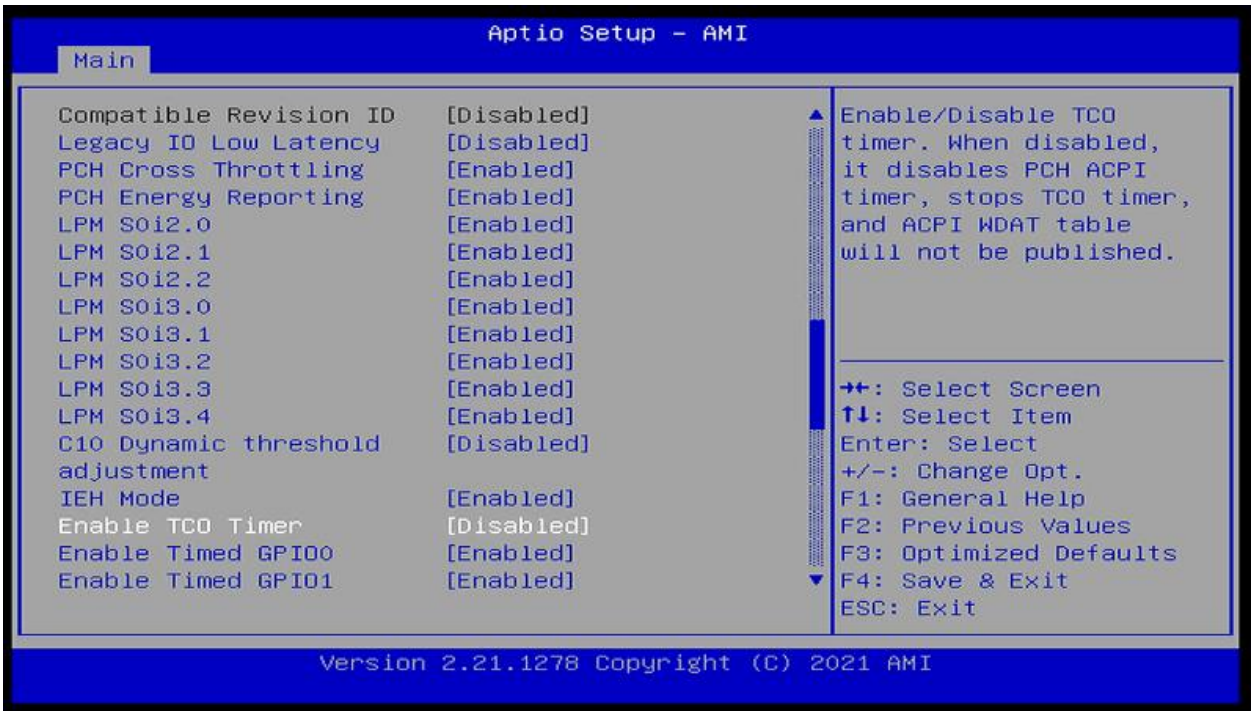
The password is upassw0rd.
And that is a number "zero" (0) in the password, not the letter "O".

Enter Setup, and use the arrow key to move over to the Boot menu. and set WDT Timer -> Disabled. If you don't, run-control will be successful, but the board will power-cycle every 30 seconds; putting a real crimp in your debugging!

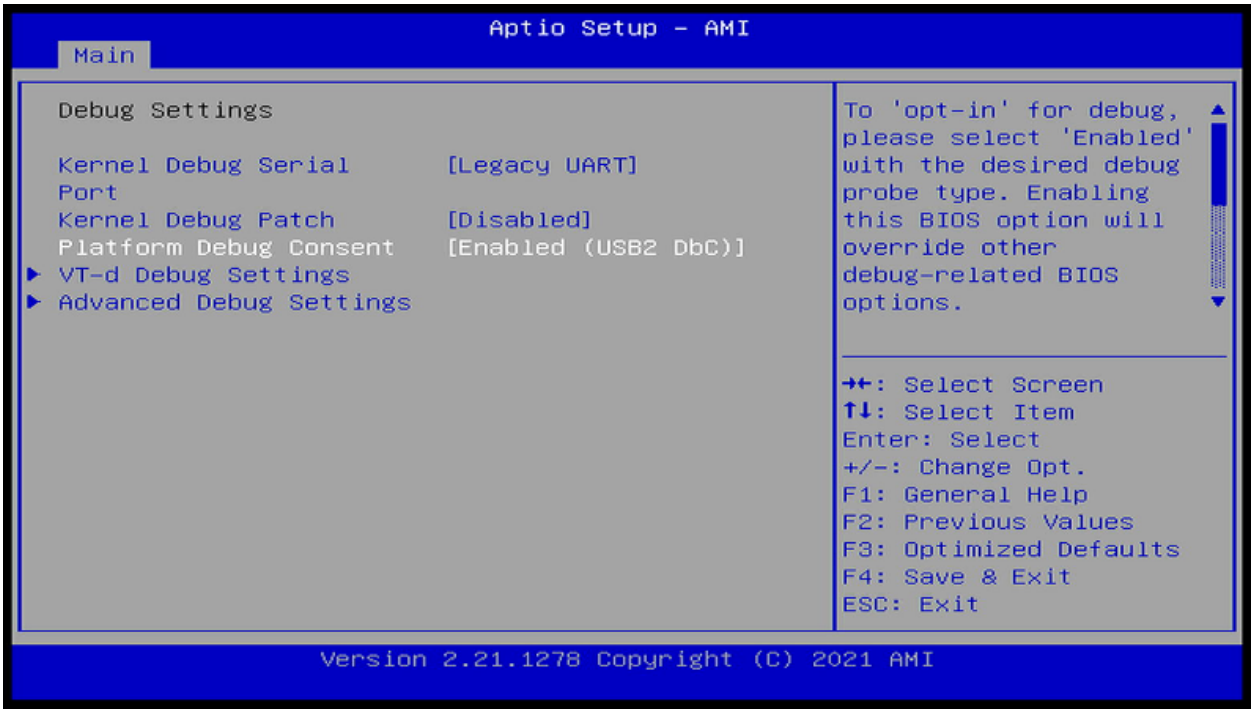


There is another setting within the CRB Advanced menu to Disable the TCO Timer from being re-enabled by Windows; this is set to Disabled by default when shipped from AAEON, but if you run into issues with run-control stability, you might want to check this:

`CRB Setup > CRB Chipset > PCH-IO Configuration > Enable TCO Timer` must be set to `Disabled`.



Finally, go to `CRB Setup > CRB Advanced > Debug Settings` and set `Platform Debug Consent` to `Enabled (USB2 DbC)`:



Remember to do a "Save & Exit" with the F4 key after the changes! Otherwise, your changes won't be saved.

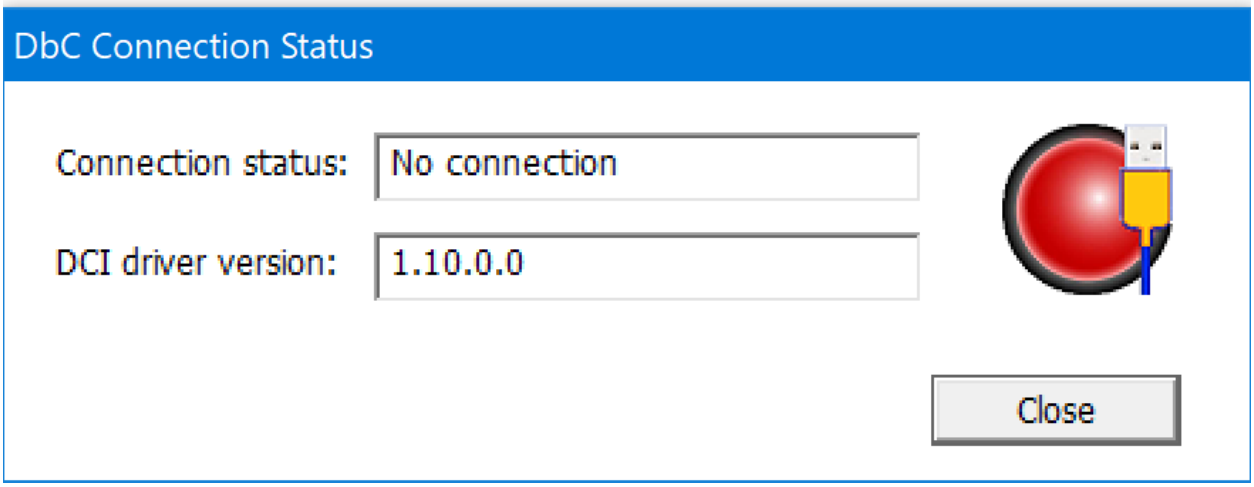
Power Tip: We have observed that, on very rare occasions, the saved settings that you've made will be undone by a "BIOS restore". If you begin to observe strange effects, like autonomous platform resets while in run-control mode, check to ensure that the changes you've made have not been undone.

You are now ready to test your connection, and then launch SourcePoint and begin debugging.

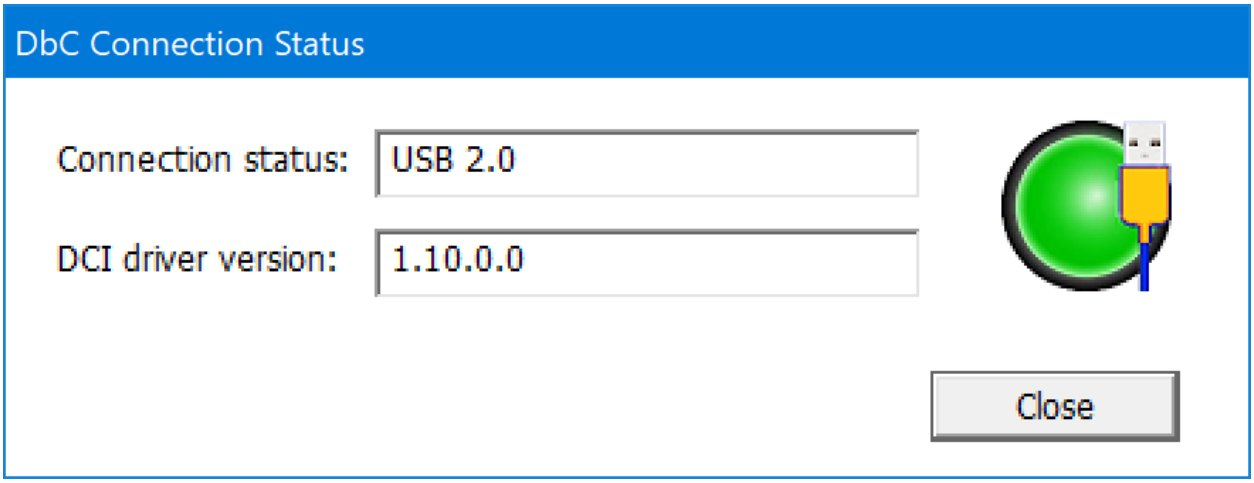
DbCStatus.exe: Red is Bad, Green and Yellow are Good

Luckily, there is a convenient application in the SourcePoint install directory that will tell you that the DCI driver is successfully installed on your computer, and it is possible to make a connection between SourcePoint and the target.

Navigate to `C:\Program Files (x86)\Arium\SourcePoint 7.12.59` (where 59 is your current SourcePoint release), and launch the `DbCStatus.exe`. You should see the red ball, indicating that there is no connection:



Ensure that the Type-C cable is firmly connected to both the host and target, and power up the UP Xtreme i11. In a moment the ball should turn green:



Let the platform boot to the UEFI shell. Congratulations! You have a working DCI connection. It's smooth sailing from here.

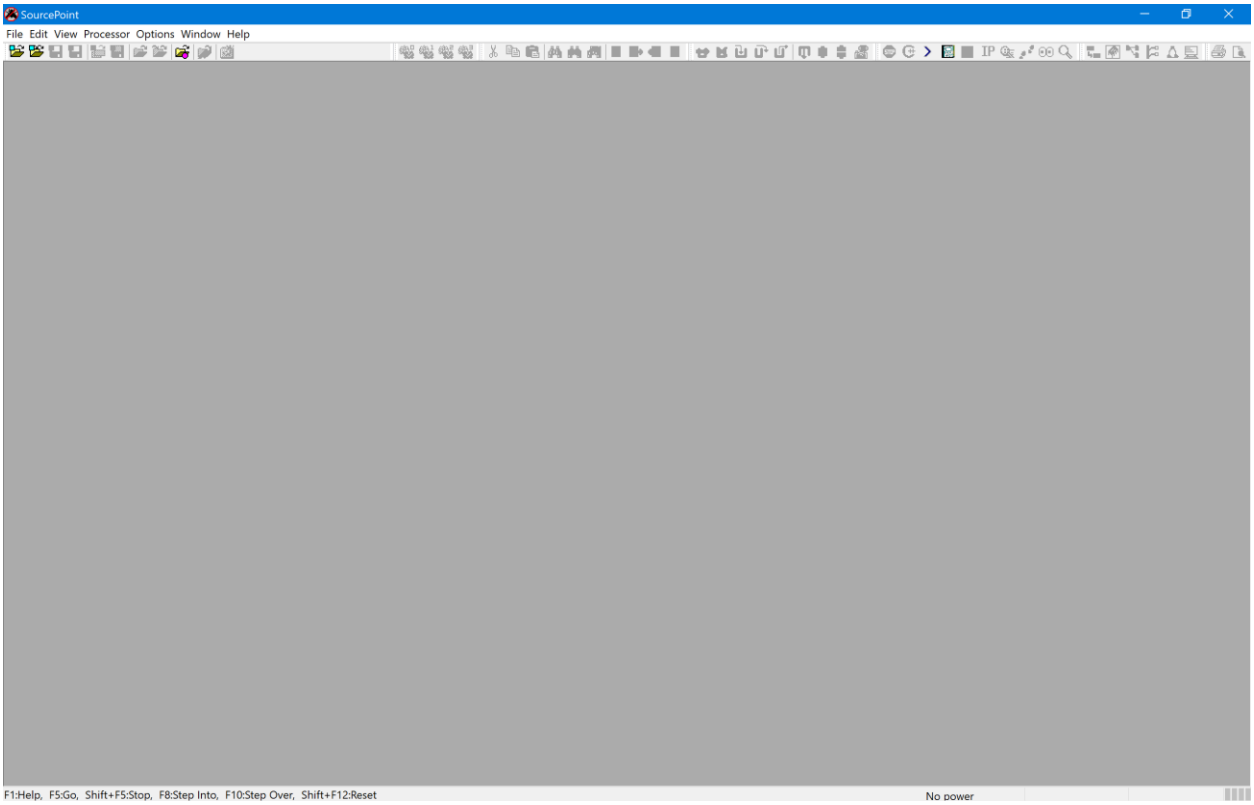
Power Tip: If you are using SourcePoint WinDbg for debugging this board, and have Windows already installed, there may be situations where you want to go to BIOS setup before booting all the way up to Windows. In this case, press F7 after powering on to stop at the BIOS setup.

Power Tip: We have found that some versions of the Celeron board will not power up successfully after a power cycle. **Here's the workaround:**

1. Apply power back to the target. You will see that it doesn't power up, and the blue light does not light up on the power switch.
2. Unplug the DCI cable from the Type-C plug.
3. The board will start to power up. Press F7 and you will stop at the UEFI menu password screen.
4. Hot Plug the Type-C cable back in. Your DCI connection will be restored, and you will see the DbCStatus ball turn green. You may continue your Windows debugging session.

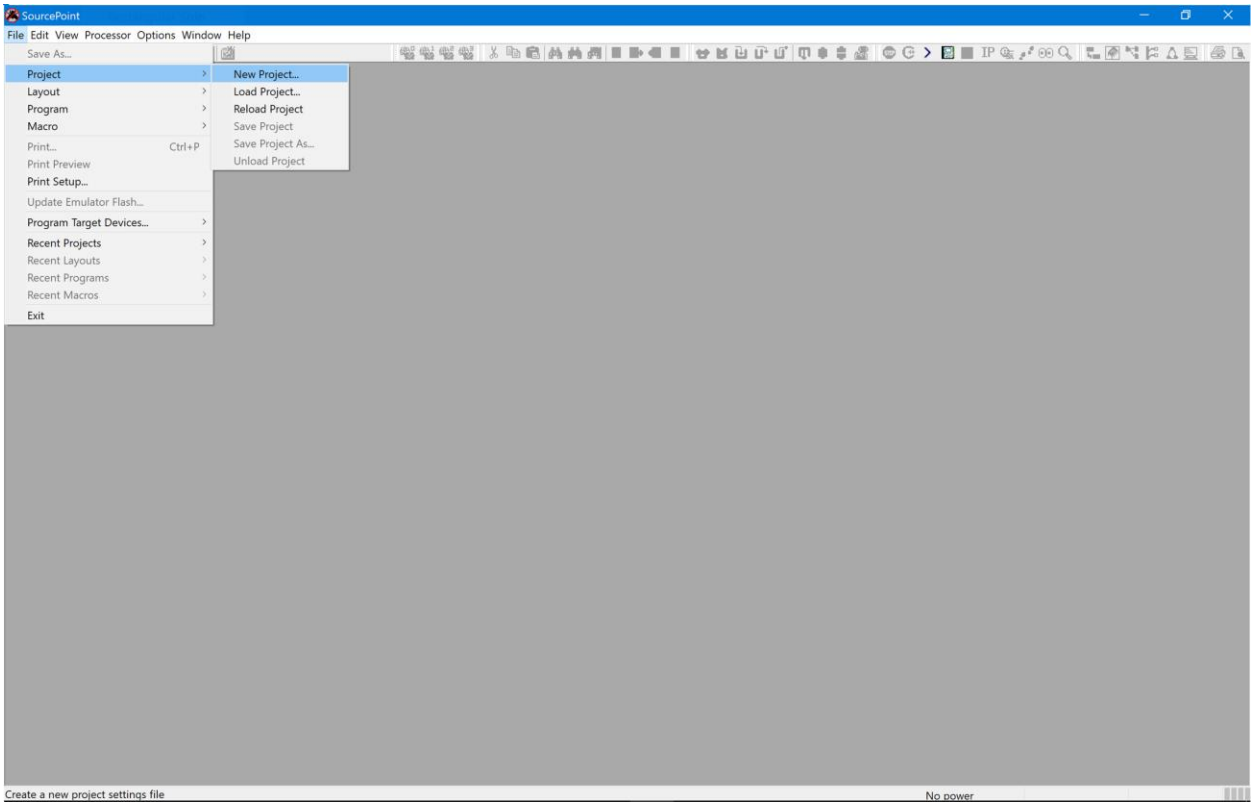
Basic SourcePoint Debugging

When you launch SourcePoint for the first time, you will see the main screen, mostly gray:

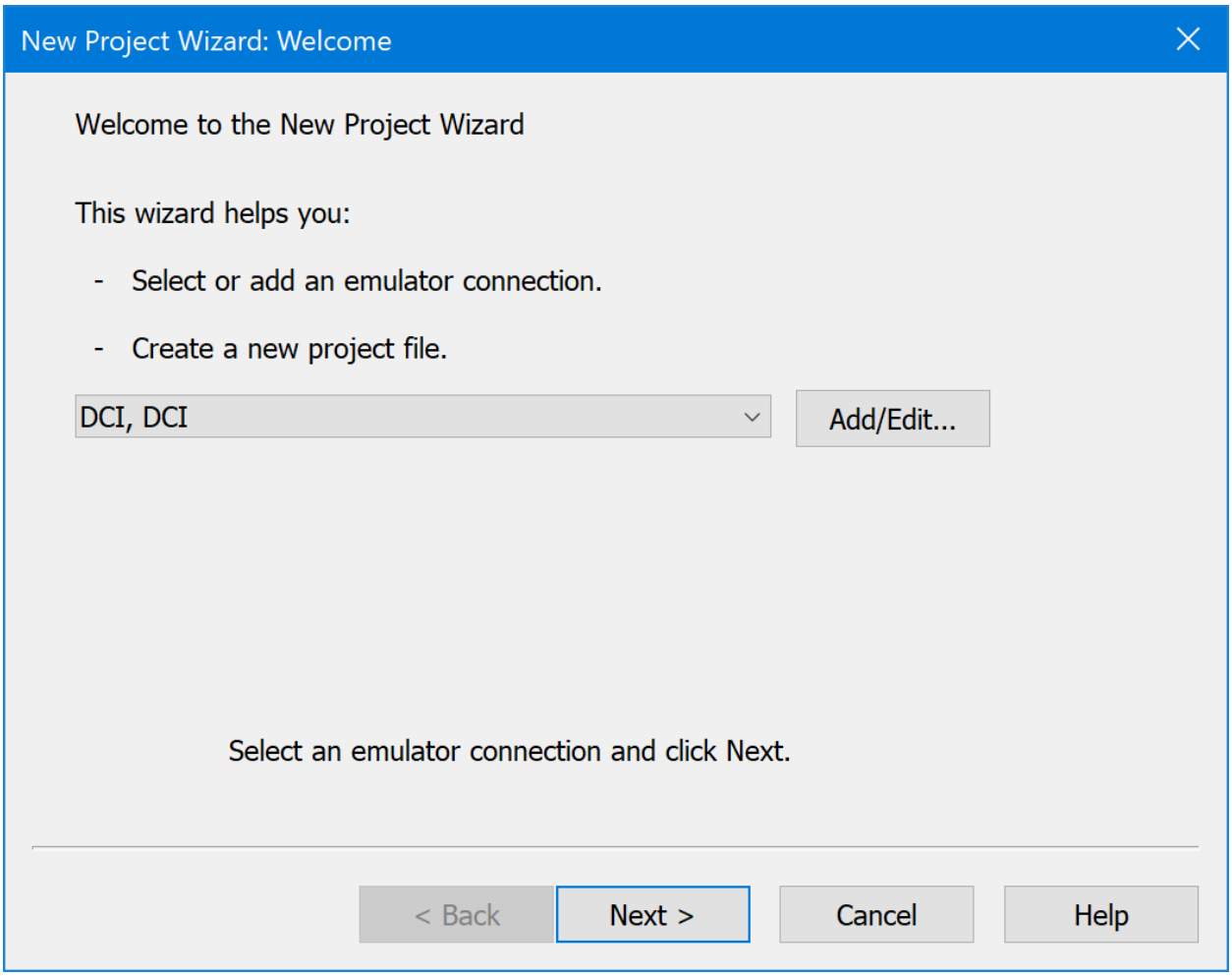


SourcePoint uses Projects (files with suffix .prj) as containers for your debugging session. You can create as many Projects as you want, with all your own preferences saved. Often, once you have the SourcePoint Project configured to your liking, you'll save it and use it repeatedly during your separate debugging sessions. Other users may wish to save a separate Project for each separate debugging session. That's really a matter of user preference and what you're debugging – it's your choice.

