OpenVX Profiling on Synopsys EV6x Vision Processors using Tracealyzer

Percepio Application Note PA-025, 2018-10-25

Today's powerful vision processors allows for great performance, but how do you ensure your solution really makes efficient use of the hardware? Perhaps some OpenVX graph node requires much more processing time than expected and overloads a core, while the other cores are mostly idle? Perhaps the application is spending a lot of time waiting for DMA transfers to complete? Perhaps you have tried to improve performance by adding more compute resources, but the performance gain is less than expected?

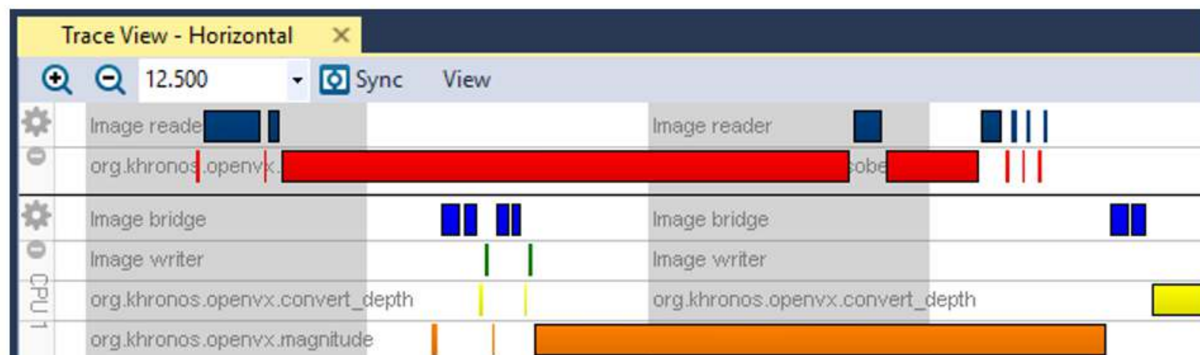
Tracealyzer for OpenVX allows you to visualize and profile OpenVX applications developed using Synopsys ARC® MetaWare EV Development Toolkit. This way you can identify bottlenecks, where optimization can make a big difference.

Tracealyzer for OpenVX provides a variety of graphical views showing different perspectives of the recorded behavior, ranging from a detailed trace view to high-level overviews and statistics. This Application Note intends to explain what you can see with Tracealyzer and how to get started.

Trace View

The trace view displays a timeline of the OpenVX graph execution, so you can to study the scheduling, pipelining and timing in great detail. The trace view can be adapted in many ways and supports both horizontal and vertical display.

An an example, we are using the demo trace provided with Tracealyzer for OpenVX ("demo_openvx.xml"). This has been recorded from a Synopsys OpenVX demo application, shown below together with a screenshot from the trace view.