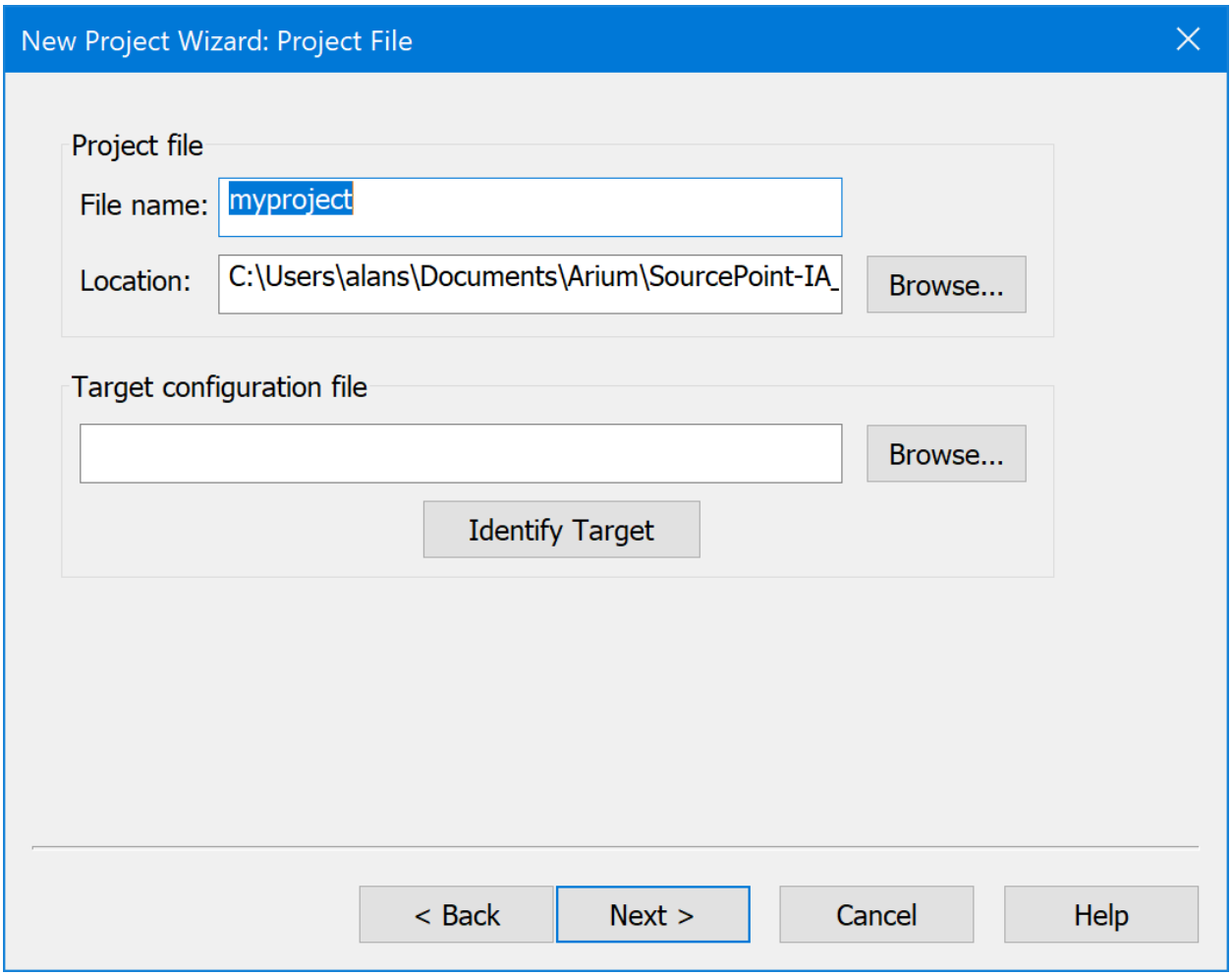
Now it's time to create the Project. Under `File > Project…` select `New Project`:



You'll be presented with the New Project Wizard (NPW). The emulator connection should be via DCI:

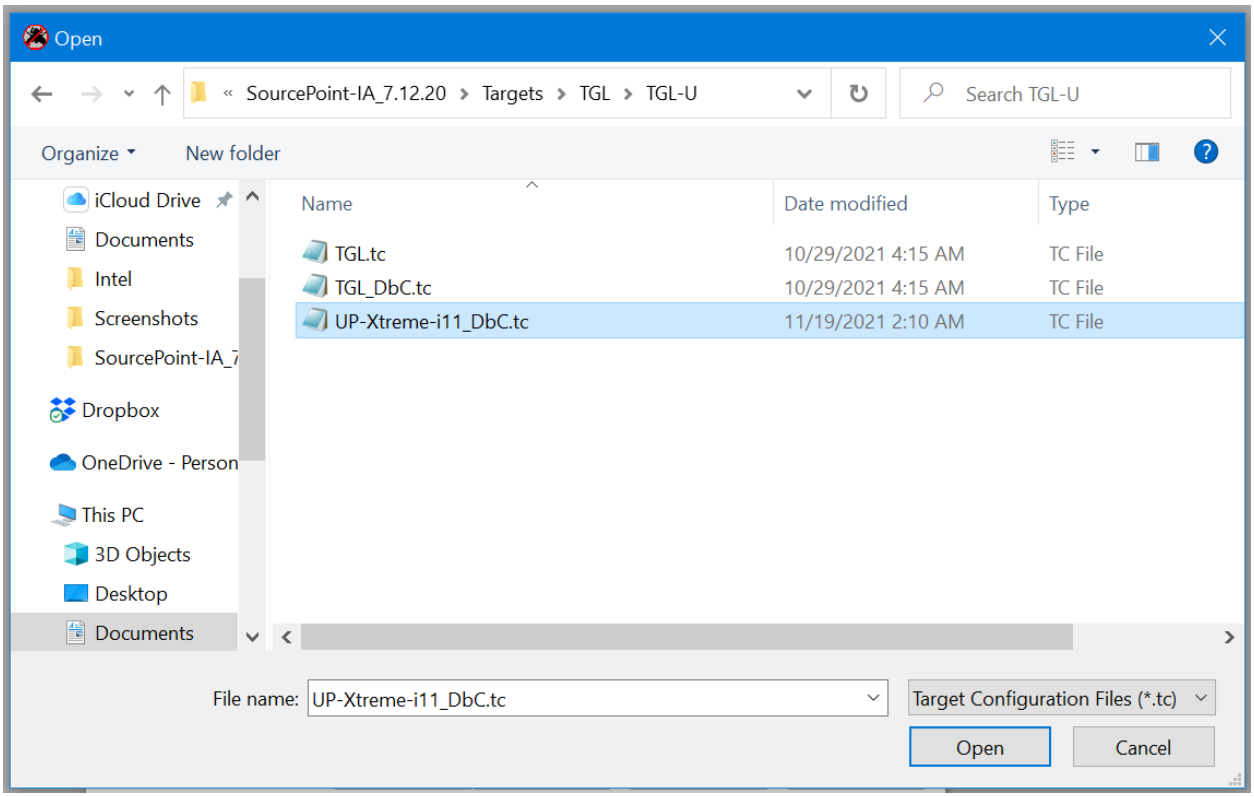


After hitting `Next`, you'll be prompted for the Project Name, Location to store the Project, and the location of the Target Configuration file:

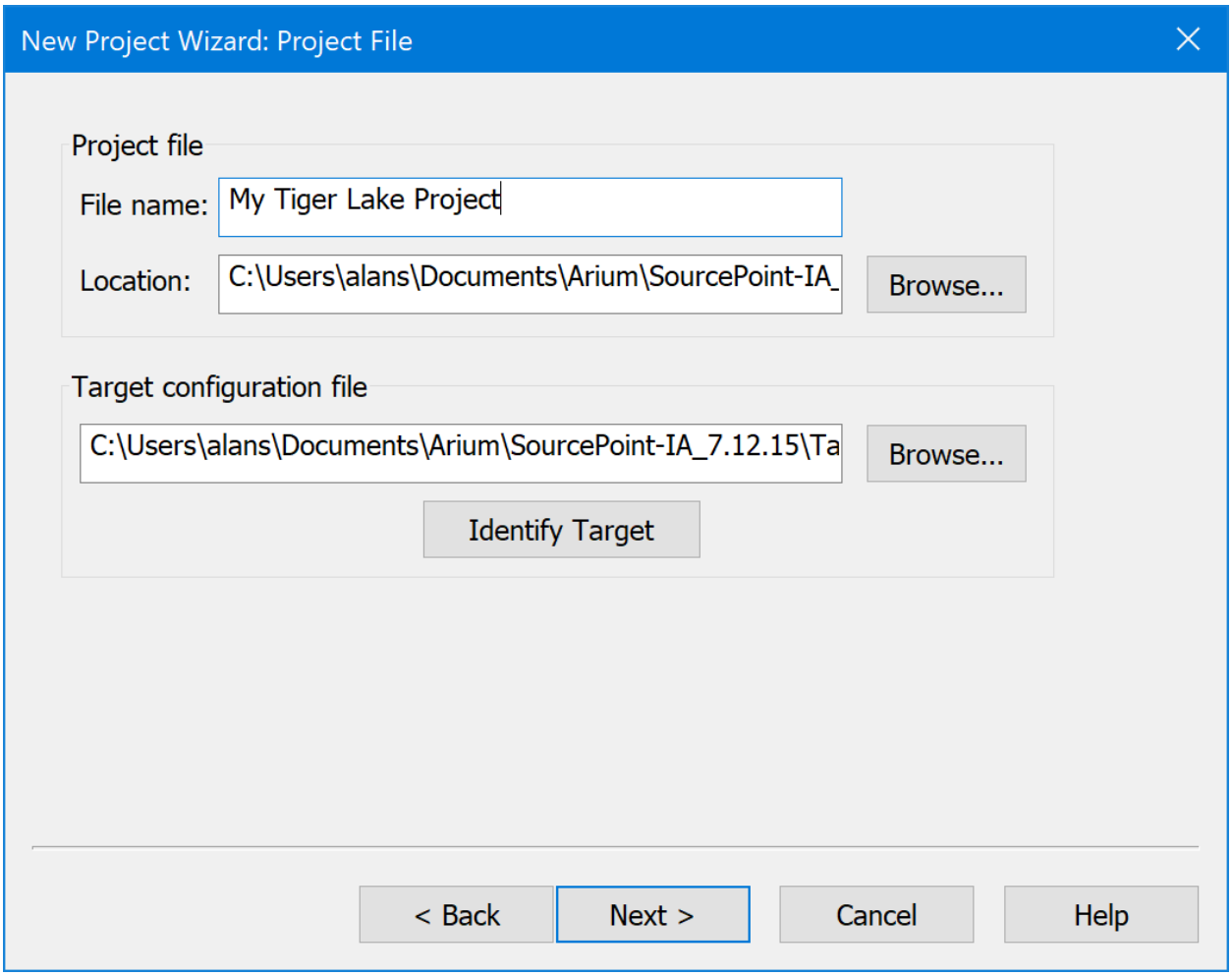


Note that the Target Configuration (TC) files located in `C:\My Documents\Arium\Targets` are used in conjunction with the `jtag-devices.xml` file to define the specific silicon and SourcePoint settings necessary to ensure a successful DCI connection.

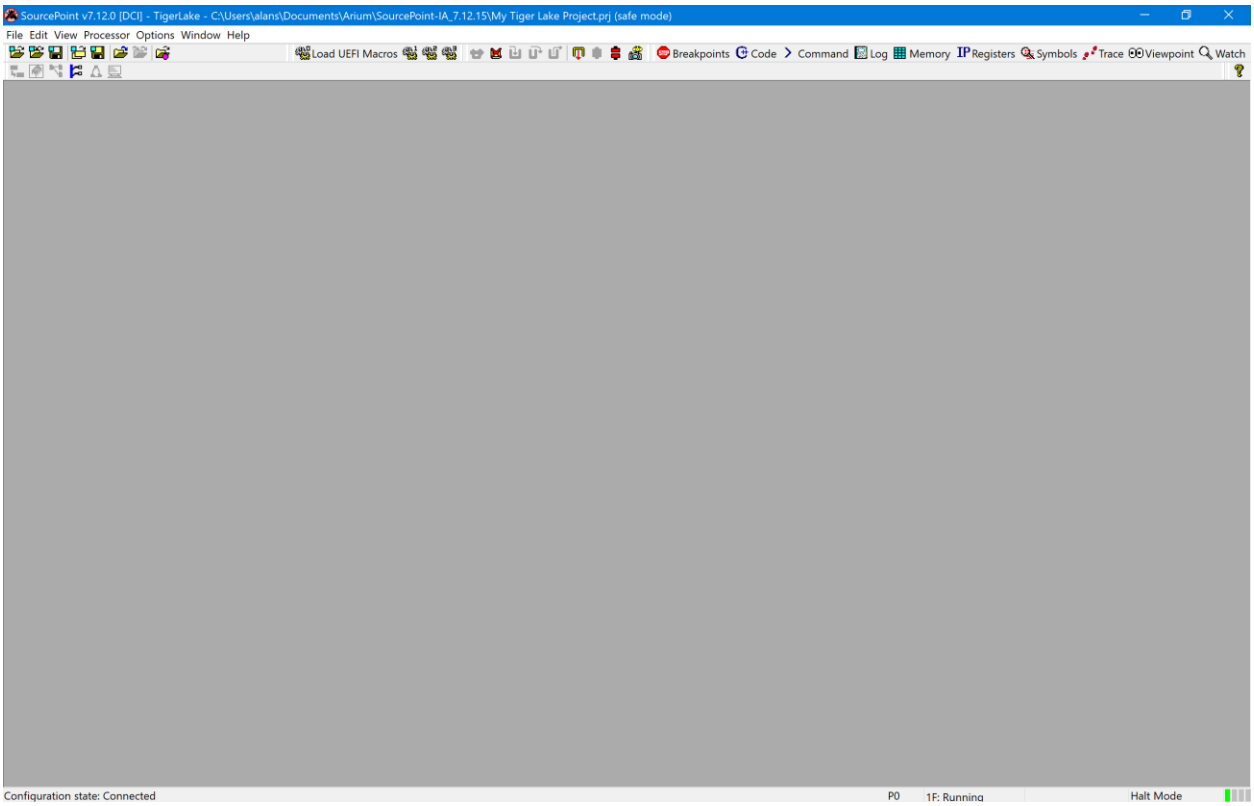
For the UP Xtreme i11 boards, custom TC files have been created, so you shouldn't do an `Identify Target` to automatically select the TC file of interest. Rather, manually select the specific TC file that is customized for this target (`TGL\UP-Xtreme-i11_DbC.tc`) and hit Open:



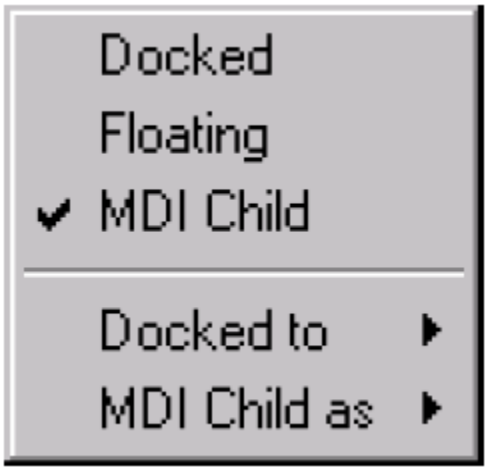
Then, your screen should look something like this, after you've replaced the default Project file File name with your preferred name (in this instance, My Tiger Lake Project):



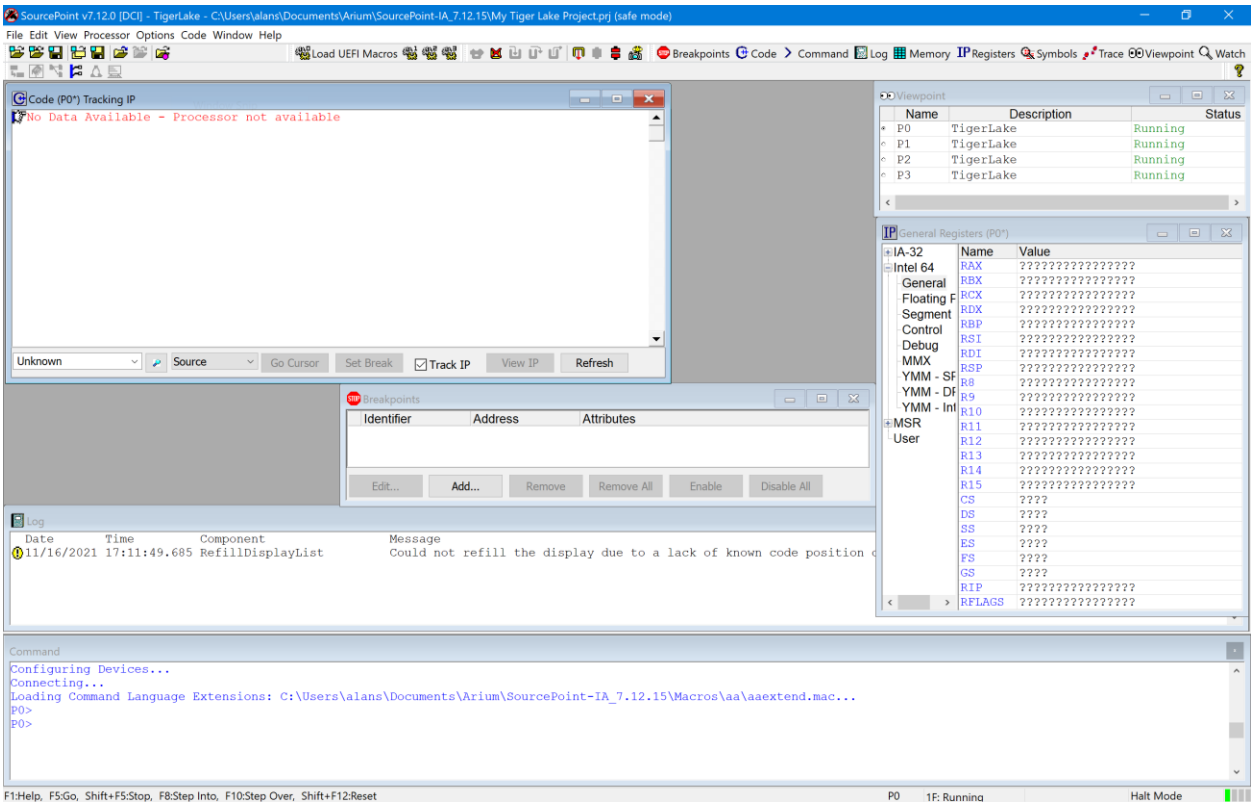
Hit `Next`, then `Finish`, and SourcePoint should successfully connect to the target. You should see "`JtagTest: Successful operation`" followed by "`Configuration state: Connected`" in the Status bar at the bottom left:



Now the fun part begins. Click on the buttons at the top to set up the Viewpoint, Code, Command, Registers and other windows to your own preference. Move the windows around and resize them to take best advantage of your available screen real estate. You can right-click in the title bar of each window to change its type and, for example, to dock the window to the bottom, right side, etc.

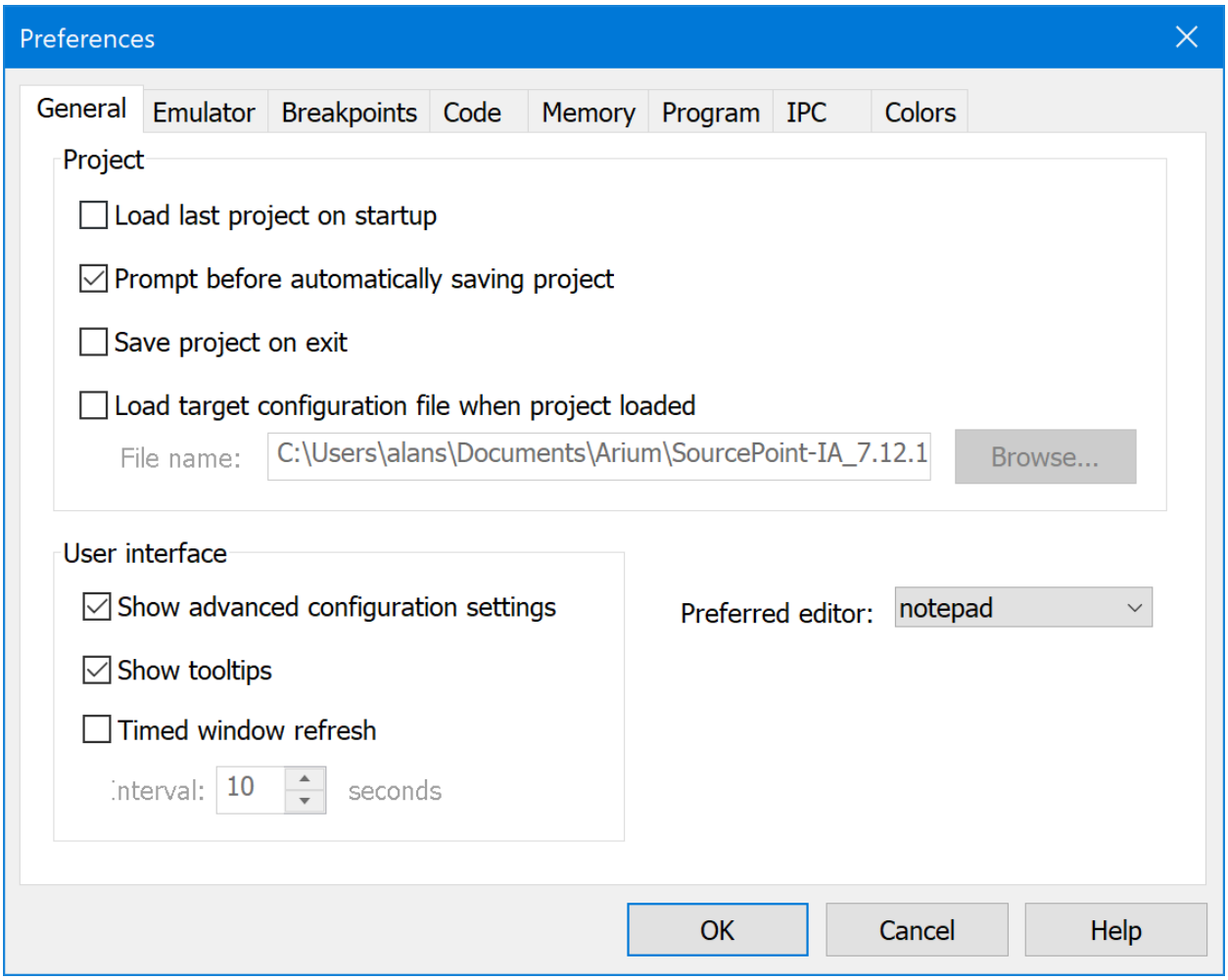


A sample layout is below:



Power Tip: the window layout can be saved as a separate file. That way, when you create a new Project, you can just Load the layout file separately, without having to create and move the windows around again. Choose `File > Layout > Save Layout…` to create a .lyt file, and then do a `Load Layout…` to save yourself time every time you create a new Project.

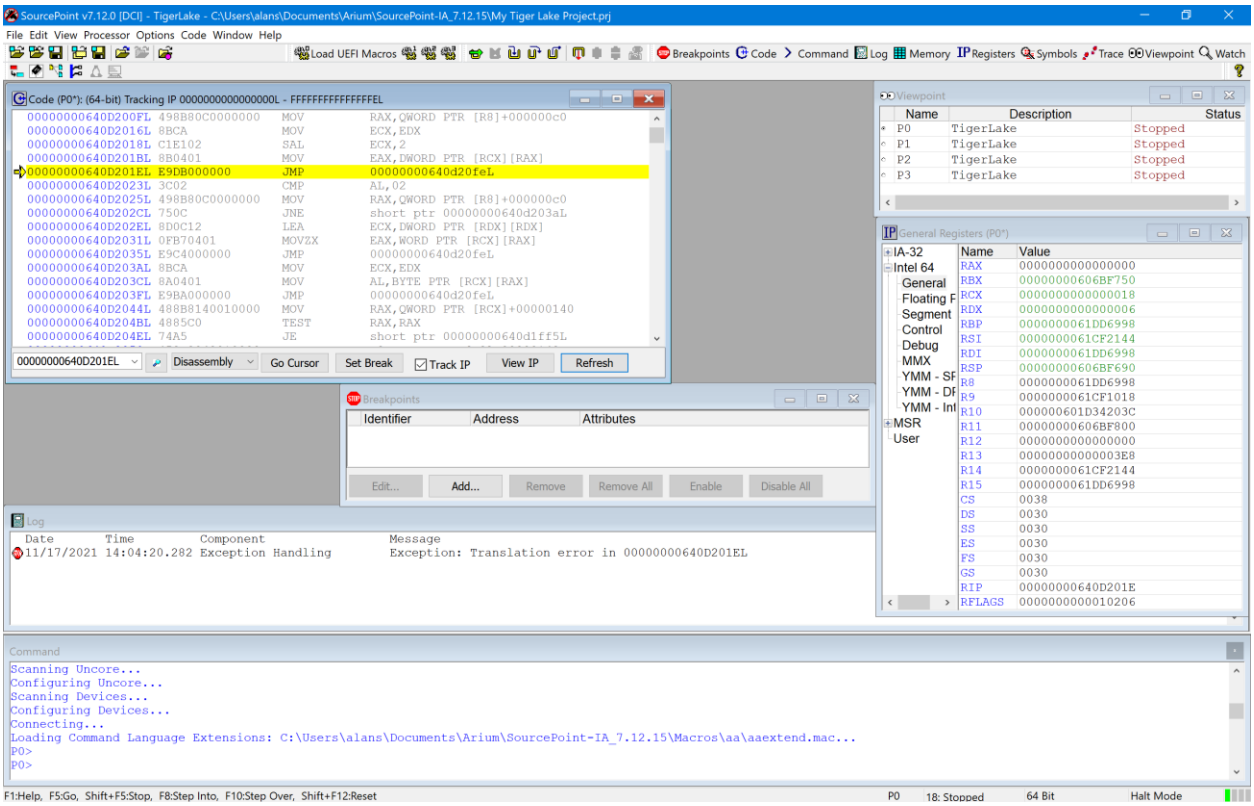
Exploring `Options > Preferences…` and some of the other menu items may give you some more labor-saving ideas. For instance, your workflow might suggest to disable both `Load last project on startup` and `Save project on`. This gives more control on entry and exit from the application:



Another suggestion is to check `Save project on exit`. It's a matter of preference.

This is a good point to do a `Project > Save Project…` That way, you don't have to start all over, if for some reason your project gets messed up.

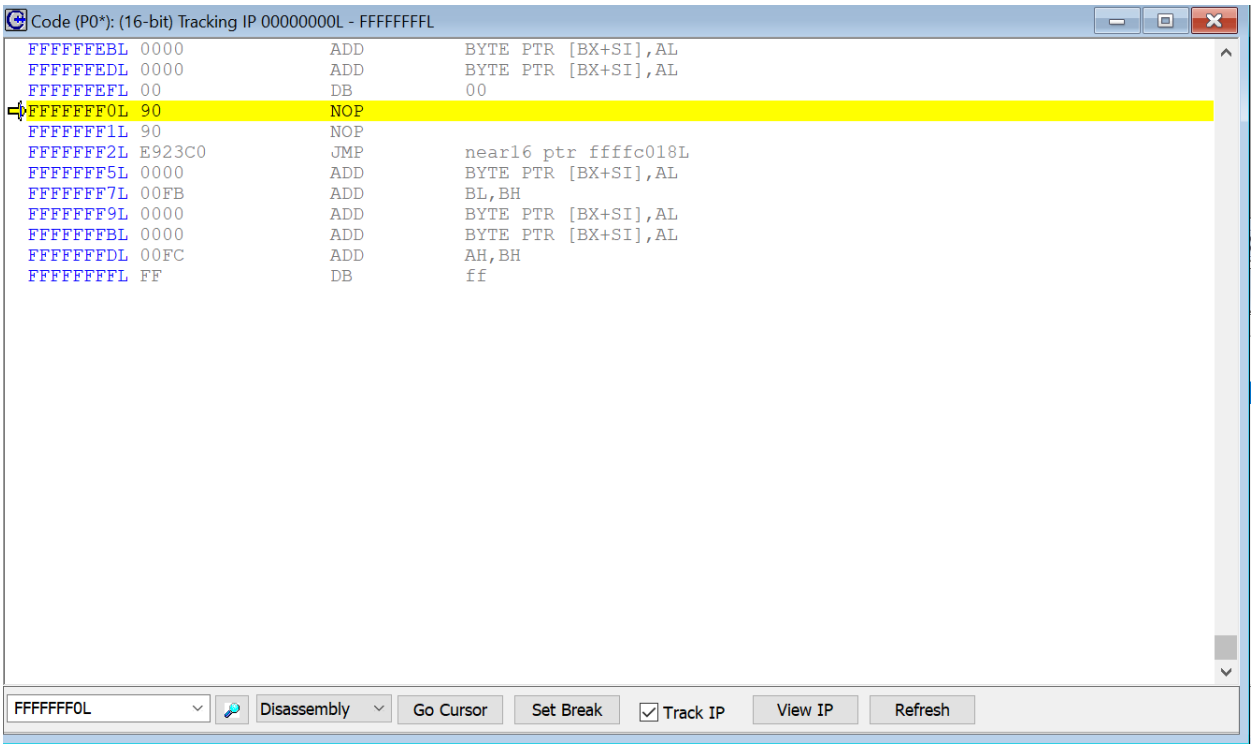
At this point, you are ready to fully begin your debug session. Many of the operations can be accessed via the toolbar at the top of the screen. You can issue a `Stop` on the target, `Step Into`, `Reset` and halt at the reset vector, set some breakpoints, `Go` to the breakpoint, and so on. Use the buttons at the top, in the SourcePoint Icon toolbar section, to initiate these actions. Use the function keys (i.e. F8 for Step Into) as you gain experience.



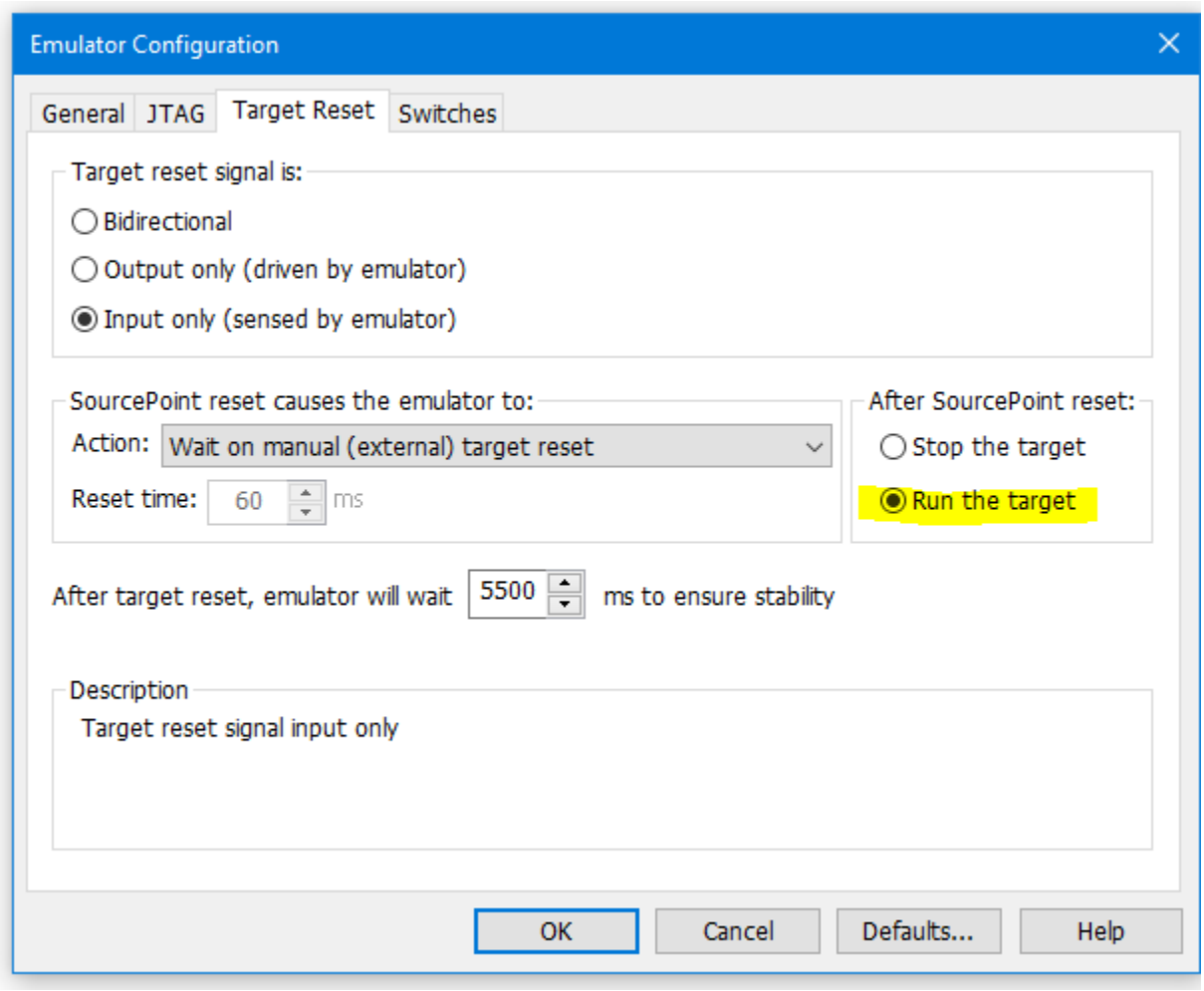
Hit the `Refresh` button in the Code window after the first Stop. This is only necessary once, to get out of Safe Mode; the Code window will automatically refresh with all run-control operations (stop, go, single-step, etc.) afterwards.

Refer to the SourcePoint User Guide in your install directory for detailed instructions on using all of the tool's features.

The board will halt automatically at the reset vector when you hit the SourcePoint Reset button (don't do this yet!):



This is NOT desirable, because some of the AAEON Tiger Lake boards have issues recovering from the reset vector. It is recommended that you disable the default halt at the reset vector by going to the Options > Emulator Configuration > Target Reset and click on “Run the target” in the “After SourcePoint reset” box:



Congratulations, you have mastered SourcePoint's basic capabilities, and are using run-control. Many users are content to just use these basic operations, because run-control by itself is very powerful. However, if you wish to master the product and use some of its more advanced features, read on.

Advanced Topics: Using Trace

Trace is by far one of the most useful debugging utilities for triaging the most difficult, hard-to-reproduce bugs. Fortunately, the Tiger Lake CPU is equipped with all the latest-and-greatest Intel trace logic, and SourcePoint supports them all.

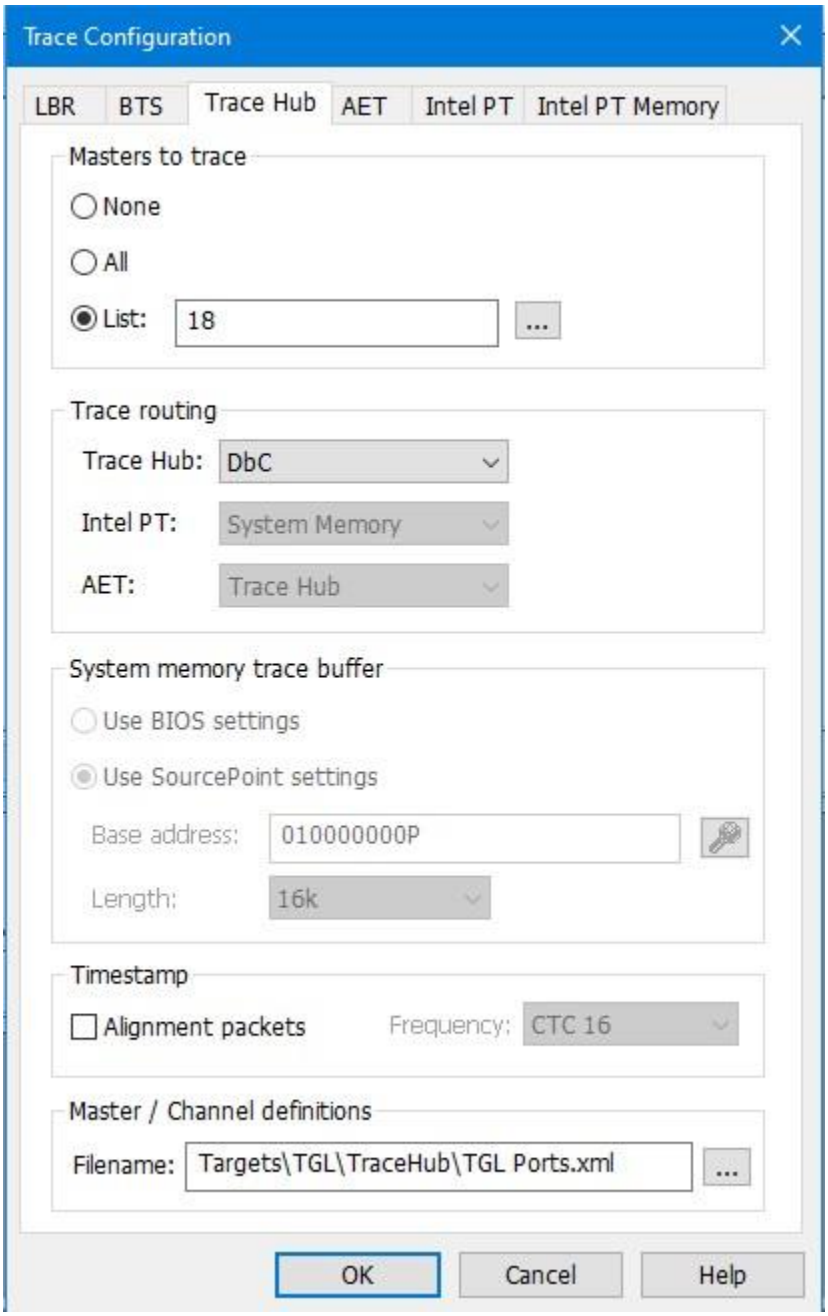
Let's look at a few of them, and how to configure their use in SourcePoint.

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.

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.

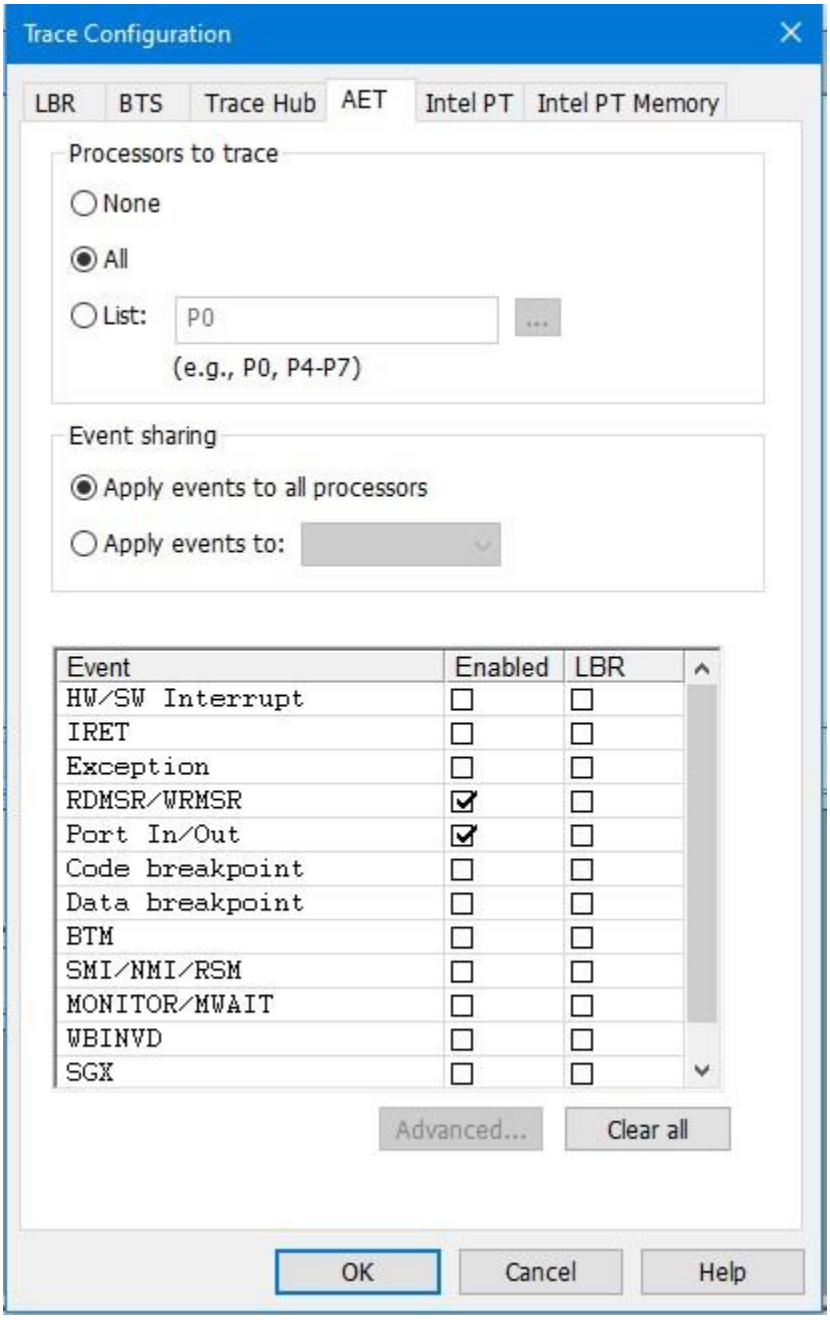
Click on the `Trace` button in the toolbar at the top, to open the Trace window; then click on the `Configure…` button; then click on the `Trace Hub` tab. Set the settings as below:



Once you've configured the Trace Hub this time, you don't have to set it up anymore. You can now take advantage of one of the main capabilities of the Trace Hub, Architectural Event Trace (AET).

Using Architectural Event Trace (AET)

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:

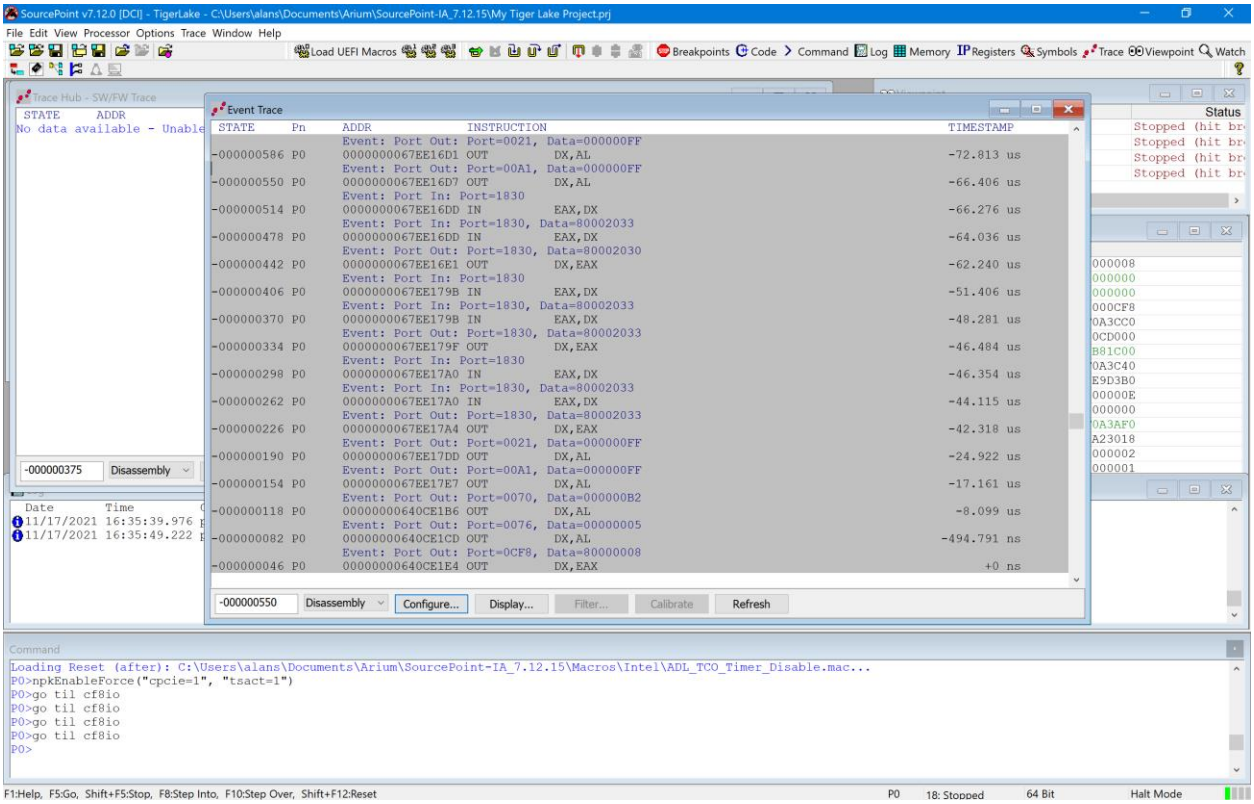


Now, you can simply do a Go/Stop to capture the event trace data. Below shows using the `Command` window to simulate a break on any read/write of port x'CF8', the PCI CONFIG_ADDRESS. This is done by typing the following into the Command window:

```
go til cf8io
```

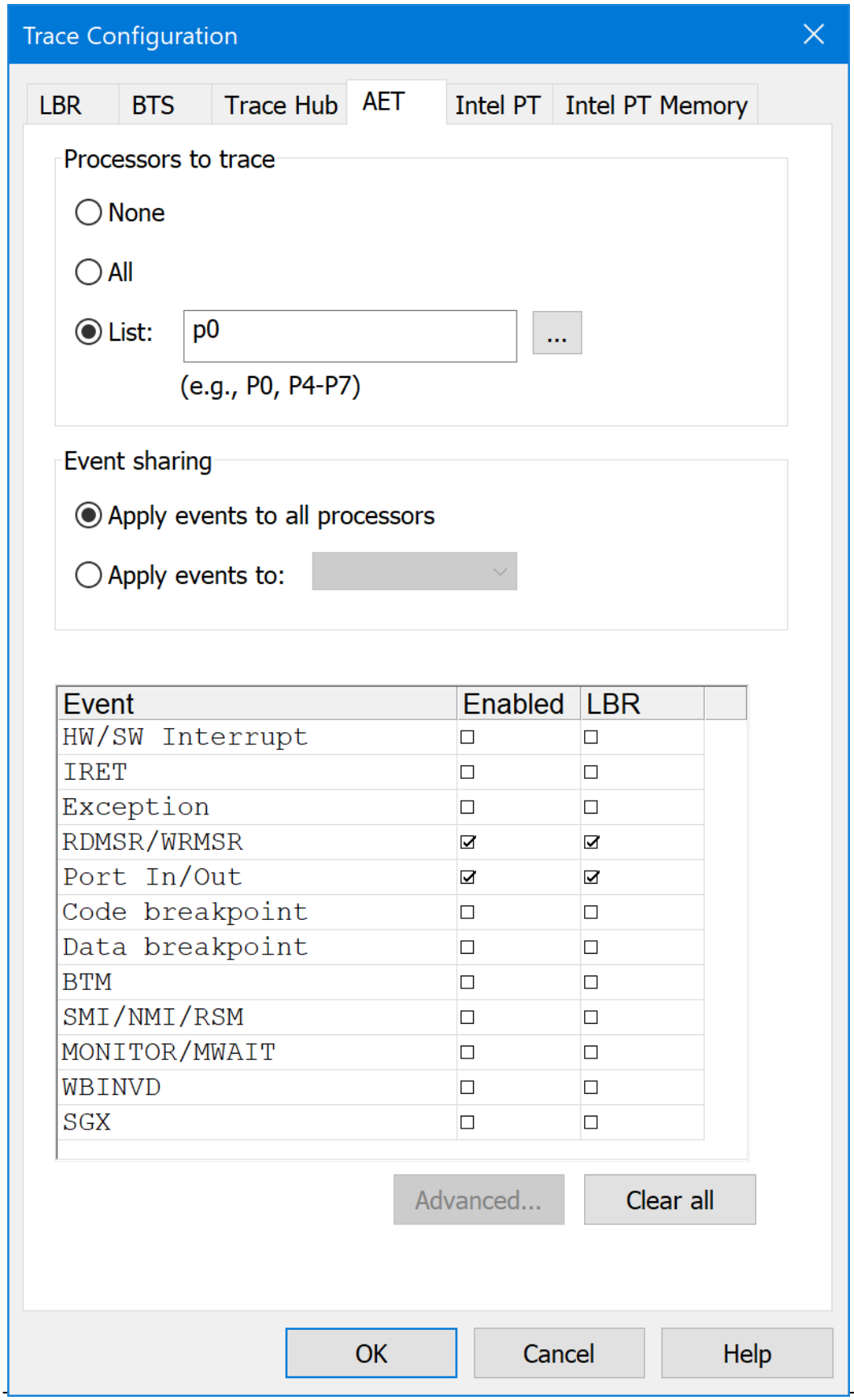
This will run the target until the next IN or OUT to CF8.

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.

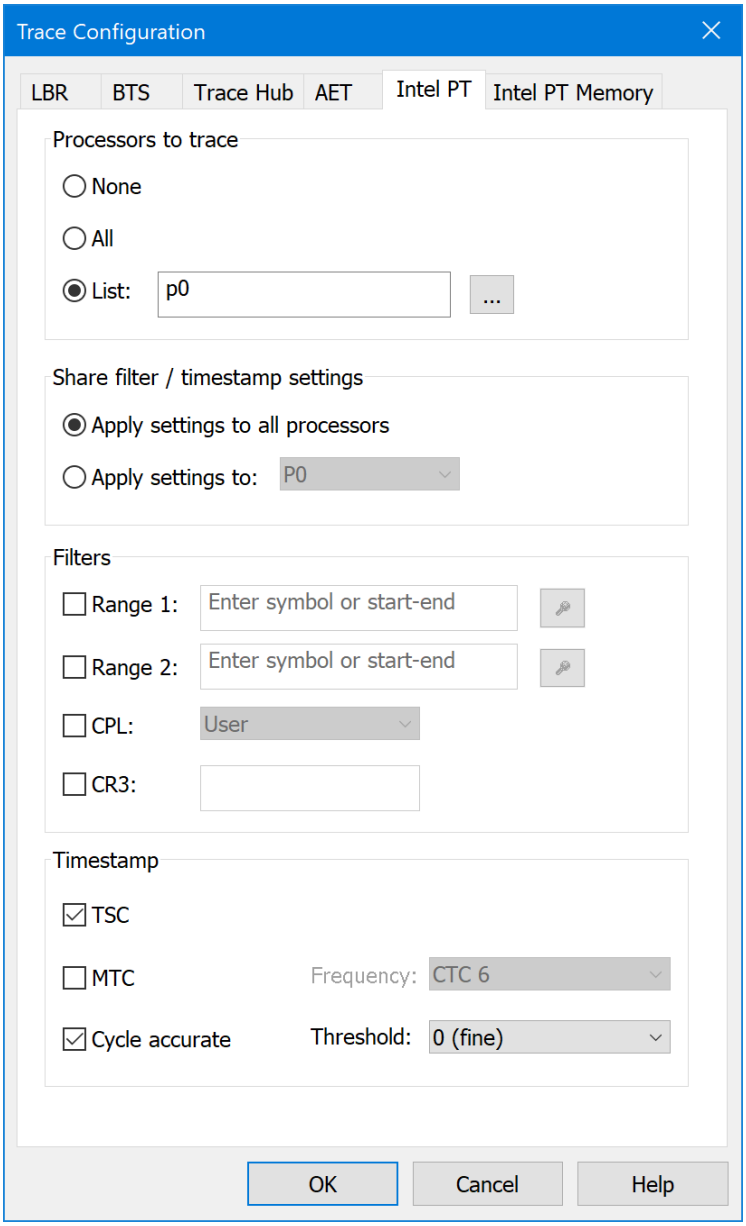
Power tip: The Last Branch Record (LBR) stack associated with each event can be captured as well. This is a very powerful debugging utility, especially when troubleshooting code execution leading up to events before system memory is initialized and Intel Processor Trace is available. Note: If you try to collect too many events and their associated LBR instruction tracebacks, you will overflow the capacity of the DbC2 connection. Scale these back until the Trace Overflows are eliminated.