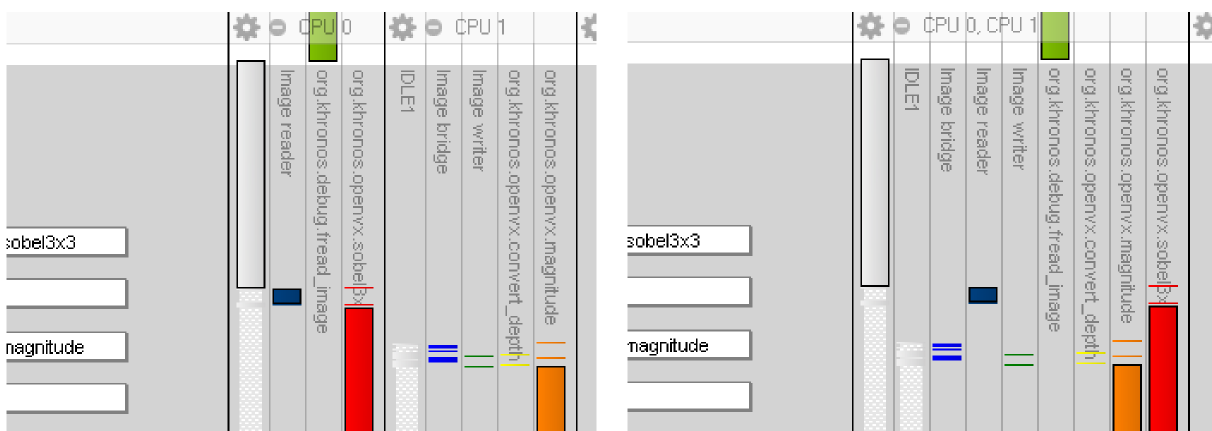


Copyright © 2018, Percepio AB

https://percepio.com

You can see how the Synopsys EV runtime (EVRT) schedules the graph using two cores. Core 0 reads the input frames and feeds it to the sobel3x3 node. The result is then processed further on Core 1, using the magnitude and convert_depth filters. Note that the processing is run-to-completion, so each rectangle (fragment) in the trace is a separate job that runs without preemptions. The short fragments of the filter functions are precondition checks, while the long fragments show the actual filter processing.

Notice how the magnitude node starts before the sobel3x3 node is completed. This is possible since the Synopsys OpenVX implementation divides each frame into tiles. One node may output multiple tiles, that are written to the output buffer one by one, as soon as completed. Thus, the following node (e.g. magnitude) does not need to wait for a full frame, but can start as soon as the first tile is available, assuming the nodes run on different cores. This allows for a pipelined processing that utilizes the cores in an efficient manner.



The Trace View is composed of fields, above labeled “CPU 0”, “CPU 1” in the left example. There are several types of fields for different types of information. OpenVX nodes are shown in a “scheduling” field, either one field per CPU core (left example) or a single field for all nodes (right example).

You may change the display in the View menu of the Trace View window. Under “View Presets” selecy one of the options “Gantt” and “Gantt (per core)”. Also note the gear icon in the top of the fields. This provides additional options for each field.

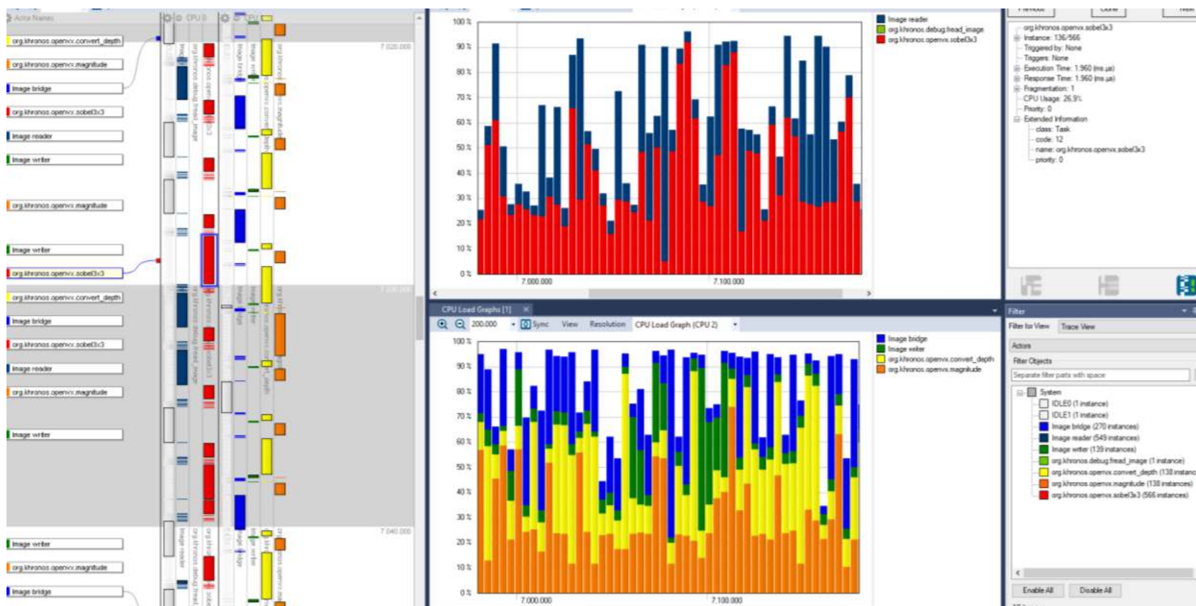
CPU Load Graph

To get an overview of how your OpenVX application utilizes your CPU cores, take a look at the CPU Load Graph, shown twice in the below screenshot (one for each core) together with the trace view in vertical mode. The CPU Load Graph allows you to see not only the overall load on your CPU cores, but also how the load varies over time and the contribution of each node.