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

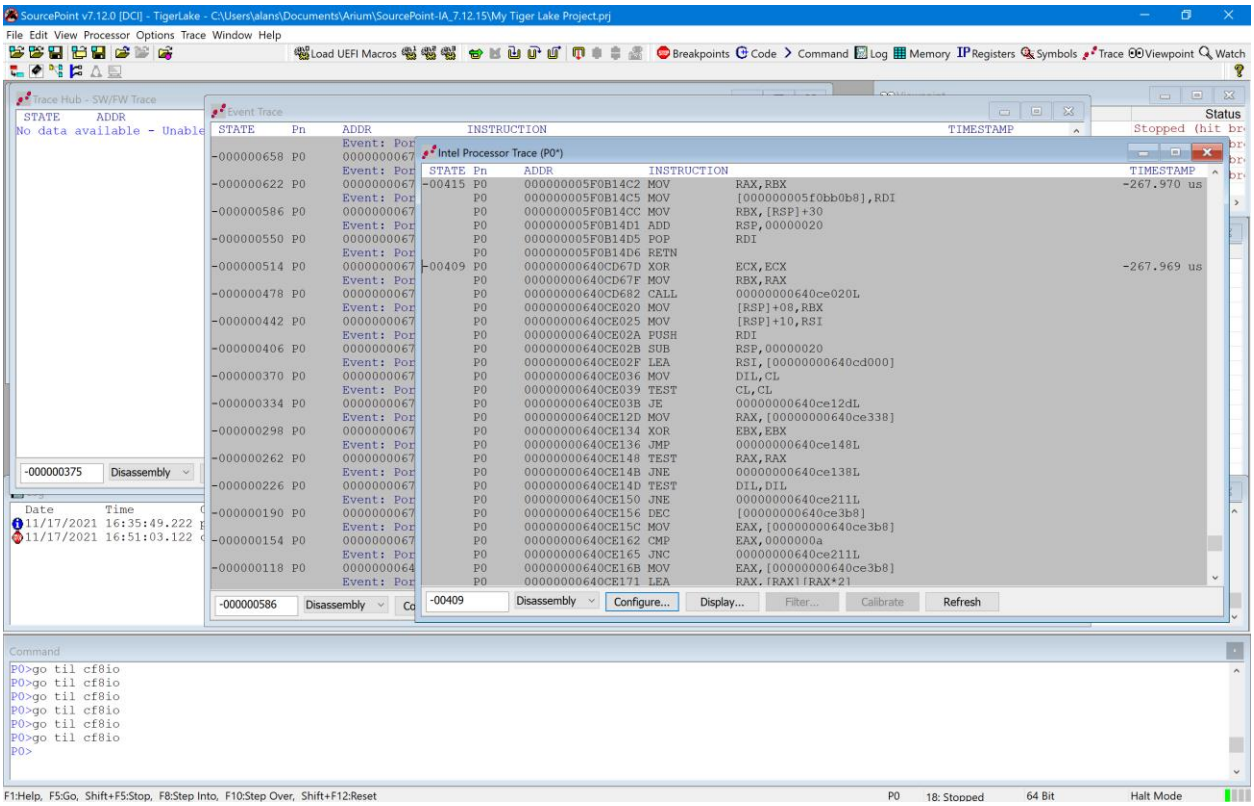
Intel Processor Trace

Intel Processor Trace (Intel PT) is available only after system memory is initialized.

It's easy to set up. Click on the `Trace` button in the top toolbar, click the `Configure…` button, click on the `Intel PT` tab, put `p0` in `Processors to trace`, and be sure that `TSC` and `Cycle accurate` under the `Timestamp` heading are enabled:



That's all. Then use the `go til cf8io` trick to capture some instruction trace data:



There's a lot more that you can see and do with these Trace utilities. SourcePoint can use Intel PT to display a Call Chart and Call Tree. You can open up Code Tracking windows that update dynamically as you walk through the code, showing you exactly where you are and the interaction between code and events. When you have source and symbols available, the firmware flow becomes much more intuitive and visual. Indulge your curiosity and imagination.

The video at https://www.asset-intertech.com/wp-content/uploads/2021/11/UP-Xtreme-i11-Getting-Started.mp4 shows some of these capabilities. The SourcePoint User Guide also provides a very thorough, comprehensive review of the tool. And visit our SourcePoint Academy for helpful "How To" content.

Basic SourcePoint Troubleshooting Tips

At some point, you'll run into something strange. 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, 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 if the issue is repeatable, we ask that you collect as much information as you can, and open a ticket with us at https://www.asset-intertech.com/support/. We'll respond as soon as possible.

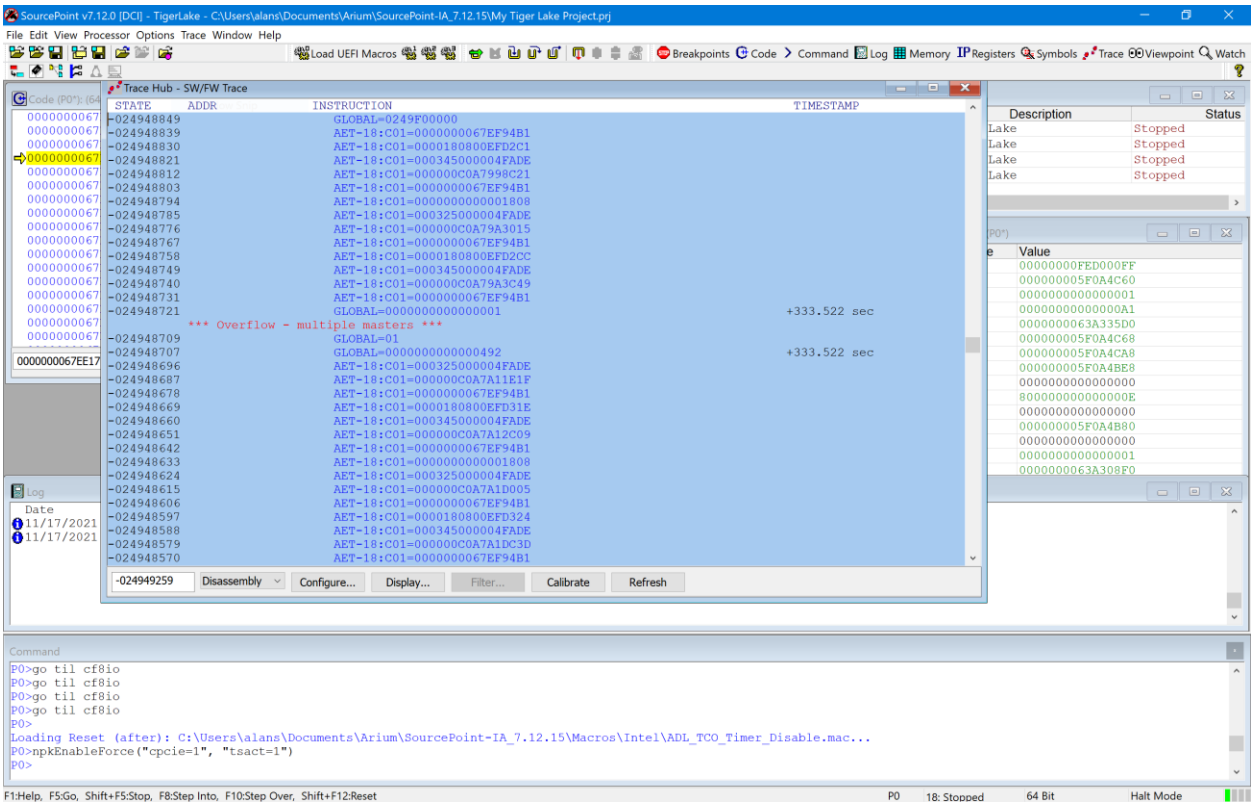
In the meantime, here are a few errata that we've noticed on the UP Xtreme i11, and the steps needed to mitigate.

Trace buffer overflows

DCI traffic processing has its limitations. When you try to collect too much trace data, the trace buffer overflows, causing aberrant behavior.

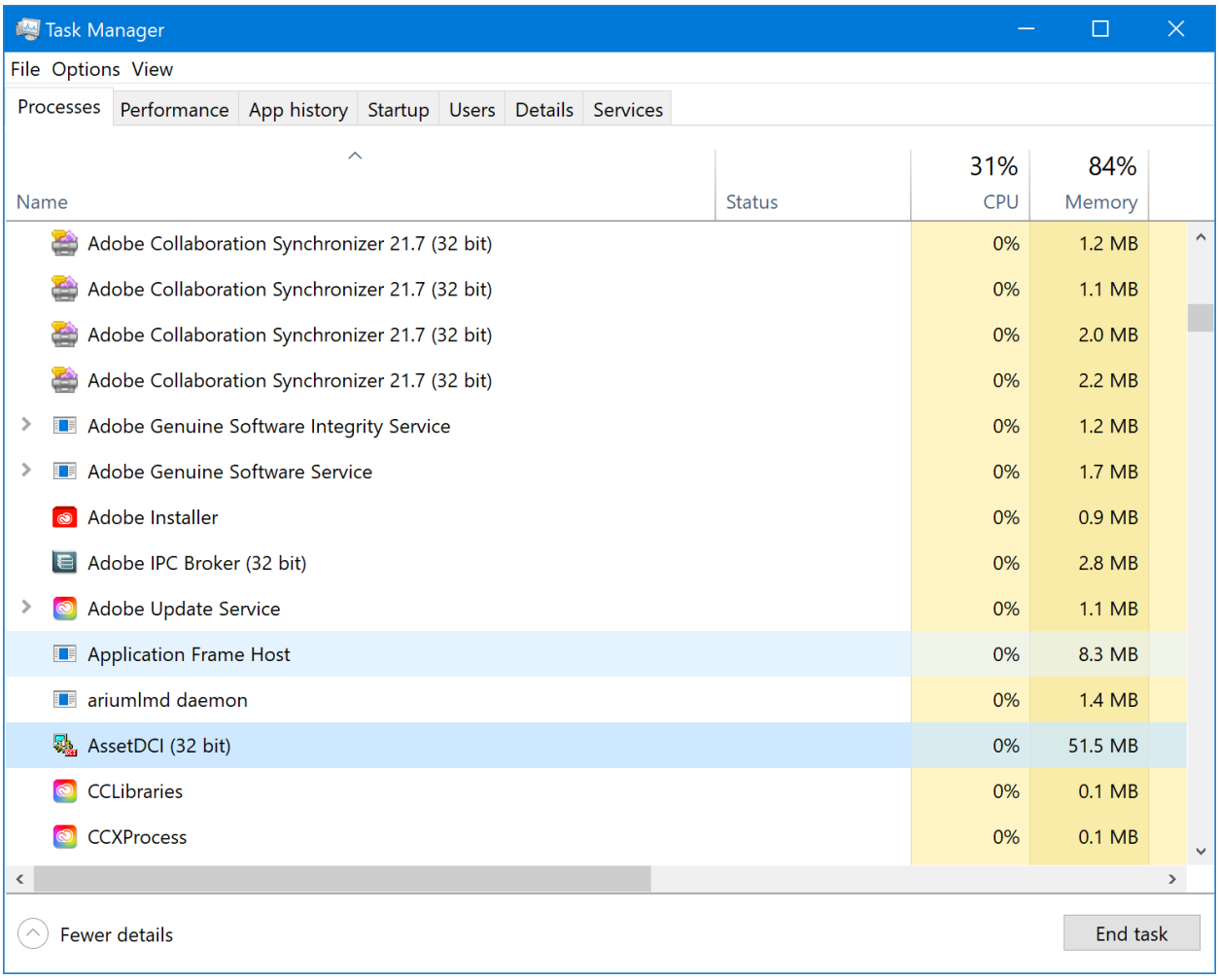
Although the trace data is highly compressed, some trace sources, specifically with AET, running through the Trace Hub can exceed the capacity of the USB 2.0 connection. In theory running at 480Mbps, in practicality SourcePoint can only process trace data at approximately 100Mbps. Beyond that, we collect ~ 20kB of trace data before a buffer in SourcePoint overflows, and we don't recovery gracefully.

You'll see these symptoms of this occurrence in the SW/FW Trace window:



A few of these overflows are no big deal. But, if you're tracing a huge amount of data, SourcePoint may spin, as it tries to process all that data, and deal with the mess. Sometimes, after maybe a few minutes, it recovers. Sometimes, you end up in limbo.

The only solution at this point is to quit SourcePoint, do an `End task` on the AssetDCI Background process, power cycle the target, and start over:

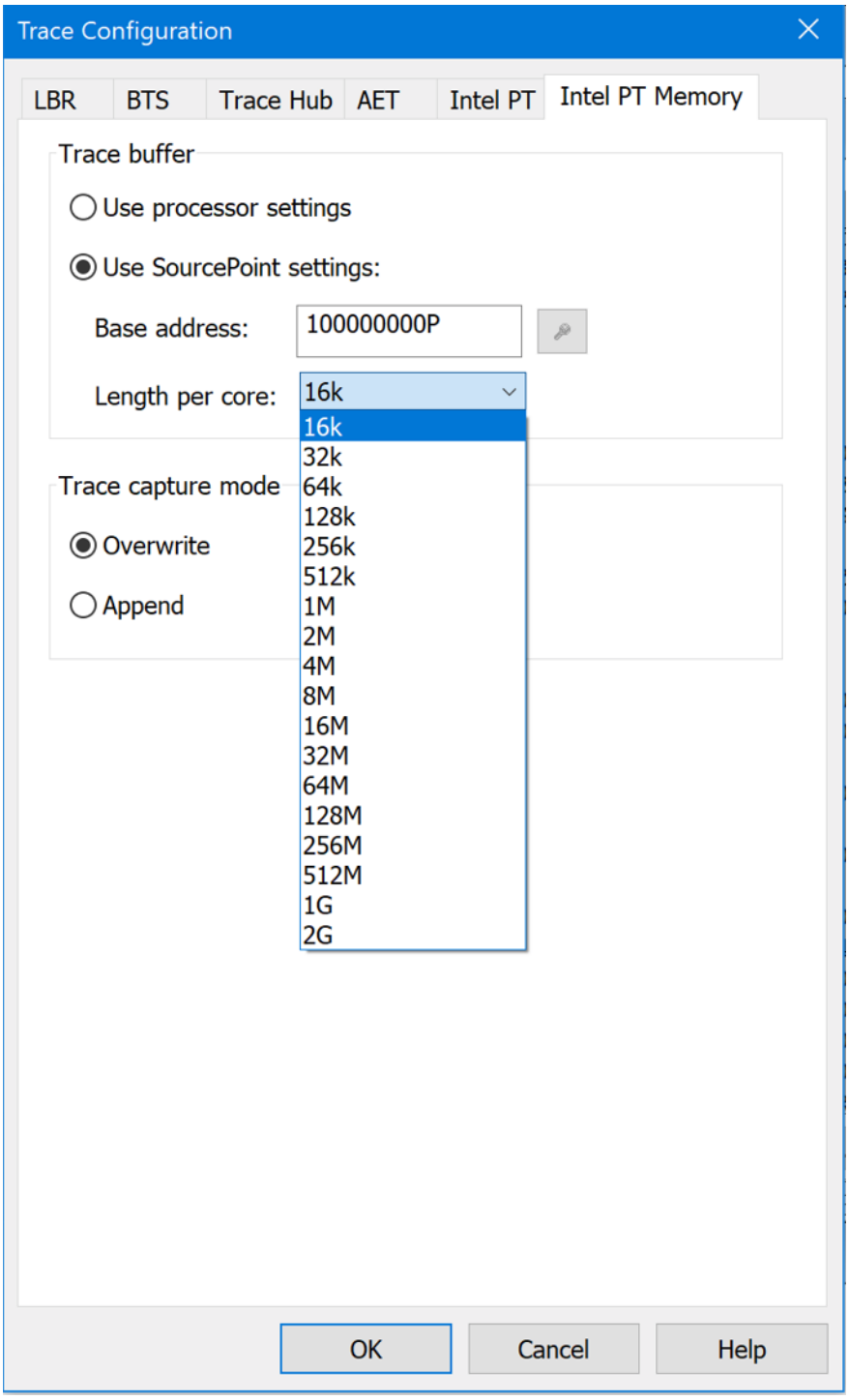


Ultimately, we're working on improving the performing of the AssetDCI driver, and behaving more graciously when an overflow is encountered. But, in the interim, it is key to ensure that the trace data collected is relatively "sparse". Focus your debugging in the specific area of interest. Don't try to collect all Port IN/OUT from the reset vector all the way through to the UEFI shell. At 750K I/Os per second, you'll swamp the host debugger processing, and you can't deal with all that data anyway.

Note that this only happens with AET – that is, you can only overflow by collecting too much AET data. It does not apply to LBR, SW/FW Trace, SVEN, or Intel PT. It may take some trial and error to limit the scope of your event data collection.

Intel Processor Trace can be slow

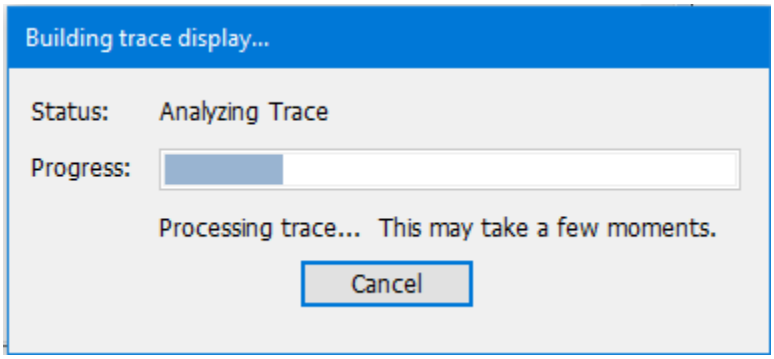
When you configure Intel PT, you can specify the size of the buffer in system memory to collect trace data:

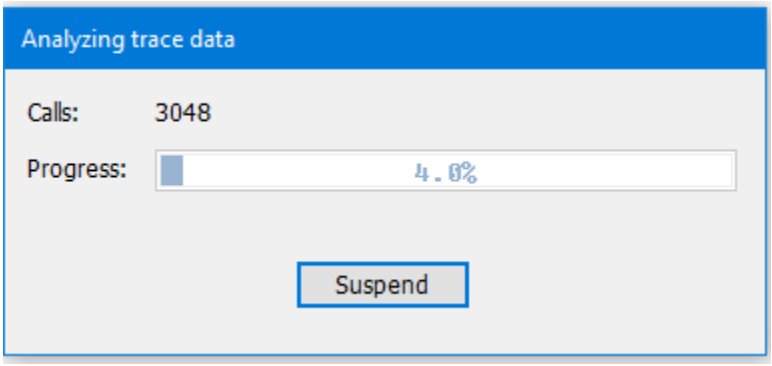


Note that Intel PT directs instruction trace data to system memory. Although highly compact and efficient, we are not streaming over DCI to the host in this case; we buffer the code execution data until the platform is halted, at which point SourcePoint uses JTAG over DCI to collect the trace data out of system memory, reconstruct and display

it. JTAG operates at fairly low speeds, and for large buffer sizes the transfer of all that data can be slow. It starts to become noticeable beyond a buffer size of 64kB.

SourcePoint compensates for that by only pulling in the trace data that's local to the code display you are in. Scrolling up in the Intel PT window will prompt SourcePoint to pull in more trace data, and display it. Be patient when this happens. If you try to collect 1GB of data from system memory over JTAG, it will take a long while for this all to be available. Displaying the Call Chart for this much data will take a while. You'll see this after initial trace data collection (that only takes a few seconds), but scrolling to the top of the Intel PT window with a 1MB trace buffer (which gives you ~ 300ms of execution trace data) takes ~30 seconds, and then doing a Call Chart takes much longer. Be patient.





My board is not booting – what now?

Once in a while, especially during an intense debug session, we have found that the target goes into la-la land. You get to the UP splash screen, and then it just stops. Or the screen stays black. SourcePoint run-control continues to work, but it won't boot all the way up to the UEFI shell. Quitting SourcePoint, unplugging the DCI cable, killing the AssetDCI process, knocking out the AssetDCI process – all are good steps in this instance, when you need to recover.

But then, once in a while, you wait the needed 20 seconds; and it doesn't boot. The screen stays blank or frozen. We know that happens with the newest Celerons, and there's a workaround that involves unplugging and reconnecting the DCI cable, but this might not be the issue.

Don't panic! For reasons we're not sure of just yet, after about 60 seconds, the board "wakes up" and should boot all the way to the UEFI shell.

Clearing the CMOS on the target has also been known to help when it still won't boot up. There's only one thing: it has gone back to the factory settings, so you'll need to reset the WDT timer, and perform the other steps, as per BIOS Settings section in this manual.

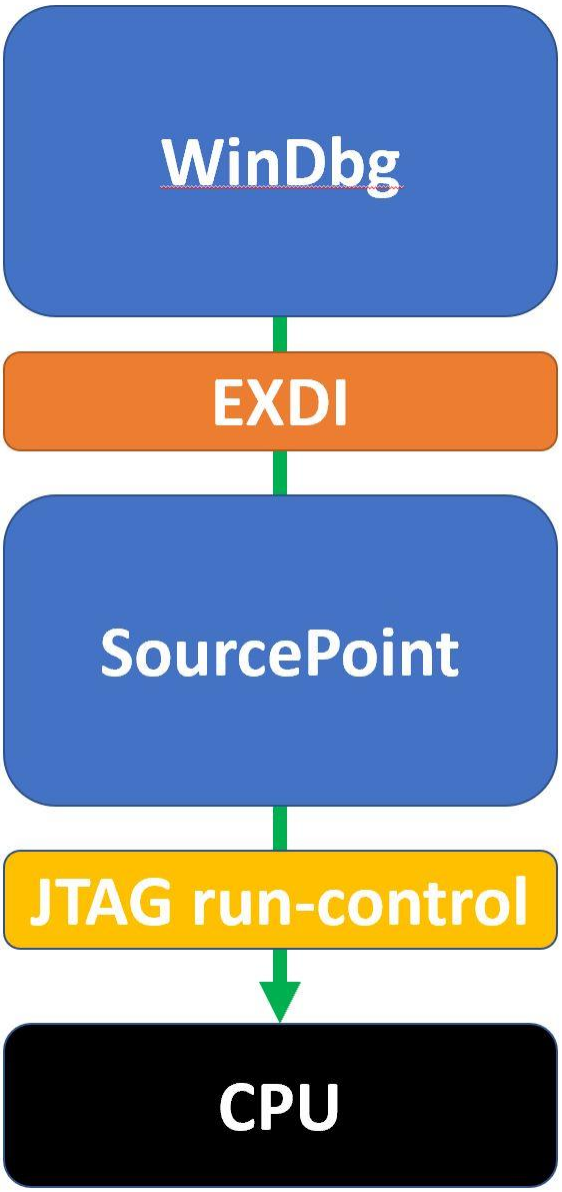
Then, you'll be back in business.

SOURCEPOINT DEBUGGING FOR WINDOWS

Once you've configured the target and become familiar with the use of SourcePoint for basic UEFI debugging as described above, you can begin debugging Windows in earnest.

This section describes the use of SourcePoint WinDbg on a target with Hyper-V disabled. For debugging of Hyper-V, skip ahead to the section on SourcePoint Debugging for Hyper-V.

SourcePoint has implemented a connection to Microsoft's WinDbg application by using a public interface known as EXDI. 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.