Note: In Tracealyzer the term “CPU” really means a logical CPU core.

The CPU Load Graph also works as an overview where you can spot anomalies. For instance, note the two spikes in the ”sobel3x3” node (shown in red) where the utilization is around 80-90%. To see what causes these spikes, you can simply double-click in the CPU Load Graph to show the corresponding section in the trace view.

Copyright © 2018, Percepio AB



All views in Tracealyzer are interconnected in similar ways, which makes it easy to drill down from high-level overviews into the detailed trace. The colors are used to make it easier to identify the OpenVX graph nodes. The same color coding is used across all Tracealyzer views.

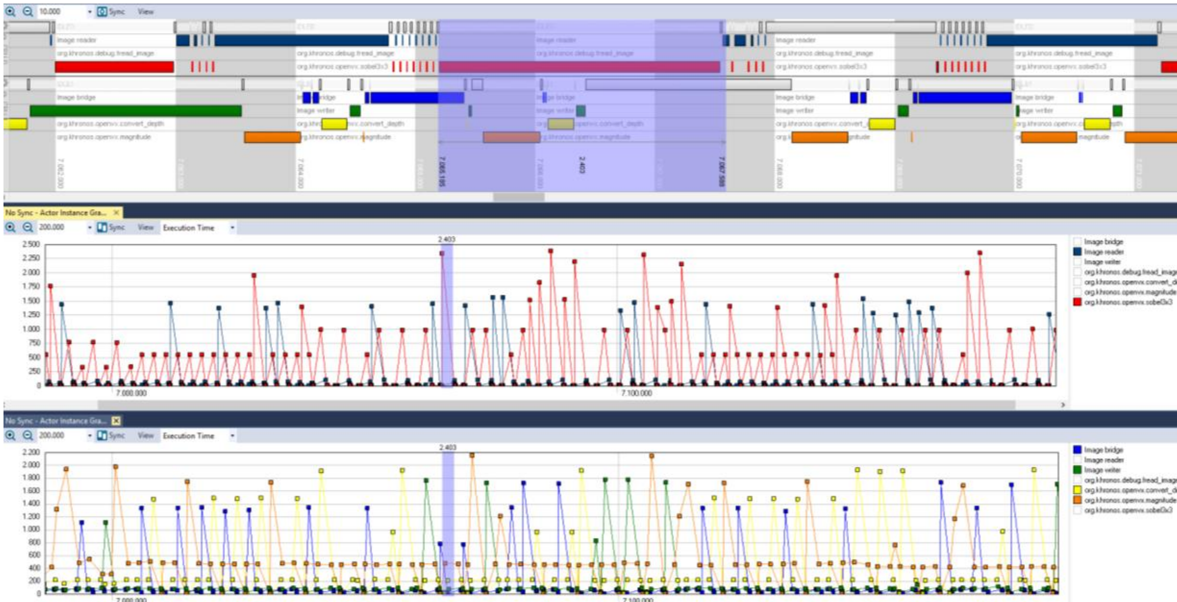
To open multiple instances of the CPU Load Graph, select “Clone View” in the local View menu and change the CPU selection in the dropdown menu. You can show all cores combined in a single CPU Load Graph by selecting “All CPUs”. Note that the CPU loads are accumulated in this mode, so with two cores the scale goes up to 200%.

The CPU Load Graph works by dividing the displayed time window into a specific number of fixed size time intervals, by default 50, and then calculates the amount of processing time used by each node within each time interval. The result is displayed as a stacked histogram, where the Y-axis shows the relative utilization within each time interval. Since the concept of CPU load is always relative to a certain time window (independent of what tool you use), zooming in or out may change the levels of the CPU Load Graph as the reference time window is changed. When zoomed in a lot, most time intervals will only contains a single node so the graph will be similar to the trace view.

Actor Instance Graph

In the below screenshot, we added two instances of the ”Actor Instance Graph”, where the Y-axis shows the execution times of the graph nodes. This way, you can see where nodes execute longer than normal and inspect the trace view to see the details.

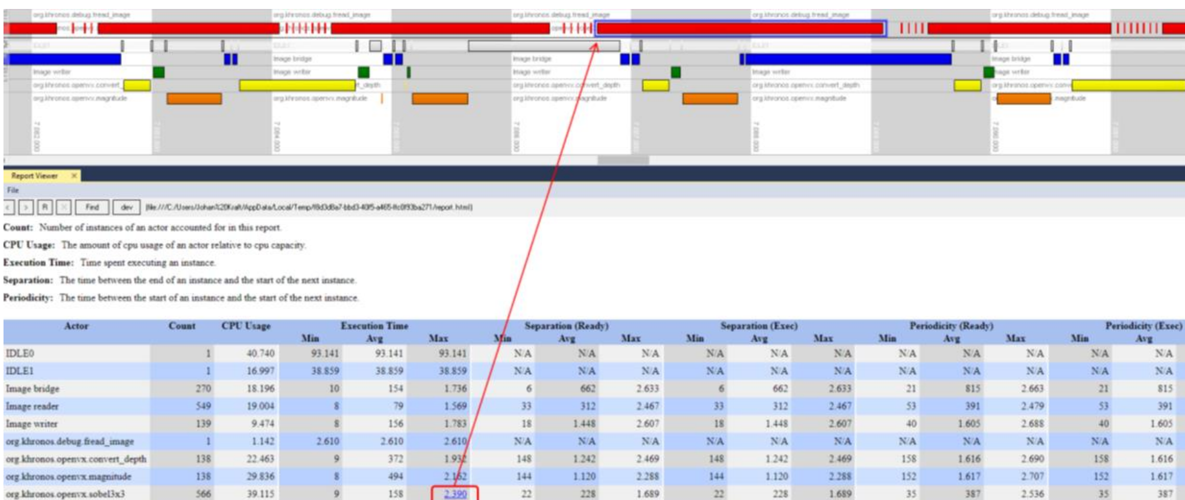
Note: “Actor” is a Tracealyzer term meaning “execution context”, corresponding to nodes in OpenVX.



The Actor Instance Graph can show various timing properties, not just execution time, but also e.g. separation and periodicity. You can change what property that is displayed in the dropdown menu that reads "Execution Time" by default. Some of the properties are however not very relevant for OpenVX traces. For instance, Response Time will always be equal to the Execution Time due to the run-to-completion scheduling.

Actor Statistics Report

The "Actor Statistics Report" gives a statistical summary of the trace, including the highest, lowest and average values observed for timing properties such as execution time. All extreme values in this report are actually links, so just click on these values to find the corresponding location in the trace view.



Actor	Count	CPU Usage	Execution Time			Separation (Ready)			Separation (Exec)			Periodicity (Ready)			Periodicity (Exec)	
			Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg
IDLE0	1	40.740	93.141	93.141	93.141	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
IDLE1	1	16.997	38.859	38.859	38.859	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Image bridge	270	18.196	10	154	1.736	6	662	2.633	6	662	2.633	21	815	2.663	21	815
Image reader	549	19.004	8	79	1.569	33	312	2.467	33	312	2.467	53	391	2.479	53	391
Image writer	139	9.474	8	156	1.783	18	1.448	2.607	18	1.448	2.607	40	1.605	2.688	40	1.605
org.khronos.debug.fread_image	1	1.142	2.610	2.610	2.610	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
org.khronos.openvx.convert_depth	138	22.463	9	372	1.932	148	1.242	2.469	148	1.242	2.469	158	1.616	2.690	158	1.616
org.khronos.openvx.magnitude	138	29.836	8	494	2.162	144	1.120	2.288	144	1.120	2.288	152	1.617	2.707	152	1.617
org.khronos.openvx.sobel3x3	566	39.115	9	158	2.390	22	228	1.689	22	228	1.689	35	387	2.536	35	387

Copyright © 2018, Percepio AB

https://percepio.com

With the Actor Statistics report you can not only find the extreme values, but with the links you can also see what was going on in the system at that time, i.e. what caused it. Although all details are not recorded you can still get valuable clues. For instance, using the Actor Instance Graph you can find other cases with similarly high execution time and check for correlations in the trace. Perhaps the high execution time only occurs under particular circumstances, e.g. due to intense activity on other CPU cores saturating the bus?