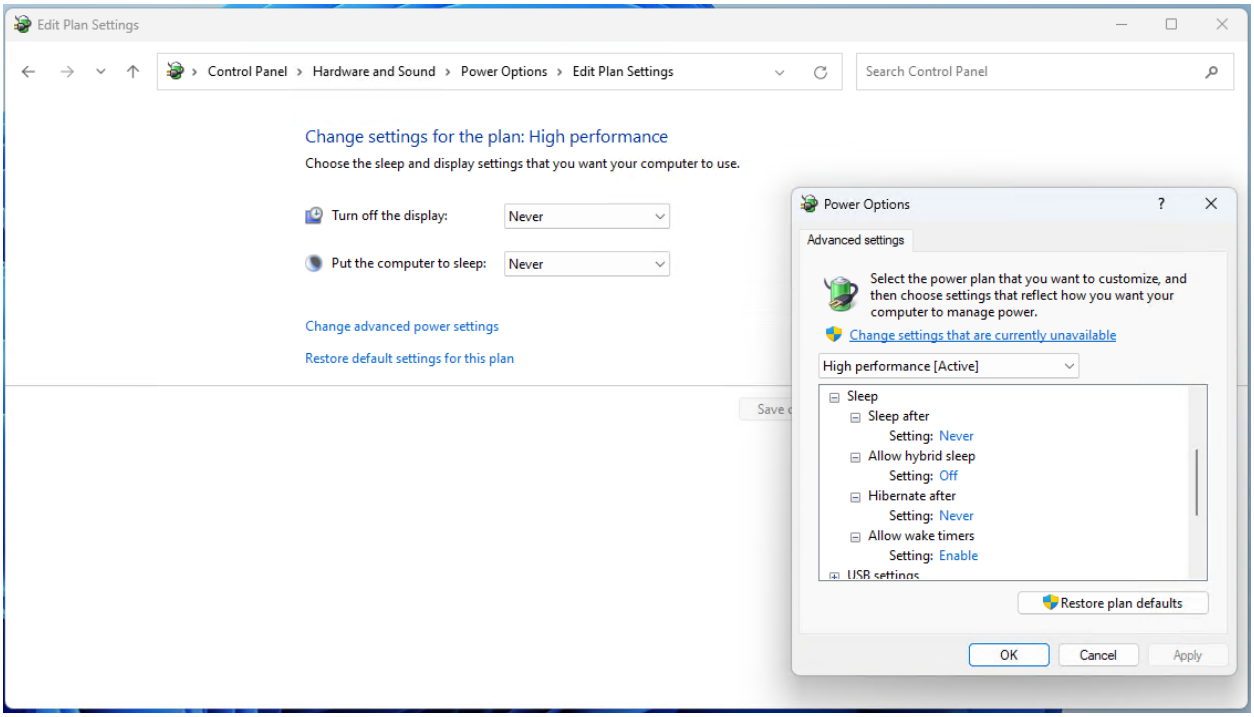
Before we get started, the target needs to be configured to not interfere overmuch with JTAG-based run-control. Then, the steps needed to set up a debugging session will be covered.

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

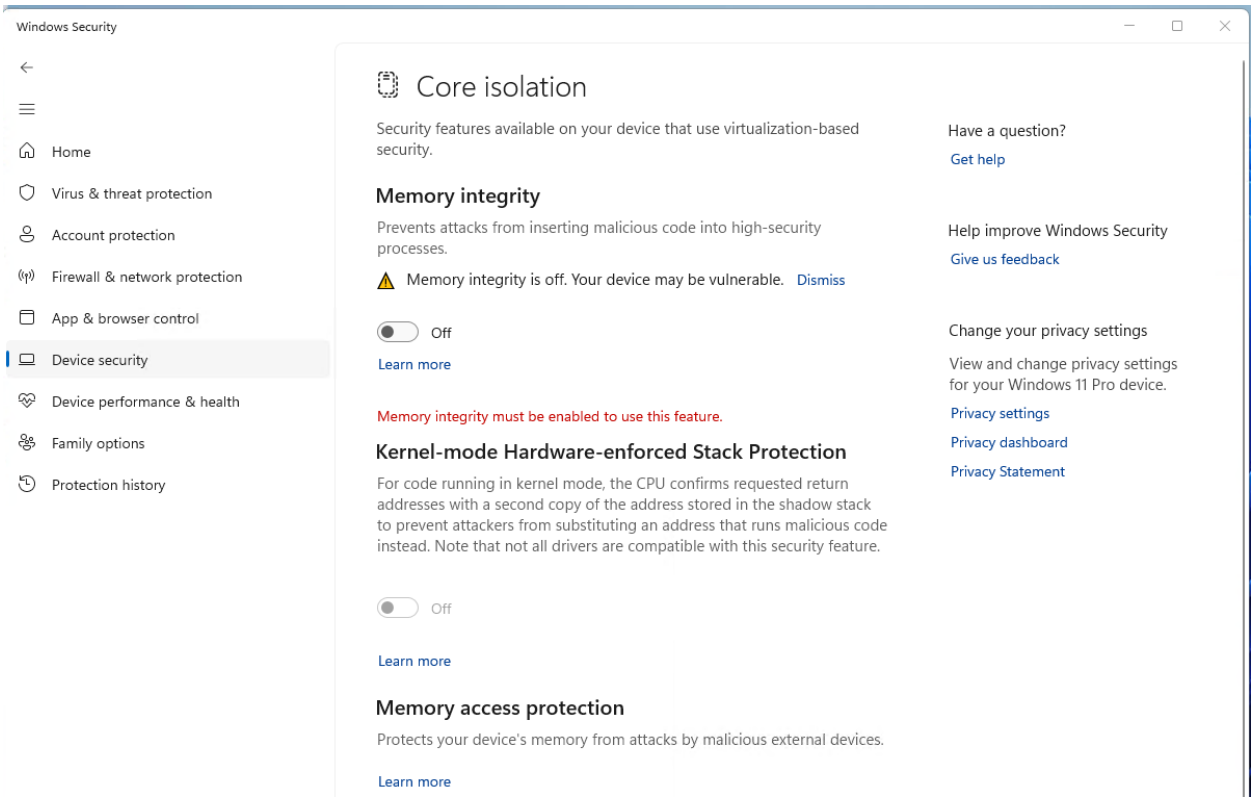
We first need to prevent Windows from changing power states from disrupting run-control prematurely, and VMX and VBS need to be disabled.

These steps are highly recommended (as of the time of writing) to have a successful initial debugging session, especially for newcomers to SourcePoint WinDbg. ***Once experience with SourcePoint is gained, Hyper-V can be turned back on again***.

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":



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.

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

1. Connect SourcePoint to the target
2. Start WinDbg via a SourcePoint macro
3. Load symbols with the LoadCurrent macro.

Step 1: Connect SourcePoint to the target

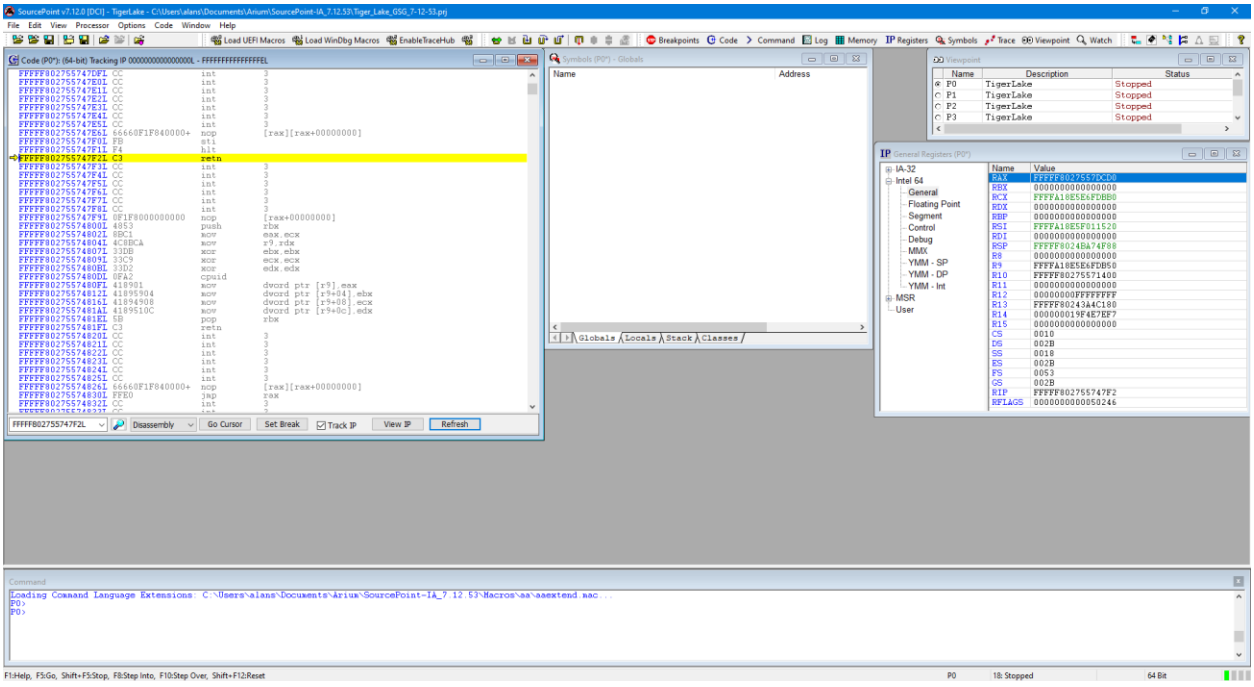
Boot the target to Windows. Log into the Windows desktop.

Follow the steps as described in the Getting Started with SourcePoint section above.

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

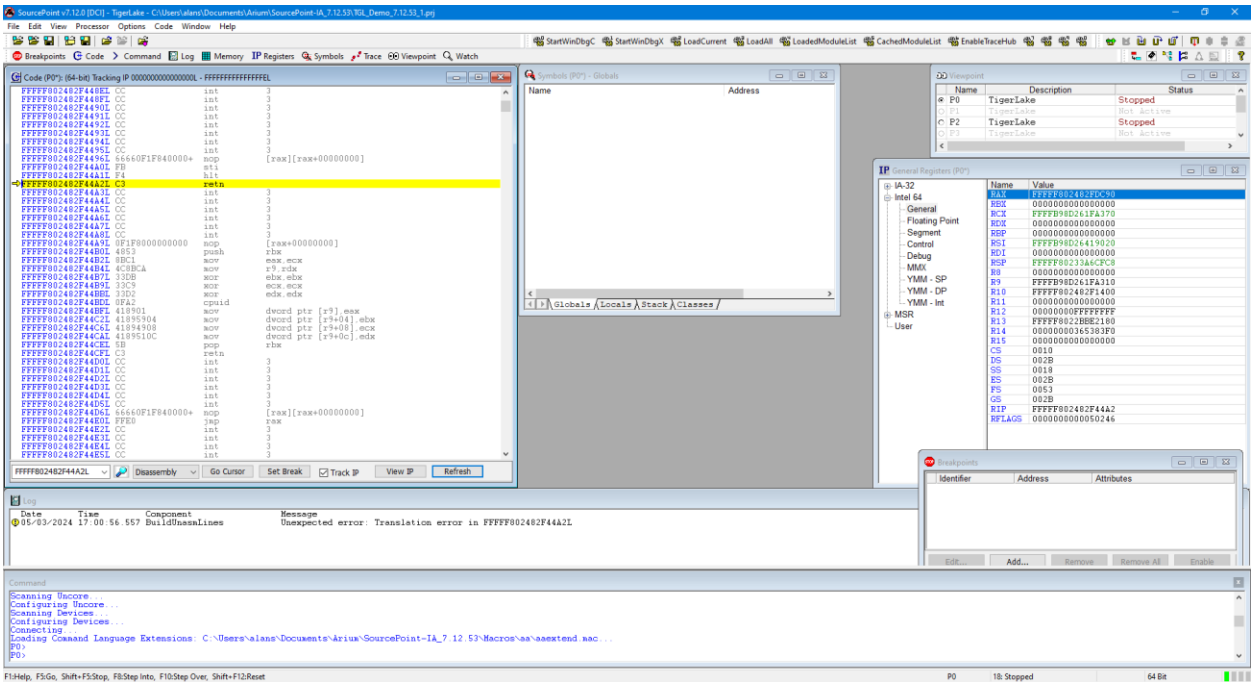
You will have to hit the Refresh button to see code displayed in the Code window. This transitions the Code view out of Safe Mode.

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:

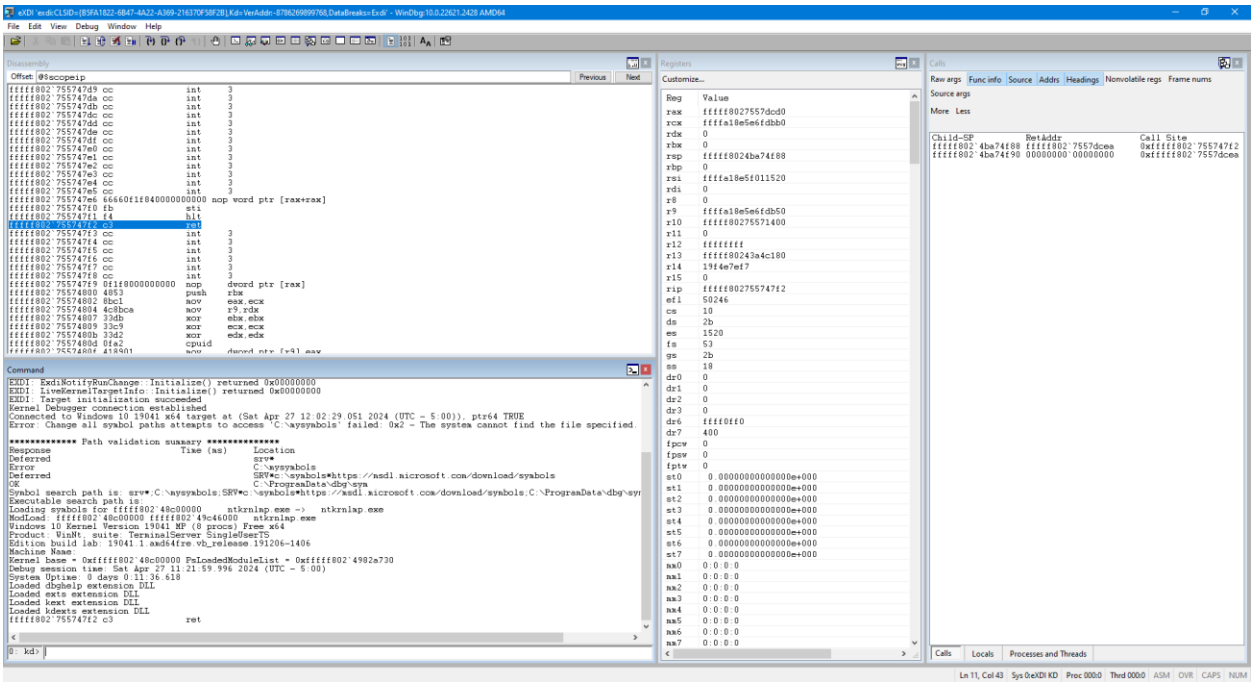


You now have extra buttons showing up, including:

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

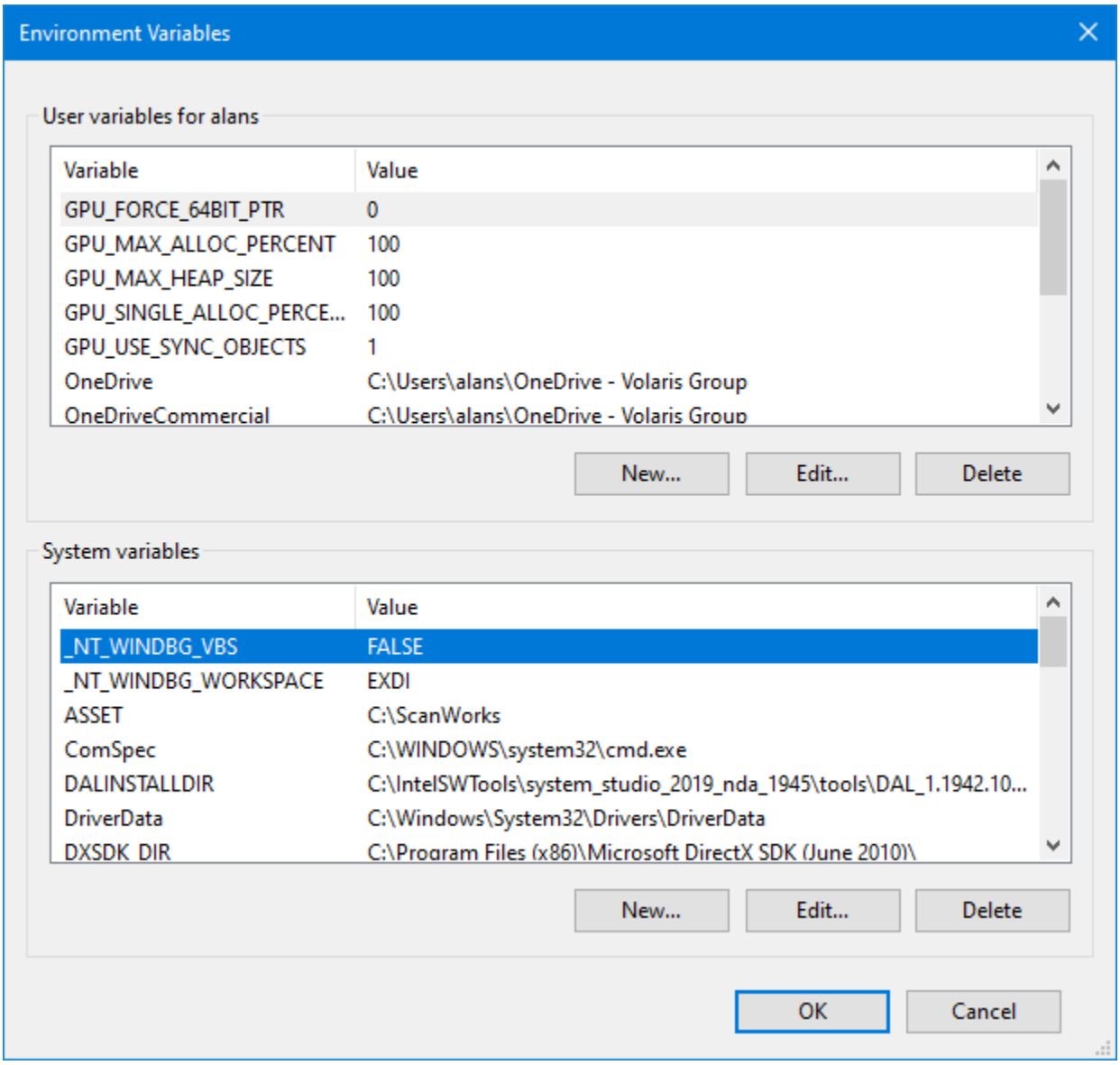
Next, it is time to run the SourcePoint macro that launches WinDbg and establishes the EXDI connection. For simplicity, click on either StartWinDbgC (Classic) or the StartWinDbgX macro button at the top of the screen. After about 30 seconds, WinDbg will open:



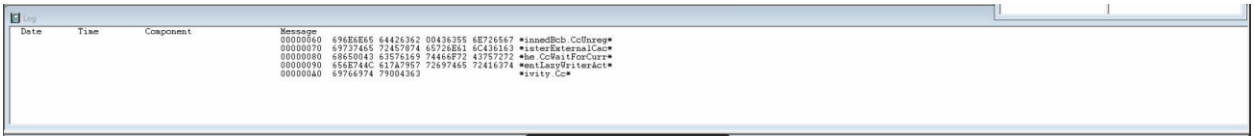
Power Tip: You have a choice of launching WinDbg Classic or WinDbgX via the macro buttons at the top. With this release of SourcePoint, ***WinDbg Classic is more stable and higher performance***. See our Troubleshooting section. Alternatively, at the SourcePoint Command line, you can type in StartWinDbg(true)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

Power Tip: Ensure an environment variable _NT_WINDBG_VBS has been created, and set it to FALSE.

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



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.

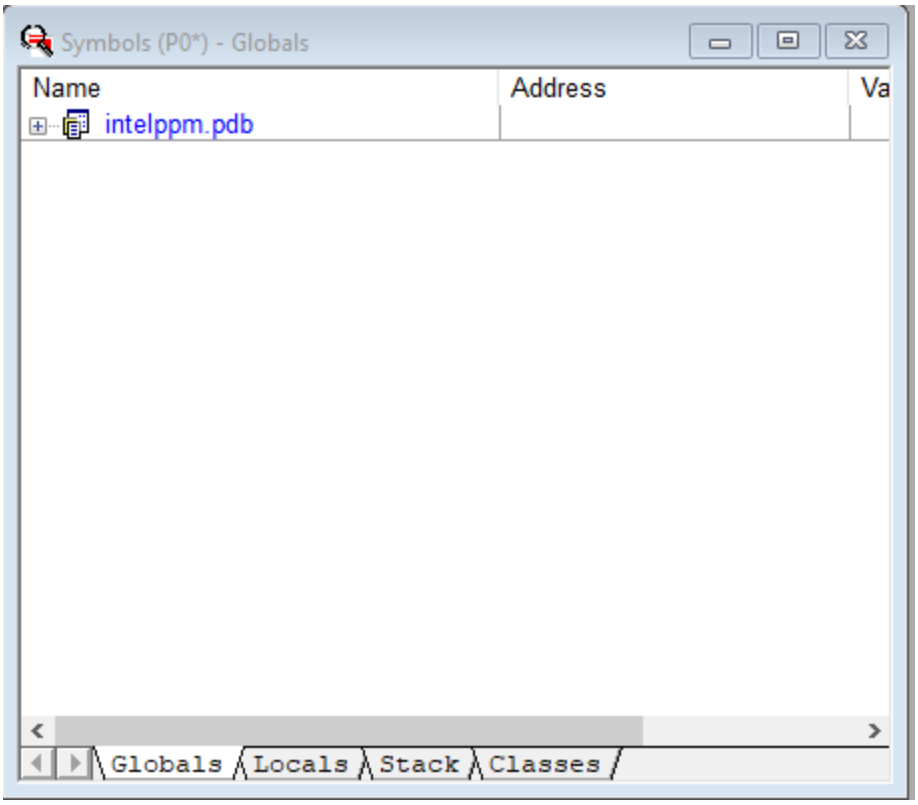
Step 3: Load symbols with the LoadCurrent macro

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:

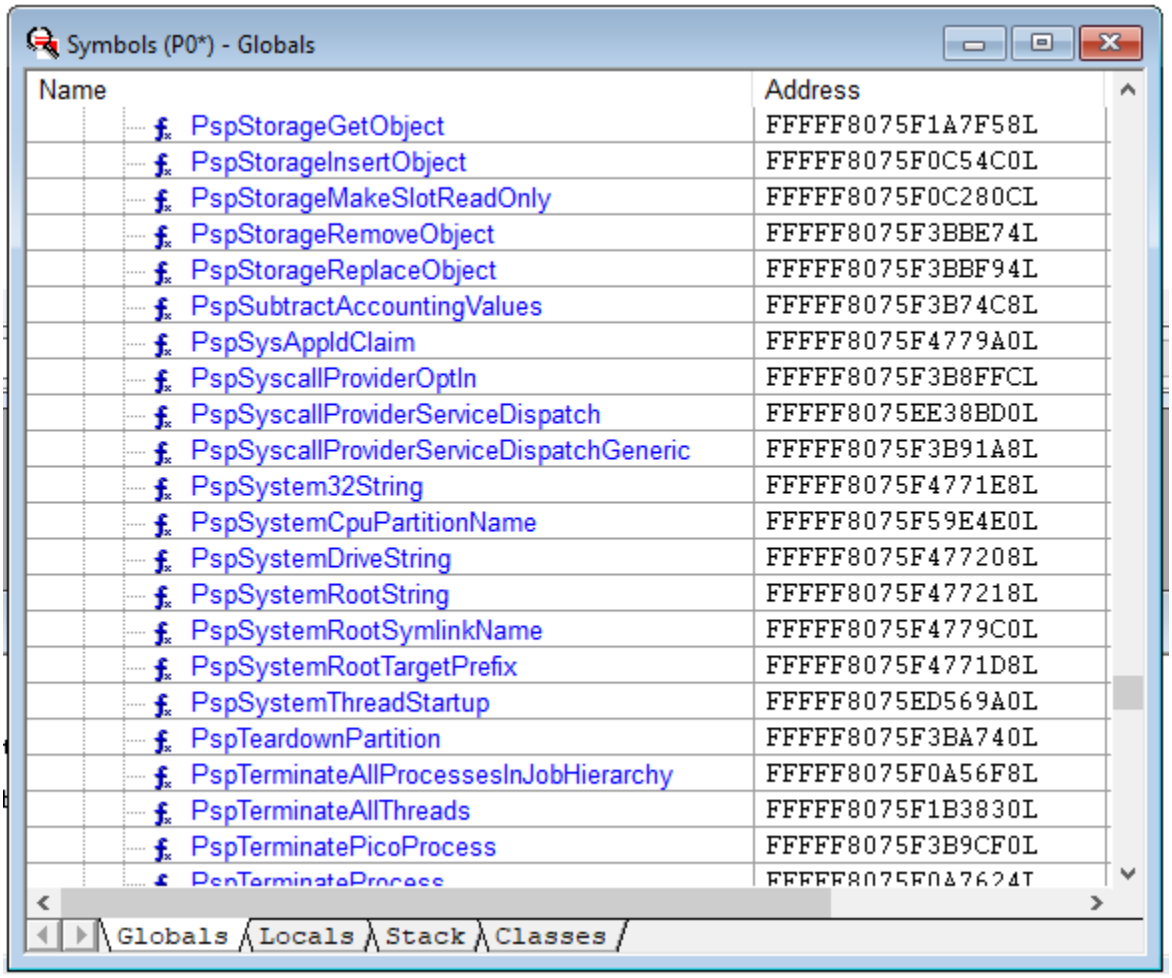
Ensure that the target is in a Stopped state.