Note that you can export and save the the statistics reports, either as formatted HTML files (like above) or as tabbed text files (example below). The latter allows for easy data import into other tools and is done by checking the option “Data Export” in the Actor Statistics Report dialog.

Property Actor	Count	CPU %	ET Min	ET Avg	ET Max	Per Min	Per Avg	Per Max	Sep Min	Sep Avg	Sep Max
Task IDLE0	1	40.740									
Task IDLE1	1	16.997									
Task Image bridge	270	18.196	10	154	1736	21	815	2663	6	662	2633
Task Image reader	549	19.004	8	79	1569	53	391	2479	33	312	2467
Task Image writer	139	9.474	8	156	1783	40	1605	2688	18	1448	2607
Task org.khronos.debug.fread_image	1	1.142	2610	2610	2610						
Task org.khronos.openvx.convert_depth	138	22.463	9	372	1932	158	1616	2690	148	1242	2469
Task org.khronos.openvx.magnitude	138	29.836	8	494	2162	152	1617	2707	144	1120	2288
Task org.khronos.openvx.sobel3x3	566	39.115	9	158	2390	35	387	2536	22	228	1689

This allows you to run measurements on alternative designs and compare the resulting performance metrics in a systematic way. For instance, in the above example we can see that IDLE0 is running 40.7% of the time, meaning that Core 0 is only 59.7% loaded, while IDLE1 only runs for 17% of the time meaning that Core 1 is 83% loaded. Perhaps that idea you got last night would give higher and more balanced utilization? Just record traces and compare!