Click on the `LoadCurrent` macro button in the SourcePoint Icon toolbar at the top. After about 20 seconds, the SourcePoint Symbols window will display the module that the current instruction is in (skip to the next section, What happens if the symbols don't show up? if it does not):



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:

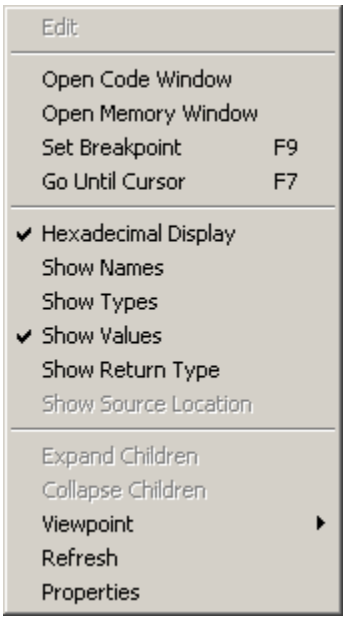


Power Tip: If WinDbg accesses symbols outside of intelppm.pdb (which it will during any typical debugging session), you'll need to run another `LoadCurrent` to additionally access these new symbols within SourcePoint.

Power Tip: If you want to load the symbols for an address outside of the current module or context, say at address 0xFFFFF80643C46000, in the SourcePoint Command window type in `LoadCurrentWinDbg(FFFFF80643C46000)`.

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:



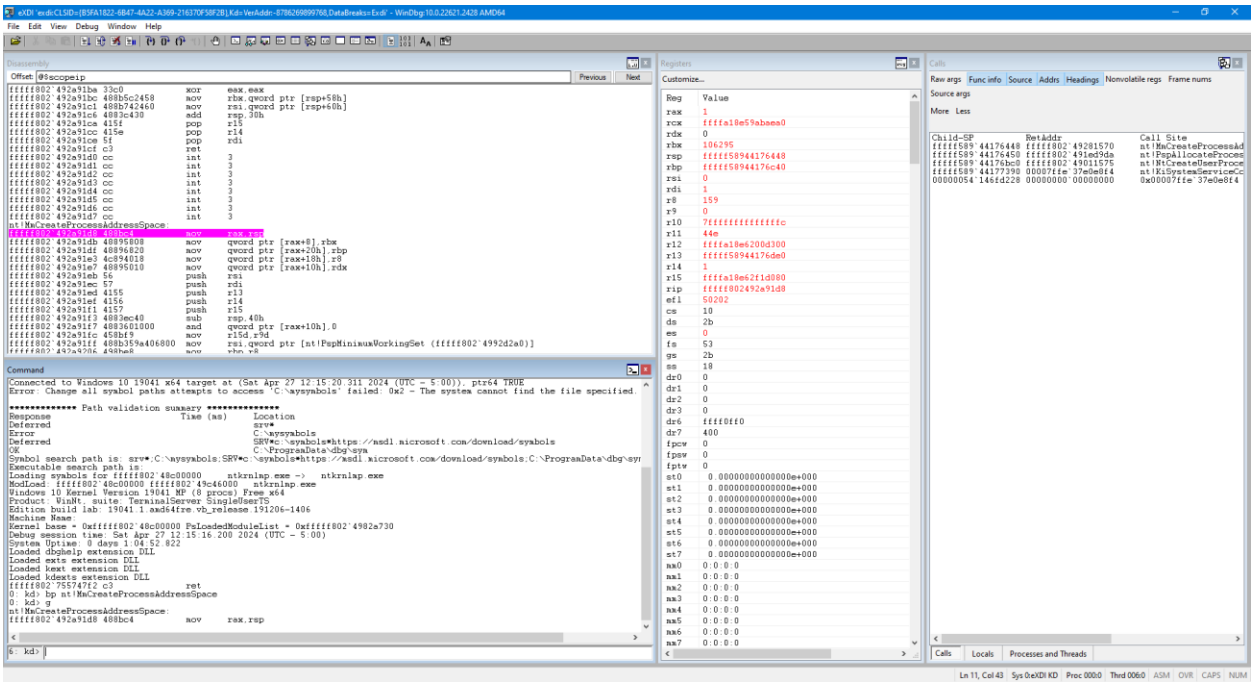
Now, it is possible to see the power of the two applications applied together. As an example, go into WinDbg and set a breakpoint on the entry point to the function `MmCreateProcessAddressSpace`:

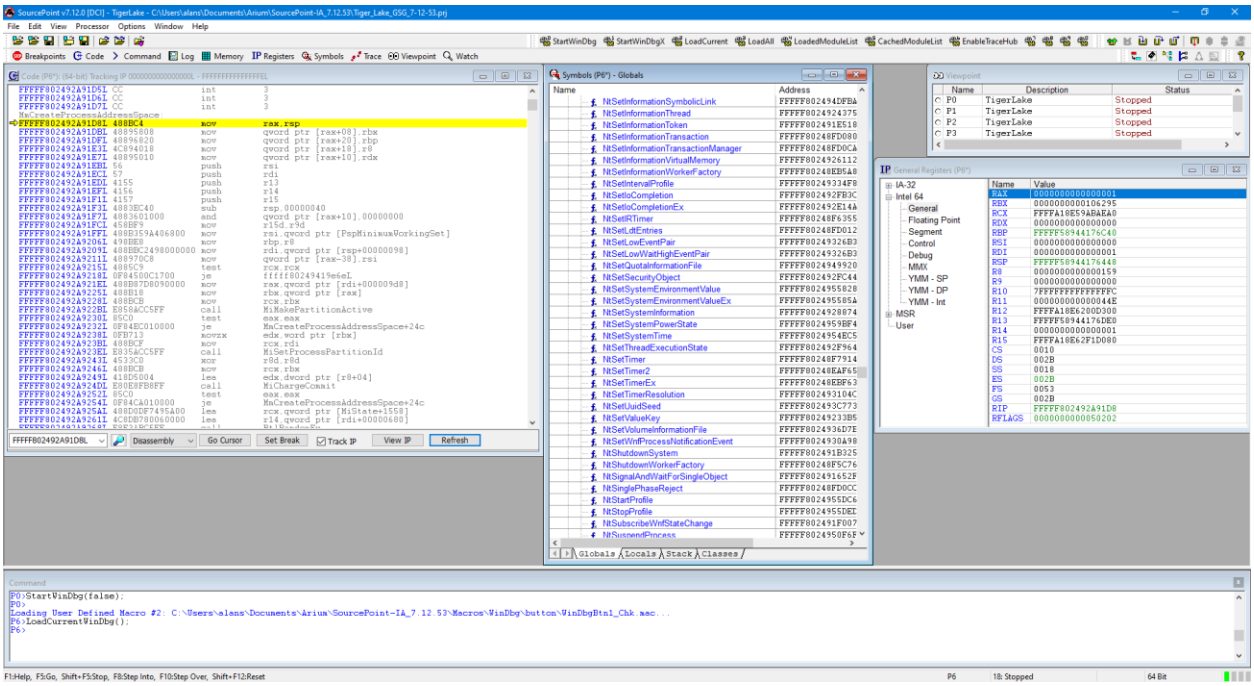
```
bp nt!MmCreateProcessAddressSpace
```

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.

Power Tip: 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.

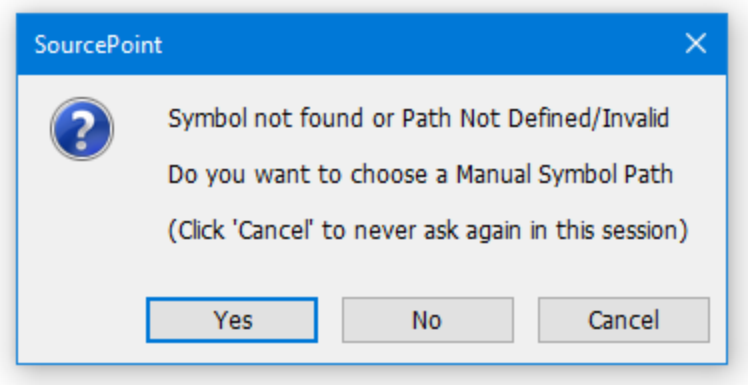
Power Tip: Once the PDB file is identified, SourcePoint will search for the symbol file in WinDbg's stored Symbol path, and then if not found, its Cache path. The symbol path in most WinDbg installations is something similar to:

`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.

What happens if the symbols don't show up?

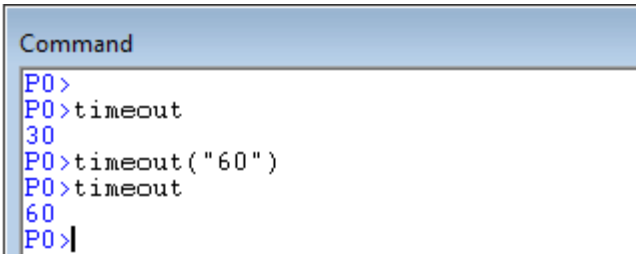
In some instances, there are no symbols available for the module that SourcePoint is halted in, or the specified path to the symbols is invalid. You will see the following:



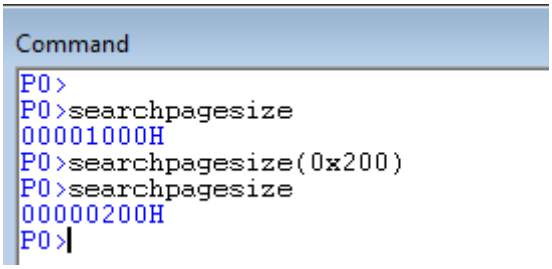
Check to ensure that your cached symbol path is correct by issuing the `sympath` command from the SourcePoint Command window. If it is correct, then the symbols are not local; fetch them from the Microsoft symbols server, presuming that they are available.

Another possibility is that SourcePoint's heuristic search for the PE32 header in system memory misses it, because the code is too small and it overshoots, or too large and exceeds the timeout period. There are two commands available from the SourcePoint Command line that can help:

timeout("nn") in seconds will change the timeout from default 30 seconds to 'nn' seconds



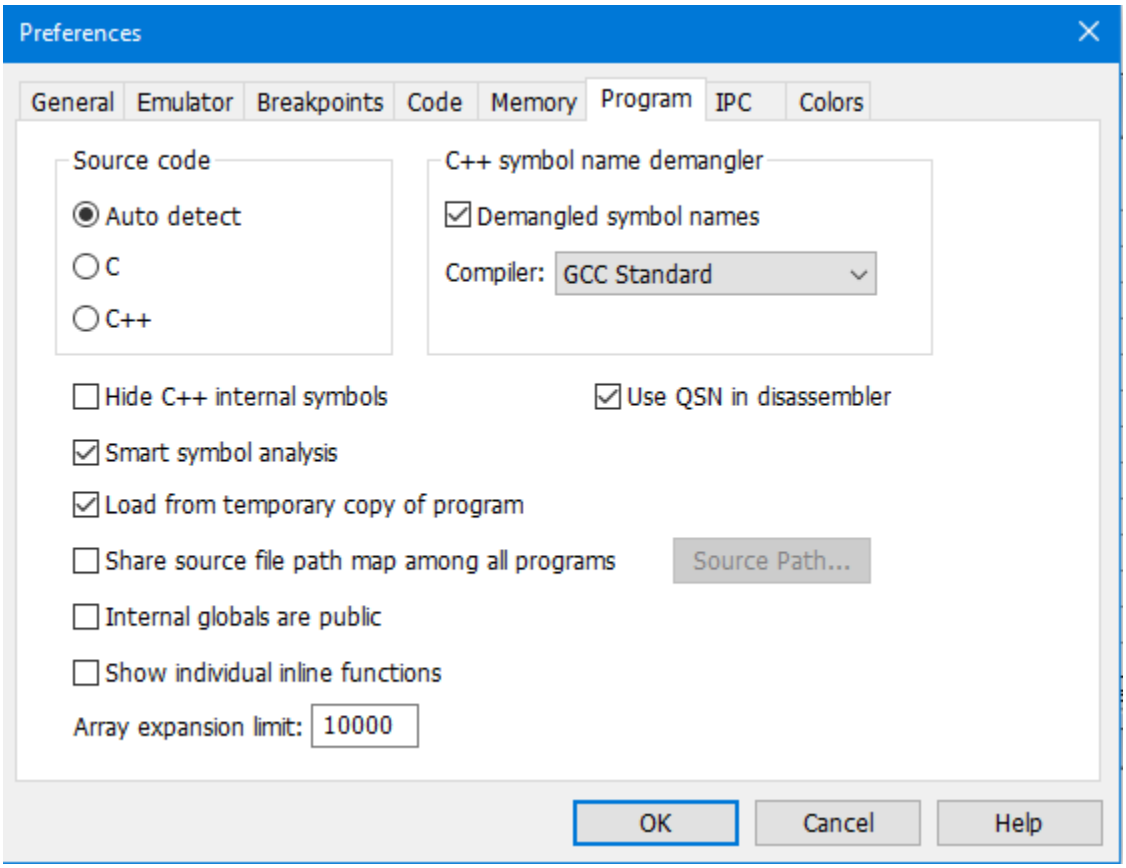
SearchPageSize(nnnnn) in hex will change the search decrement for detecting modules (base).



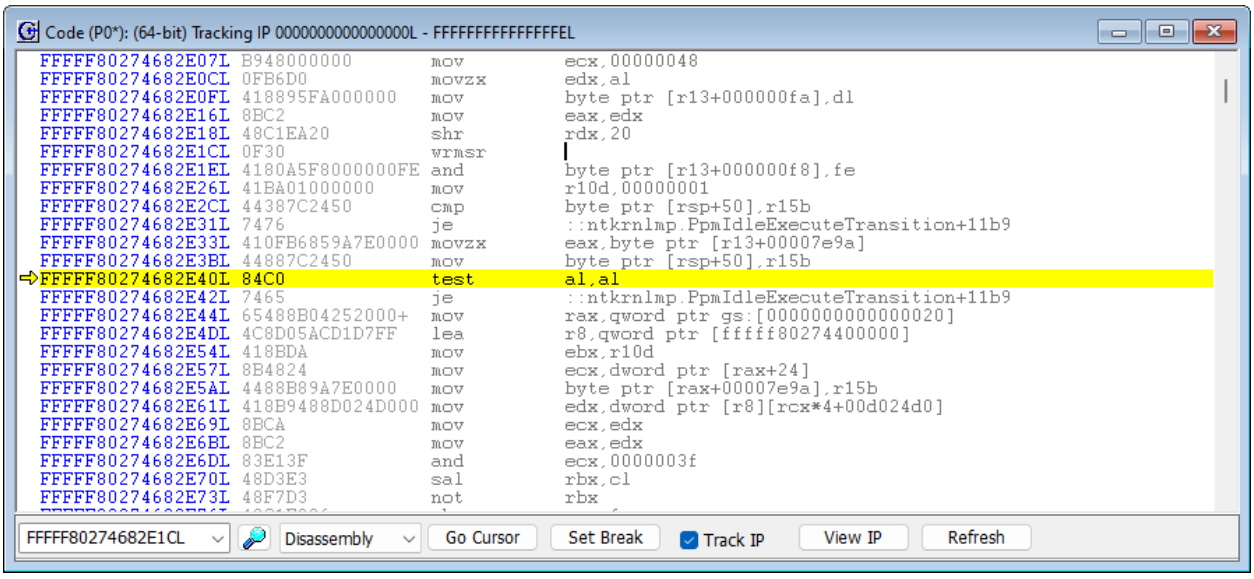
Getting SourcePoint to display module names as well as function names

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.

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

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. The vast majority of time, quitting out of SourcePoint and power cycling the target restores things to normal. But, in all of our testing, we've each had to reinstall Windows at least once. Really, it's no different from reinstalling Windows in a VM, only more onerous. But we've not ever been able to reproduce this type of failure.

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. In general, they can be ignored.

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, so we recommend its use, instead of WinDbgX.**

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.

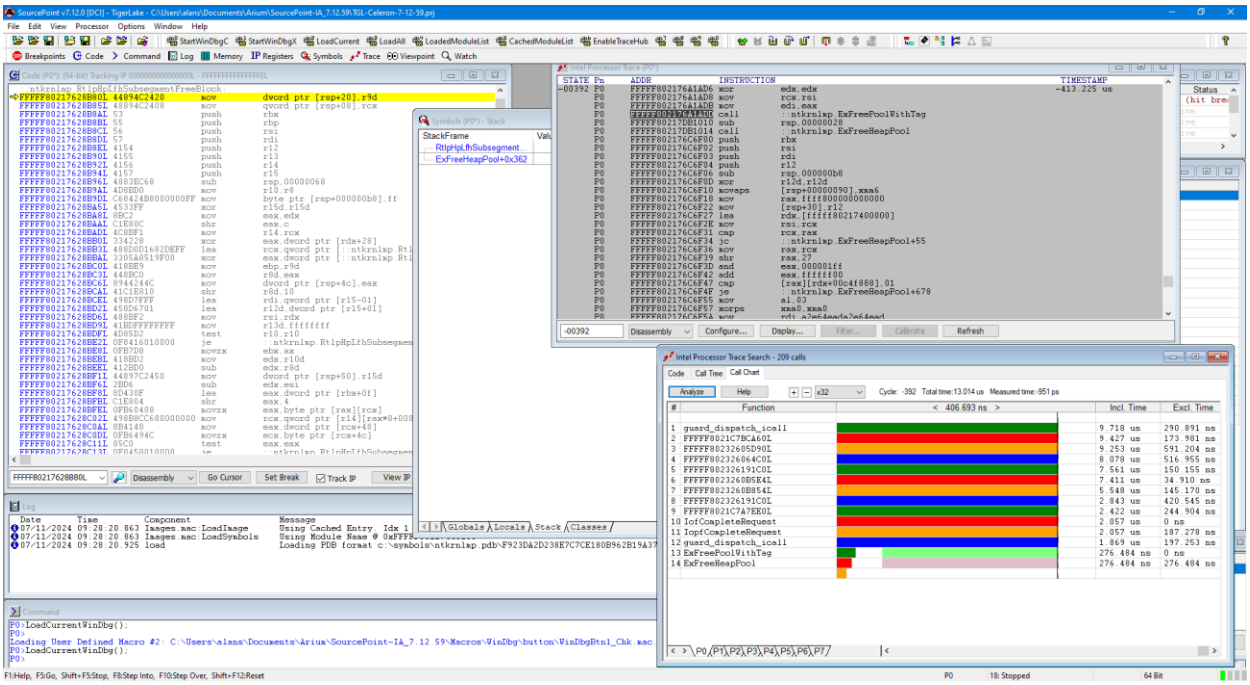
COM(32) Surrogate

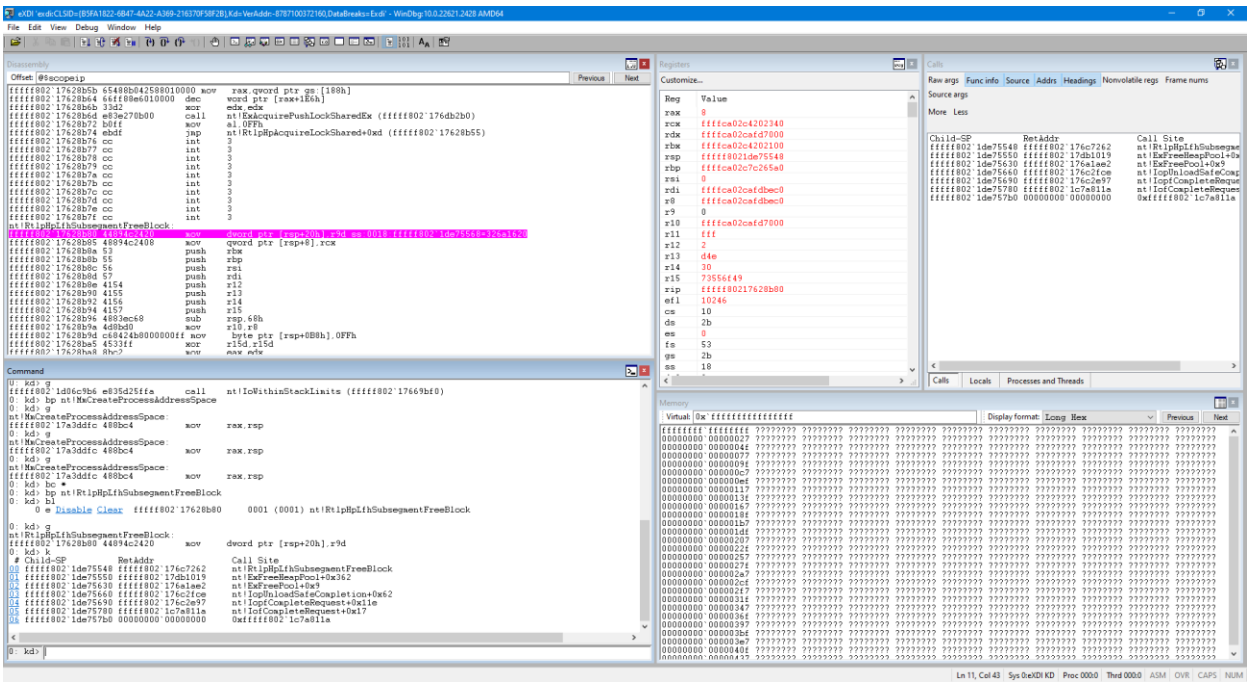
After a crash, when you restart SourcePoint, once in a blue moon it will misbehave. Run-control will not work properly.

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

Viewing the Stack

SourcePoint's Stack display in its Symbols window has been improved in the most recent release, but needs further enhancement. You will find that it gives different results from what will be seen in the WinDbg "k" command, and which may further differ from what you may see in an Intel Processor Trace traceback. Here is an example:





If you look carefully, you'll see that "guard_dispatch_icall" does not display in the Stack tab of the SourcePoint Symbols window. It has a special call frame/sequence that our Stack traceback does not display. But it does show up in Intel Processor Trace.

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 -> \0000000000000000000000000000000\
```

We are working on this.

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
```

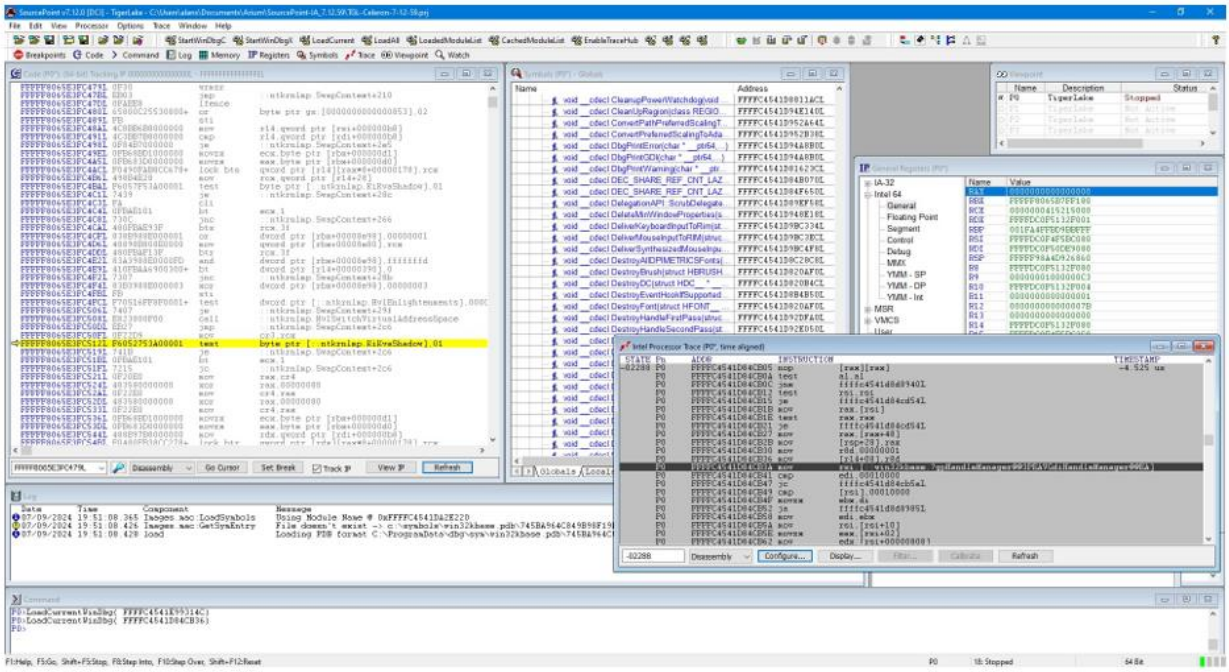
does not work properly. Instead of halting in the instance of nt!NtReadFile associated with explorer.exe, it will halt at the first instance of the shared code, likely in a different process. This is because EXDI does not provide process/thread information down to SourcePoint, unlike the standard WinDbg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

Mangled function names

SourcePoint has a built-in name demangler.

Some of the Windows kernel is written in C++ code, such as in win32kbase.sys, and the demangler comes in handy for this.

You may see this in action for example by looking at some of the win32kbase code here:



Note that to see the demangled name in the SourcePoint Code window, you need to turn “Use QSN in disassembler” off under Disassembly in menu Options > Preferences > Program.

WinDbg FP register display is not working

WinDbg does not display the floating point registers contents when using EXDI. SourcePoint displays the register values correctly.

Problems with symbol loading from local cache

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:

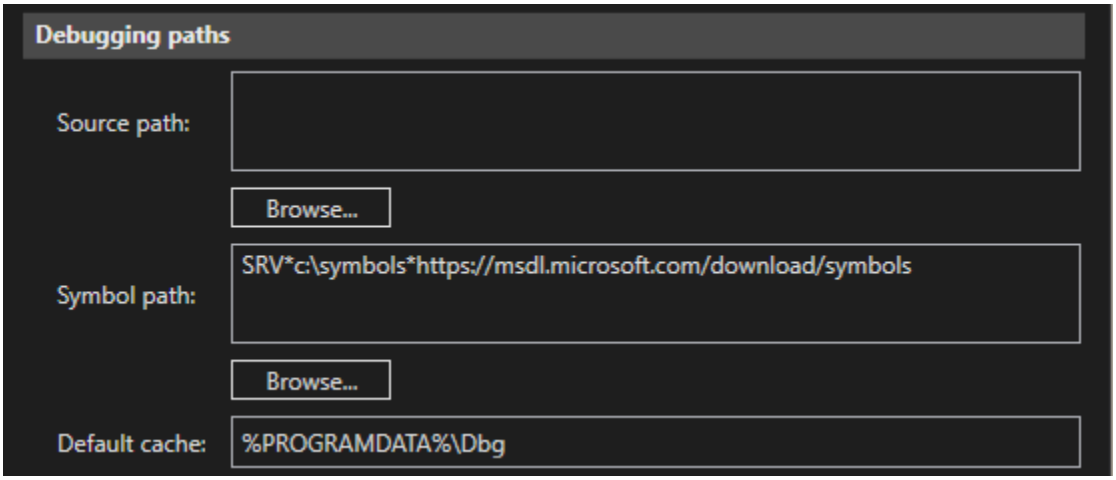
```
windbgc
```

Alternatively, typing:

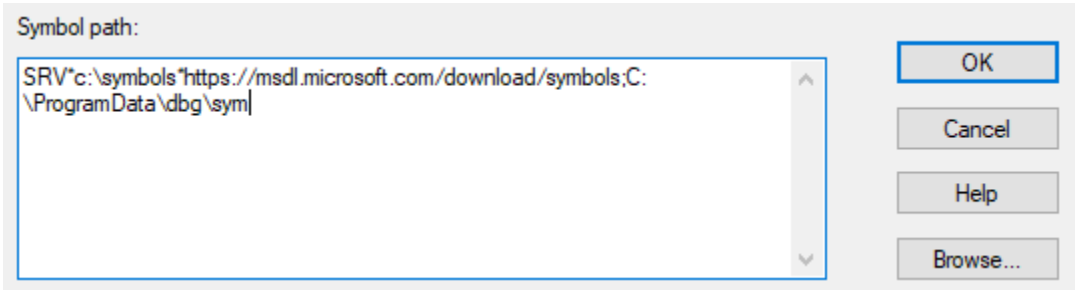
```
windbgx
```

sets up the symbol path as defined within WinDbgX. Note that with WinDbgX, SourcePoint will automatically set its own sympath.

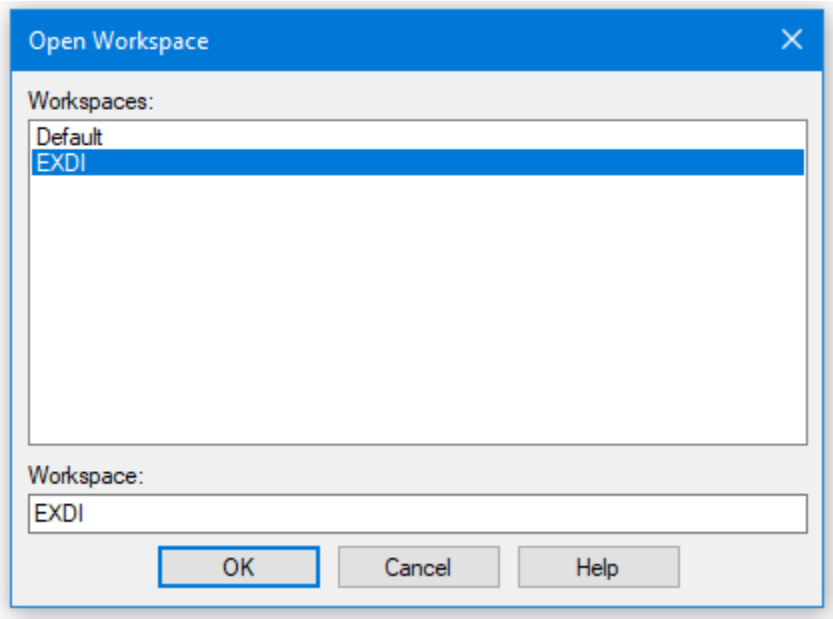
But, symbol and cache paths are a little trickier with WinDbg Classic. WinDbgX allows for an explicit description of the cache path:



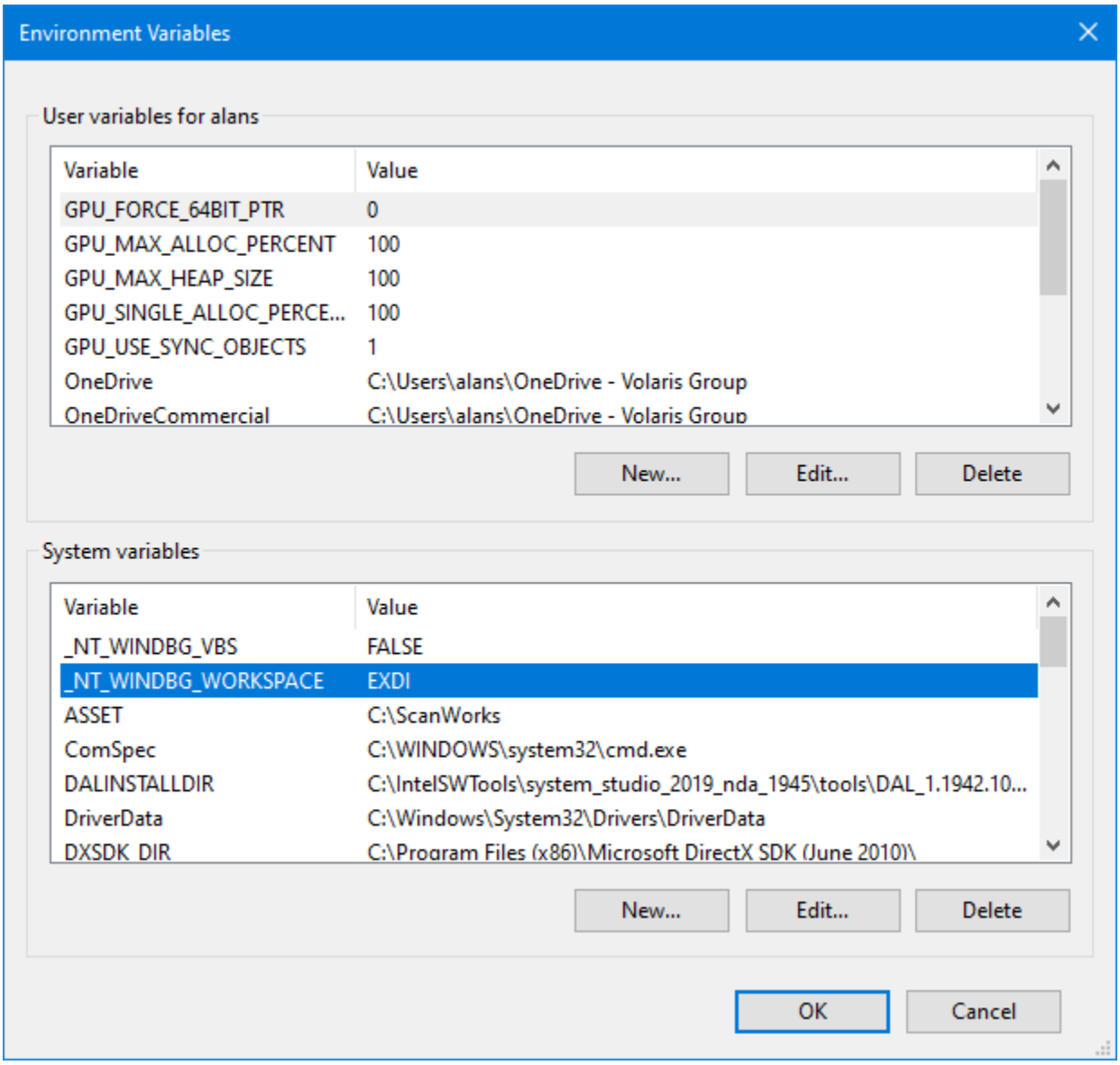
Whereas WinDbg Classic does not; you have to embed it in the Symbol path, and that is squirrelled away in the registry:



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):



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
```

Power Tip: 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.

SOURCEPOINT DEBUGGING FOR HYPER-V

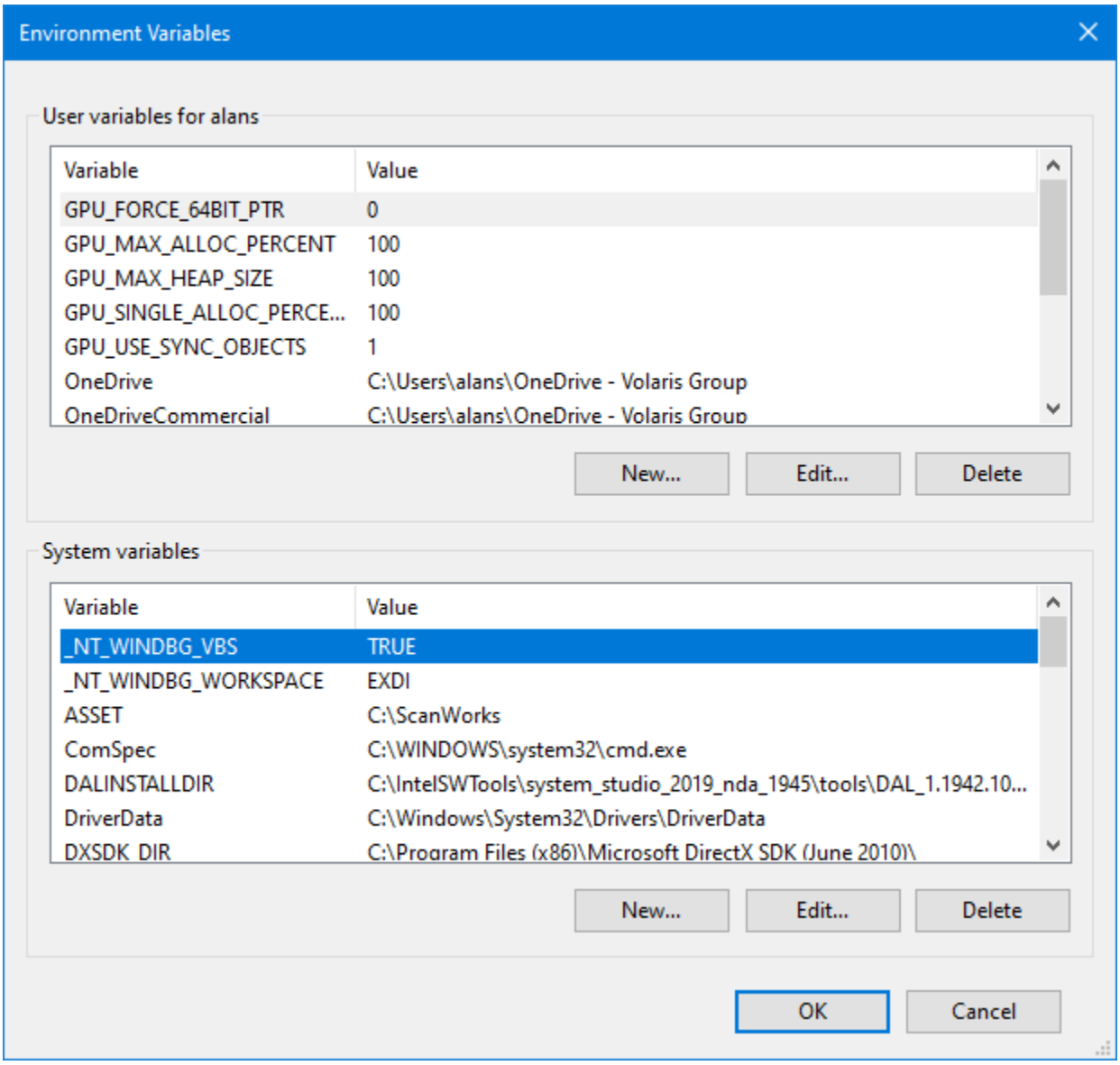
Debugging with Hyper-V and Virtualization-Based Security (VBS) is supported in the SourcePoint 7.12.59 and later releases. Visualization of the virtualization environment, and probe mode access to it, provide an unprecedented level of capabilities.

Getting Started

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

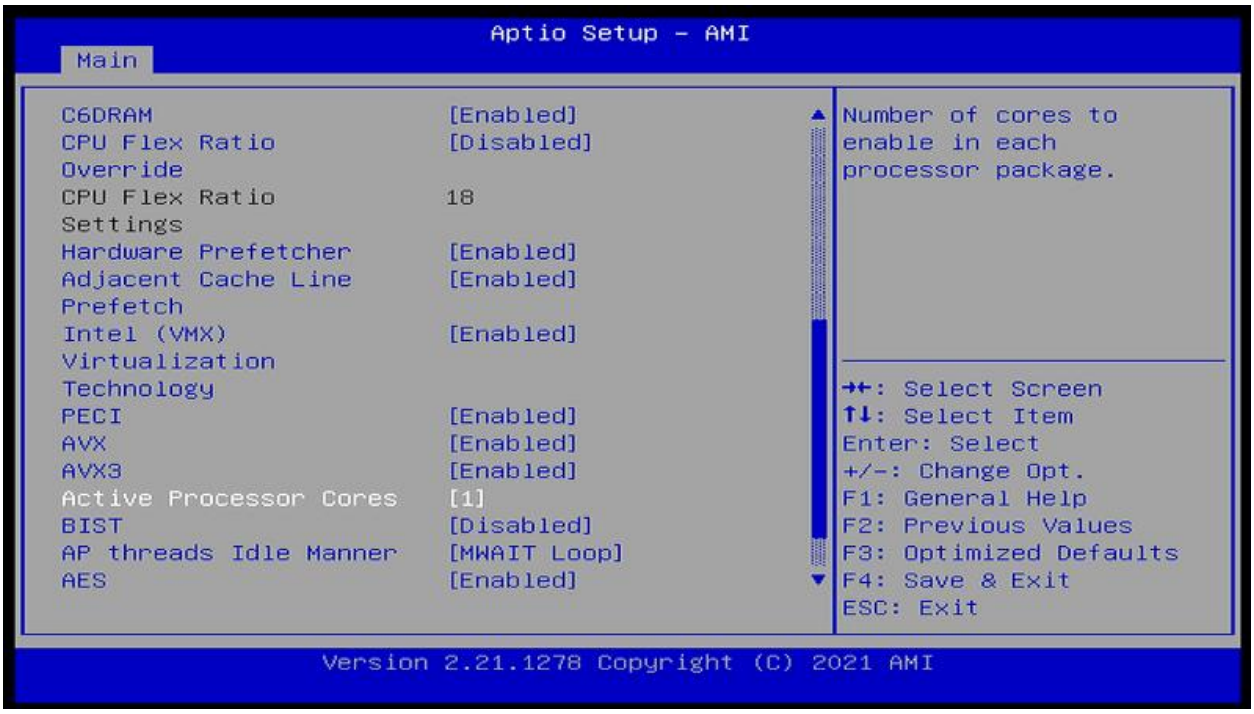
Ensure that the target has Hyper-V and VBS enabled.

Then, 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:

Via the advanced BIOS settings (with the upassw0rd password – see the SourcePoint UEFI Getting Started Guide), 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:



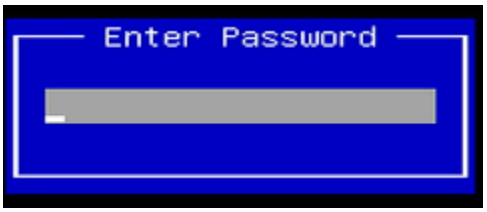
Disable the synthetic watchdog on the target to ensure that the target does not autonomously reset itself in probe mode:

```
>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. This is not really necessary, and is inconvenient, so to save time, you can disable recovery boot from an Administrator CMD window using the two bcdedit commands:

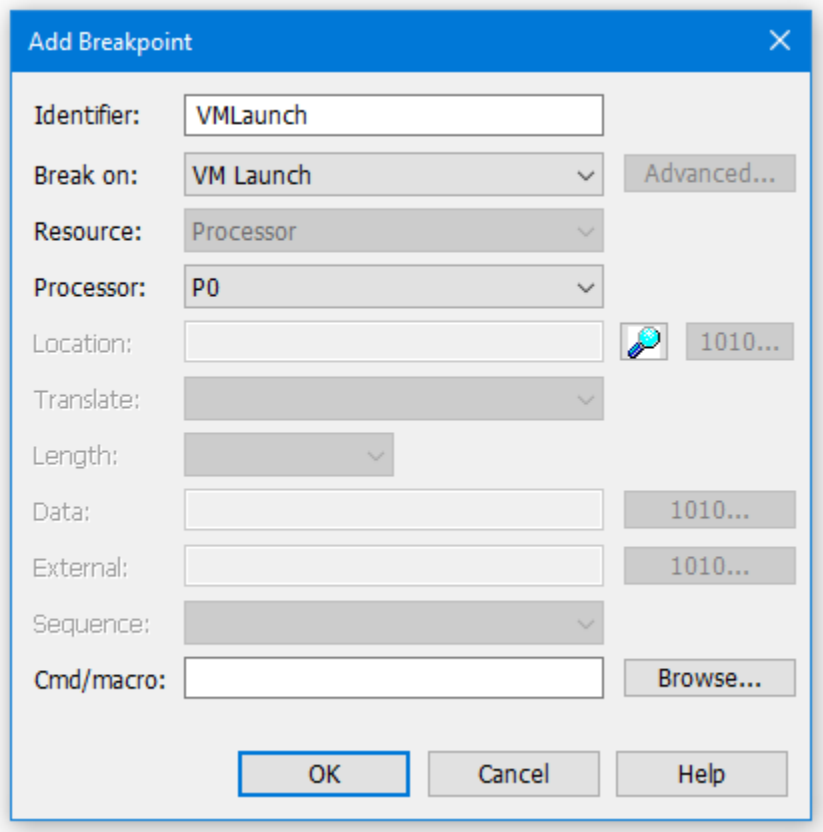
```
>bcdedit /set recoveryenabled No >NUL
>bcdedit /set bootstatuspolicy ignoreallfailures >NUL
```

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:

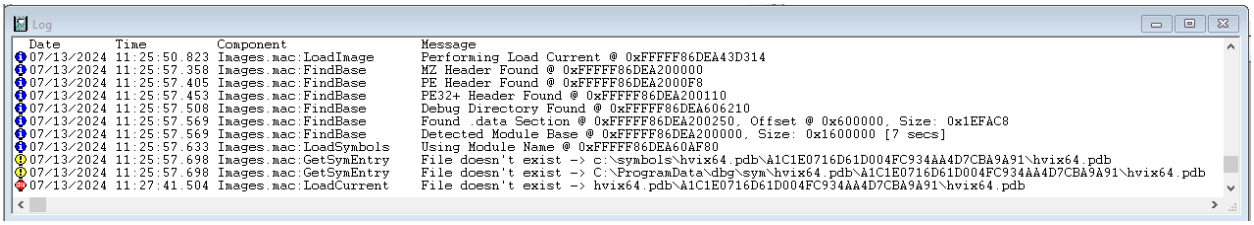


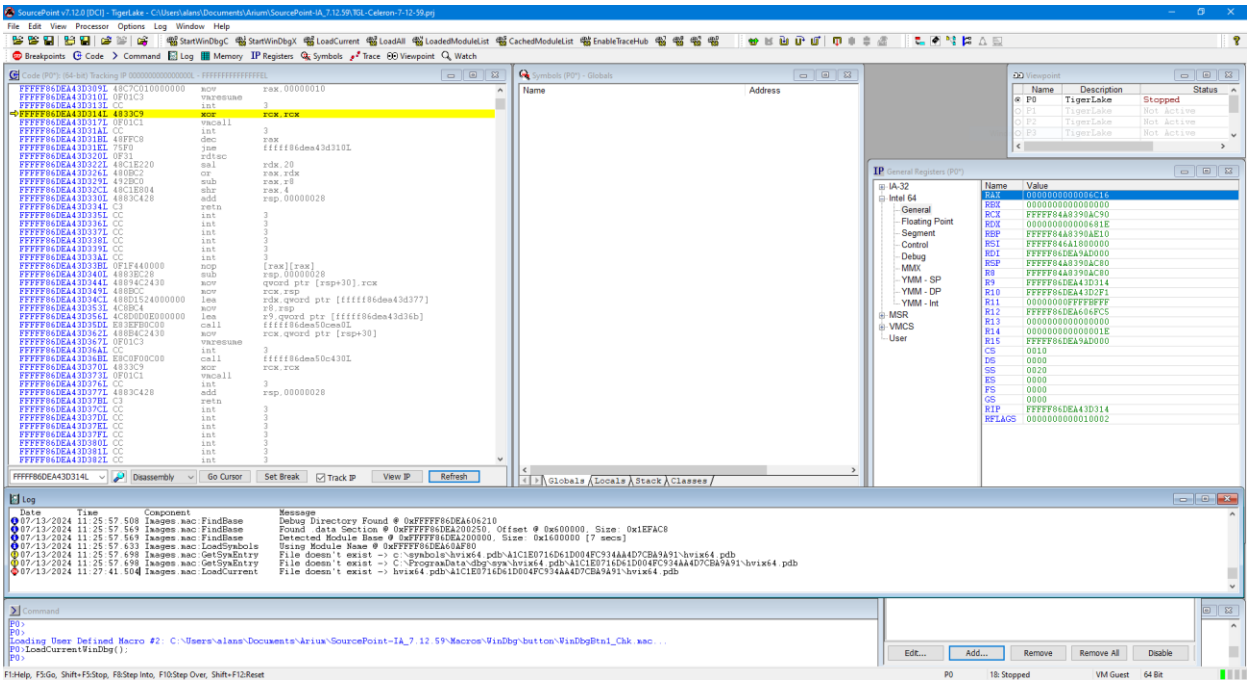
VMM breakpoints, and debugging the Secure Kernel

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



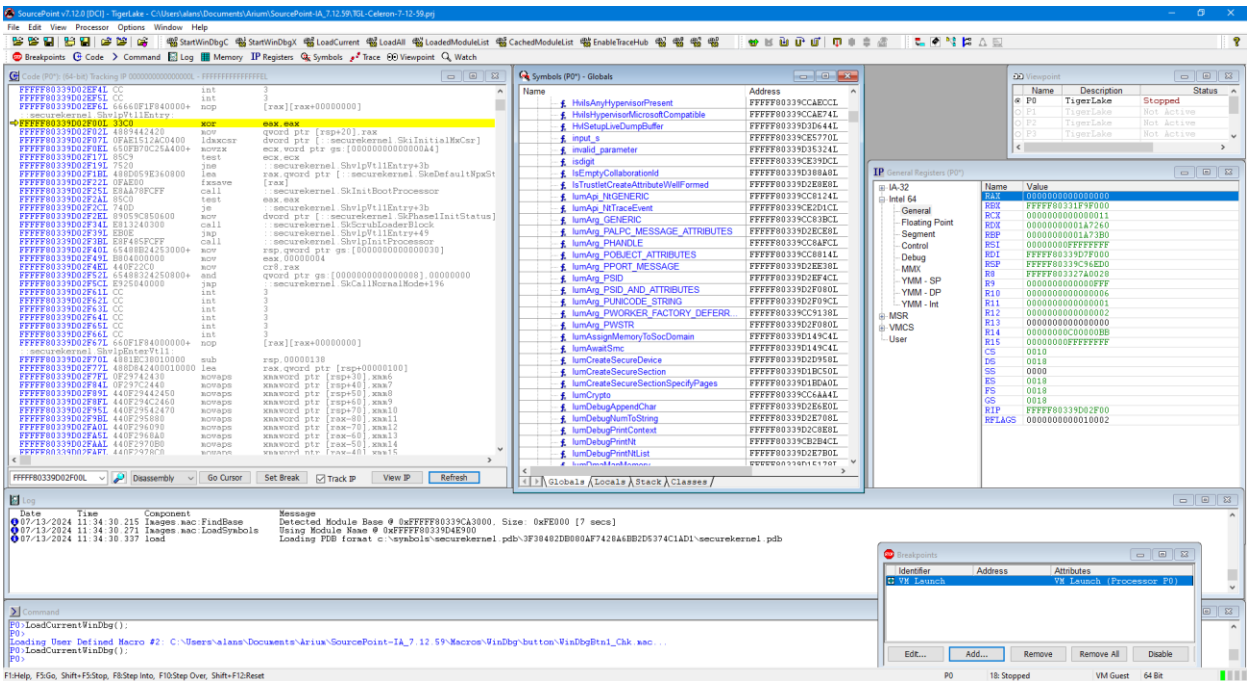
Hit Go in SourcePoint. Be sure to quit out of the target's UEFI shell password prompt (hit Esc and then accept "Yes") to start the Windows boot process. You will break at the first VM launch, and land in the hypervisor, in VM Guest mode. Click on the LoadCurrent macro button, and say "No" to the "Symbol not found" prompt to see we're in hvix64, for which there are no symbols:





Hit Go, and then LoadCurrent a second time. Do a LoadCurrent again. You're in hvloader (still no symbols), again in VM Guest mode.

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!

Power Tip: The above strategy works for having symbols display within SourcePoint while debugging in a Hyper-V environment. But what if we want the symbols to show up in WinDbg as well, while debugging via EXDI? There are several ways of doing this, but the easiest approach is to get the module base and size from within SourcePoint, and then use the `.reload` command within WinDbg to load the symbols there.

Example: while in the secure kernel with SourcePoint, click on LoadCurrent, and you'll see something like:

```
07/13/2024 13:57:30.211 Images.mac:IsKPCR          Invalid KPCR Pointer. Check MSR values.
07/13/2024 13:57:30.211 Images.mac:LoadImage       Performing Load Current @ 0xFFFFF8021FD48F00
07/13/2024 13:57:36.020 Images.mac:FindBase        MZ Header Found @ 0xFFFFF8021FCE9000
07/13/2024 13:57:36.066 Images.mac:FindBase        PE Header Found @ 0xFFFFF8021FCE9108
07/13/2024 13:57:36.066 Images.mac:FindBase        PE32+ Header Found @ 0xFFFFF8021FCE9120
07/13/2024 13:57:36.169 Images.mac:FindBase        Debug Directory Found @ 0xFFFFF8021FD8FED0
07/13/2024 13:57:36.272 Images.mac:FindBase        Found .data Section @ 0xFFFFF8021FCE9300,
Offset @ 0xB9000, Size: 0x1A528
07/13/2024 13:57:36.272 Images.mac:FindBase        Detected Module Base @ 0xFFFFF8021FCE9000,
Size: 0xFE000 [6 secs]
07/13/2024 13:57:36.336 Images.mac:LoadSymbols     Using Module Name @ 0xFFFFF8021FD94900
07/13/2024 13:57:36.402 load                       Loading PDB format
c:\symbols\securekernel.pdb\3F38482DB080AF7428A6BB2D5374C1AD1\securekernel.pdb
```

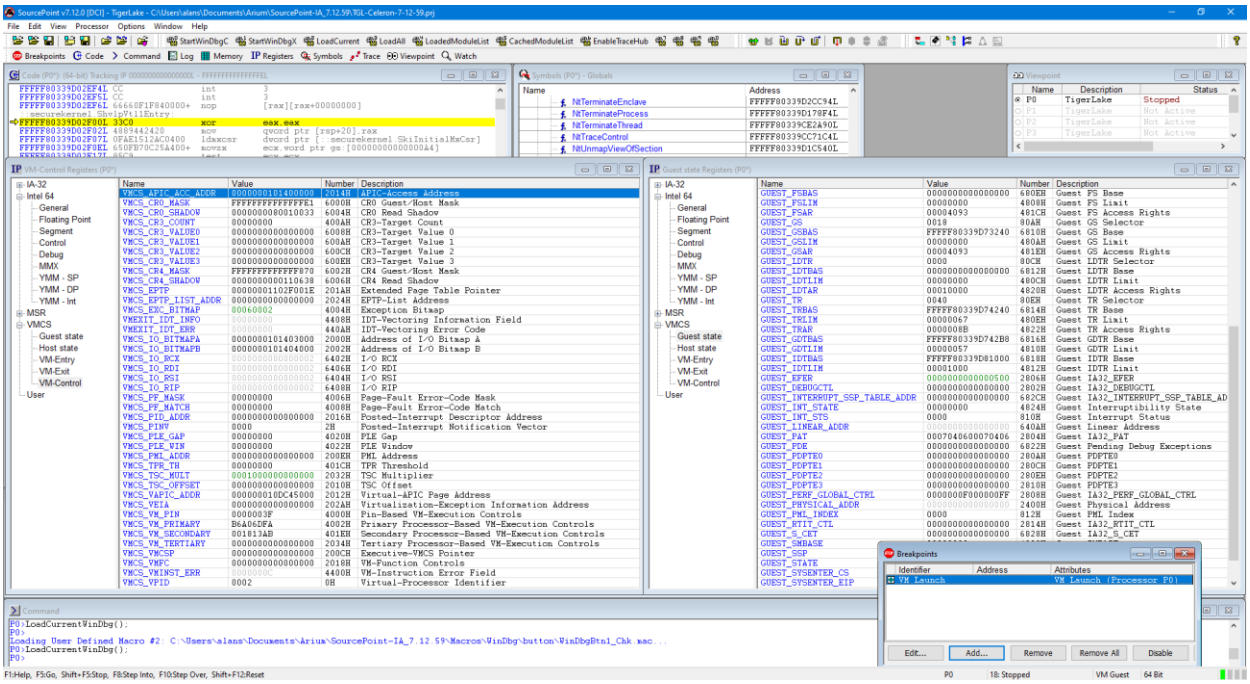
Note the highlighted sections above. Use these to load the symbols within WinDbg via .reload /f /i module=<address>,<size>:

```
kd>.reload /f /i securekernel.exe=0xFFFFF8021FCE9000,0xFE000
```

VMCS Viewer/Editor

But it gets more interesting. We are in the VTL 1 Secure Kernel, in Guest mode of course. Let's inspect both the Guest mode and Host mode VMCS (yes, it is possible to look at the state of the Host while you are in Guest mode – you can only do that with JTAG!).

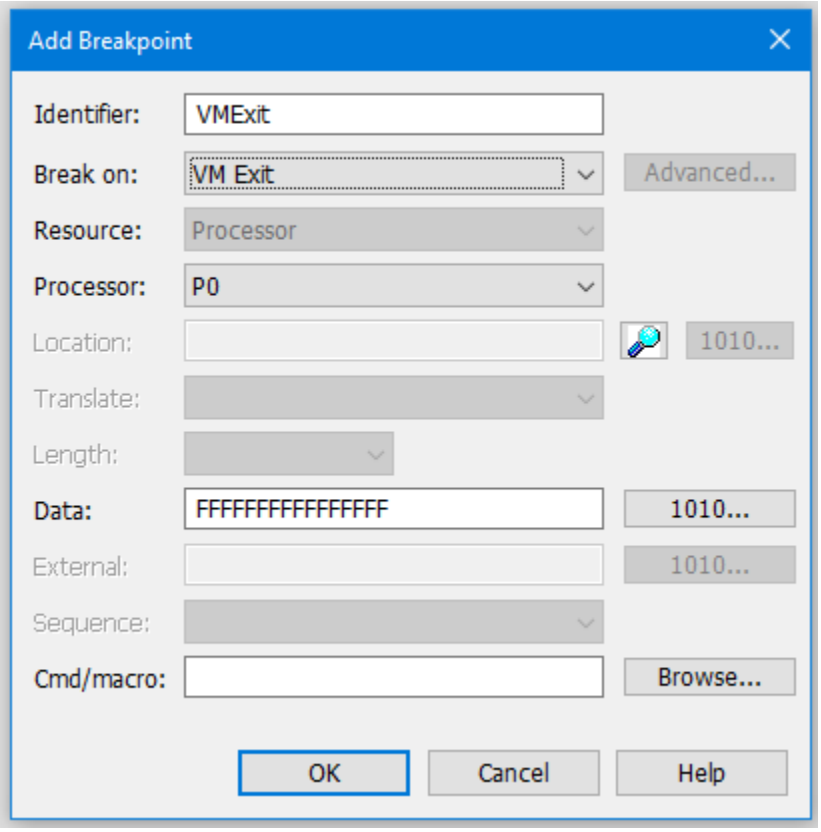
In the General Registers window, click on VMCS and select Guest state. Open up a new Registers window, and select Host state, or any of VM-Entry, VM-Exit and VM-Control registers. Have fun.



VM Exit breakpoints and Basic Exit Reasons

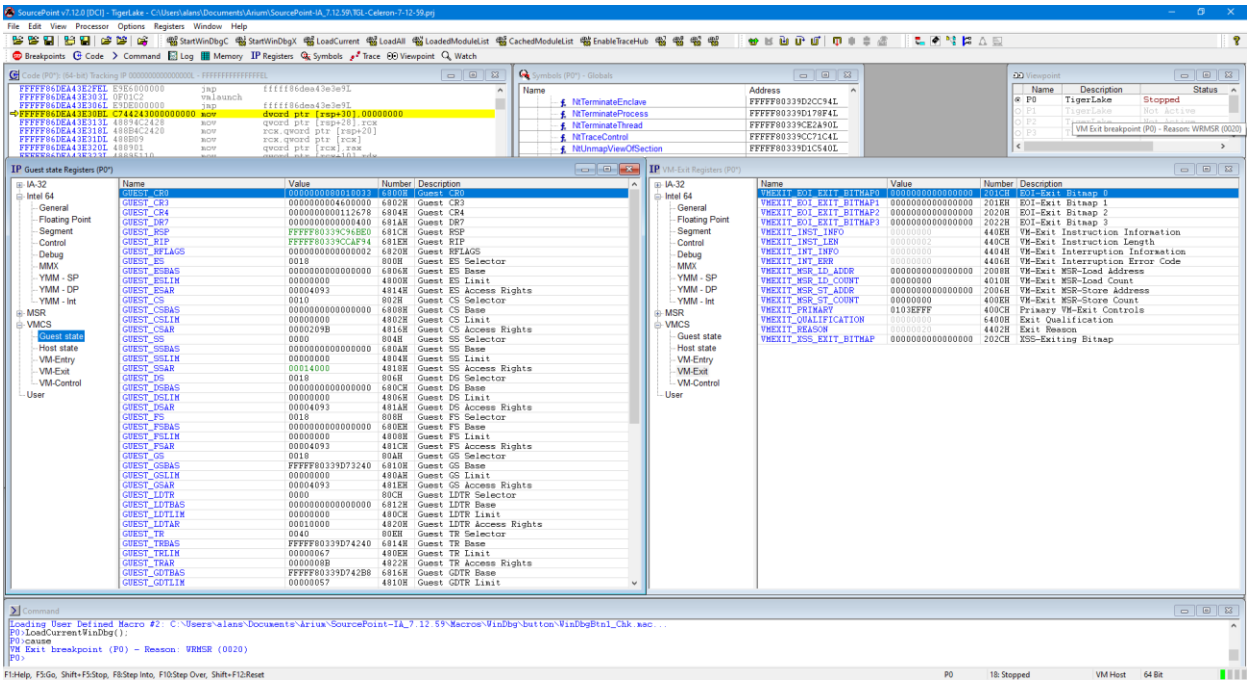
For a more advanced topic, let's look at using VM Exit breakpoints to capture Guest to Host transitions.

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.

Then hit Go. The VM Exit breakpoint will be hit, and this puts us back into hvix64, and this time we're in VM Host mode:



Note that you can see above that the VM Exit reason is x'20 that is WRMSR, by typing in the cause command in the SourcePoint Command window, opening up the VM-Exit register in the Registers window and looking at VMEXIT_REASON, or hovering over the processor in the Viewpoint window and looking at the tooltip.

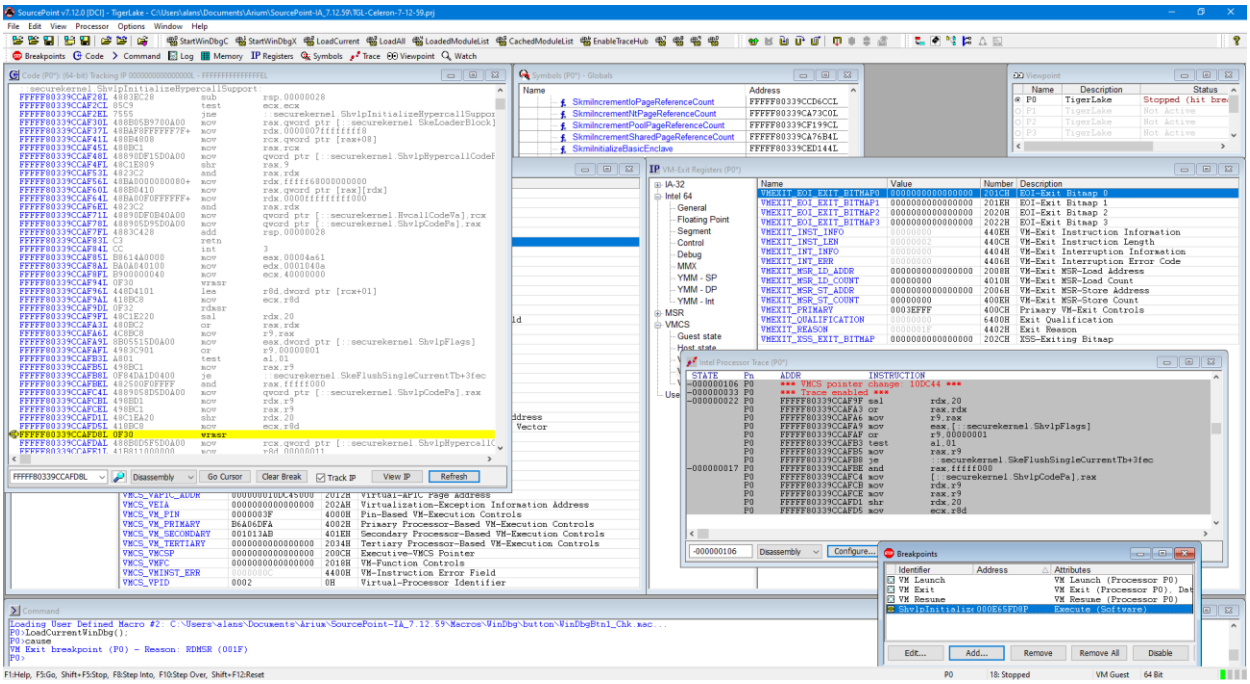
Using Intel PT with Hyper-V

Uncheck the VM Exit breakpoint, and use a VM Resume breakpoint to put the target back into the Secure Kernel.

Then open up a Trace window and activate Intel Processor Trace.

Disable the VM Resume breakpoint, and enable a breakpoint some point down into the Secure Kernel code.

Hit Go. You will collect Intel PT within the Secure Kernel! Here's an example:



Power Tip: Collecting Intel PT data from the Host while the target is in Guest mode will cause the target to crash. As one example, if you're in Host mode, set a VM Resume breakpoint, and hit Go while Intel PT is enabled, this will hose the target. As another example, while in Guest mode, if an interrupt comes along and takes the target from Guest to Host to Guest again, Intel PT will attempt to capture the Host mode instructions, and that will crash the target.

This will be addressed in a future SourcePoint release. Again, this capability will only be available via JTAG.

Suggested Hyper-V Reading

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. **Note: the vmcs macro has been deprecated and replaced by the VMCS Viewer/Editor; but there’s some interesting tidbits in this article regardless.**

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.

Part 6: JTAG debug of Windows Hyper-V / Secure Kernel with WinDbg and EXDI
An article about about Windows Secure Image Objects, an important structure used in sharing information between VTL 0 and VTL 1, the normal kernel and secure kernel.

Troubleshooting Tips on Hyper-V/ VBS Enabled Targets

VM Resume breakpoint with Intel PT crashes the target

When transitioning from Host to Guest mode, and halting in 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. It is recommended to disable Automatic Repair with:

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

Also, don't forget to turn off the synthetic watchdog:

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

Hardware breakpoints don't work well in the Secure Kernel

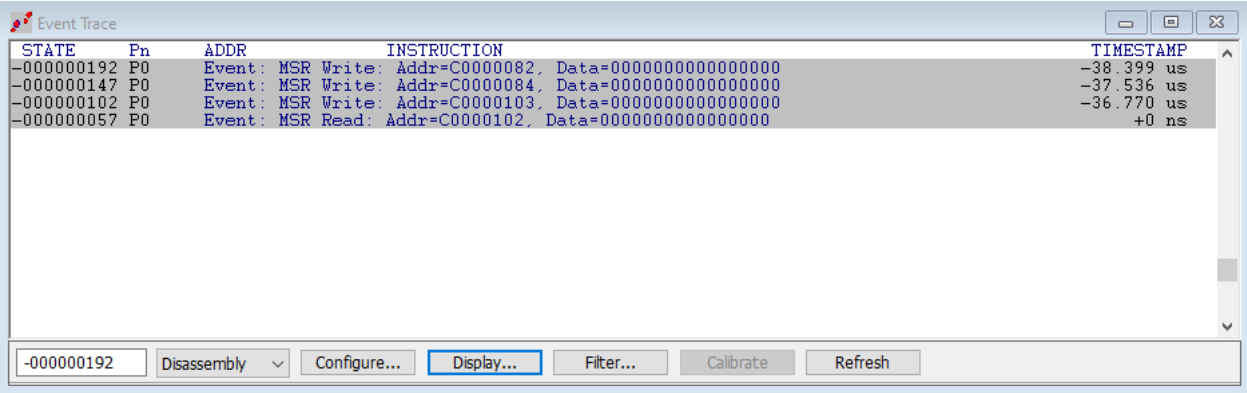
There are a few issues here, including:

(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.

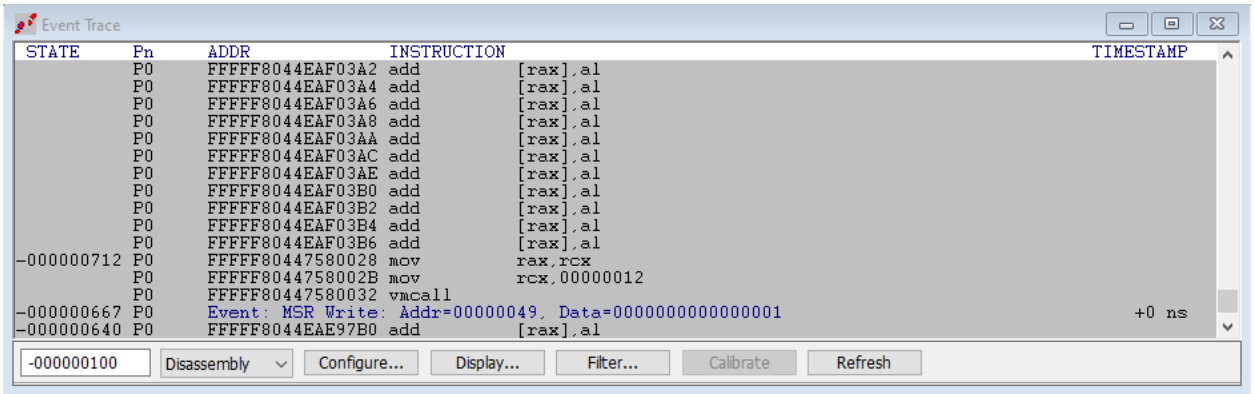
These are all to be fixed in the upcoming release.

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, when transitioning from Host mode to Guest mode, and where the events occur in Host mode, you don't see the actual disassembly in the Event trace window (remember, for now you can't read Host mode memory from Guest mode), 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):



This behavior of AET can actually be of benefit. In some cases, it is not possible for Intel PT to capture Host mode instruction execution while in Guest mode, without crashing the target; AET does not suffer from this limitation – you cannot see the instructions executed, but you can capture the occurrence of the event.

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

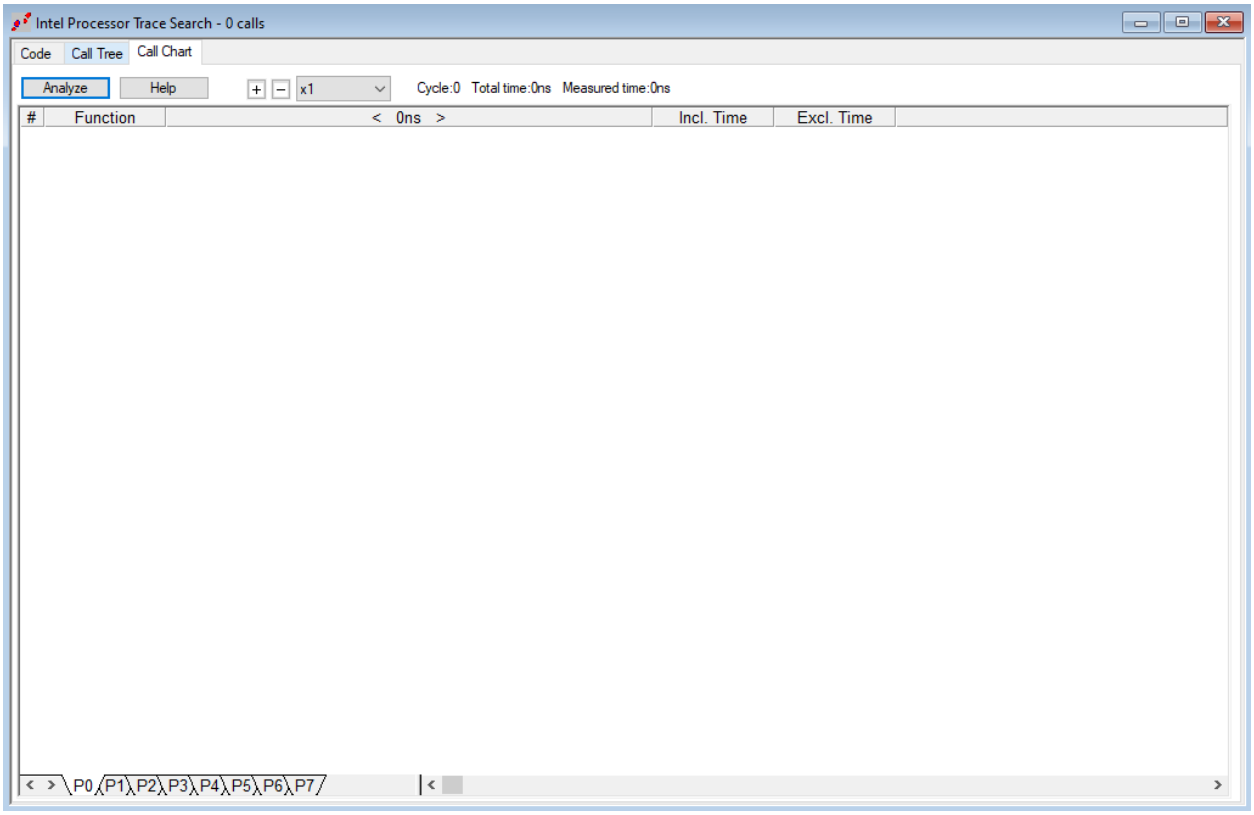
76 SEAMCALL

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, so we have not implemented it.

CONCLUSION

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