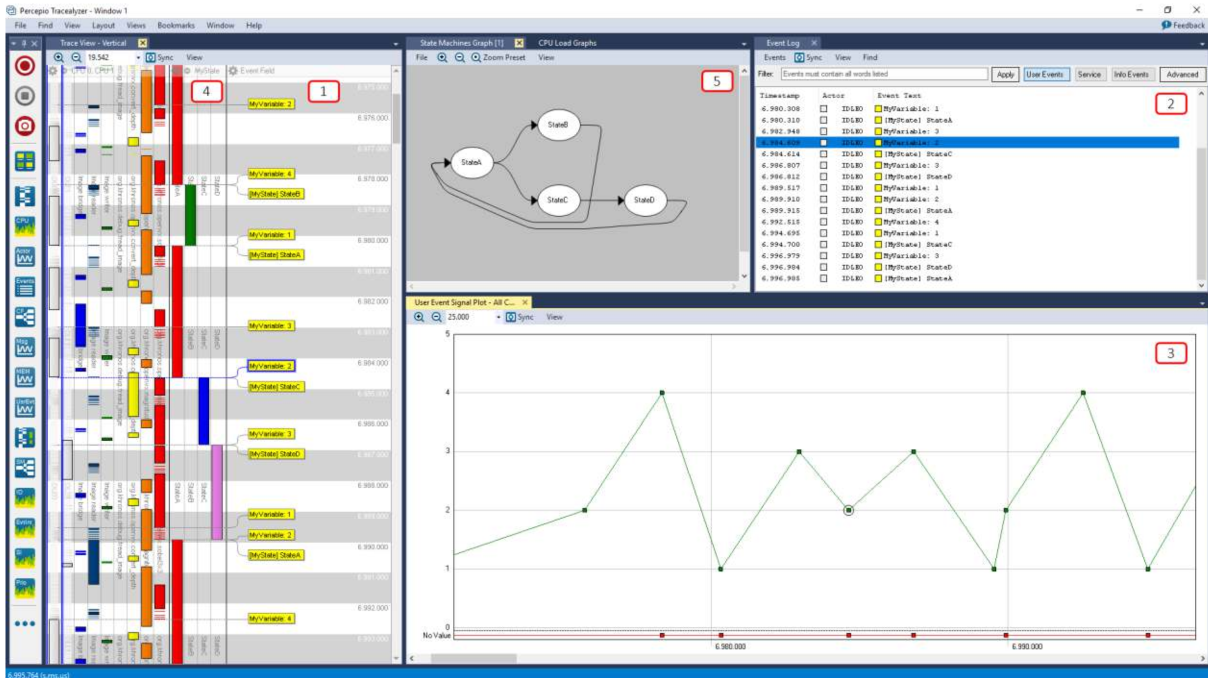
User Events

Tracealyzer allows for adding your own *user events*, i.e. custom events logged from the application code. User events allows you to visualize just about anything in your application, like diagnostic messages, variable values and states. This can be displayed in several ways.

In the below screenshot, we see an example where user events have been logged on two user event channels, ”MyVariable” showing values of an integer variable and ”MyState” showing state names. Tracealyzer can display such user events in several ways, e.g. as event labels in the trace view (1) and as entries in the Event Log (2). The User Event Signal Plot (3) allows for plotting numerical data from user events.

Moreover, if you have important state variables in your system, you can log the state changes as user events and define a State Machine in Tracealyzer to see the states in the trace view timeline (4). You can also see a summary of the state changes as a state machine graph (5). You can even get statistics on the time spent in each state, or the time between any two events by defining a ”custom interval”.

Copyright © 2018, Percepio AB



To add user events into your trace, see the below examples.

```
#include <evthreads.h>

// Register strings for two User Event Channels
EV_TRACE_OBJECT ch1 = evTraceRegisterObject(TR_USER_OBJECT, ”MyVariable”);
EV_TRACE_OBJECT ch2 = evTraceRegisterObject(TR_USER_OBJECT, ”MyState”);

// Register a string event, in this case a state name
EV_TRACE_OBJECT stateA = evTraceRegisterObject(TR_USER_OBJECT, ”StateA”);

// Log the value of var1 on channel ch1 (”MyVariable”)
evTraceCustomEvent(ch1, &var1, sizeof(var1));

// Log the string event on channel ch2 (”MyState”)
evTraceObjectEvent(ch2, stateA);
```

Use evTraceCustomEvent() to store numerical values and evTraceObjectEvent() for string events. Note that evTraceRegisterObject is used to store strings in the trace, both for user event channel names and for textual user events. For further information, please refer to the EVTracer API described in the *Synopsys DesignWare EVRT Software User’s Guide*.

Copyright © 2018, Percepio AB

Installing Tracealyzer

Visit https://percepio.com/download and make sure to select OpenVX/Synopsys as Target Platform. The download link will be sent to the provided email address. If you don't have a Tracealyzer license already, sign up for evaluation in the download registration form. You will then receive an addtional email with an evaluation license key. The evaluation time is 30 days for this version of Tracealyzer.

To install on Windows, just launch the executable installer. On Linux, extract the Tracealyzer .tgz archive and read "RunningOnLinux.txt" for further instructions.

When Tracealyser is started for the first time, select "Activate License" to enter your license key.

You can begin exploring the features right away using the example trace shown in this document. This is included with Tracealyzer and available directly from the initial "Welcome to Tracealyzer" screen.

Recording Traces with ARC MetaWare EV Development Toolkit

To record a trace from your OpenVX application, follow these steps:

1. In your application code, add the following include statement:

   ```
   #include <evtreads.h>
   ```

2. Add a call to evTraceDumpFile, like below. This should be made after your application has executed for a while, typically when a particular test has finished.

   ```
   evTraceDumpFile("dump.bin");
   ```

3. Run your application to record a trace and save it to dump.bin:

   ```
   $ make run
   ```

4. Copy the resulting `dump.bin` to $EVSS_HOME/software/tracing.
5. Convert the trace using the python script "dump_to_tracealyzer.py", found in the $EVSS_HOME/software/tracing directory:

   ```
   $ python dump_to_tracealyzer.py dump.bin trace.xml
   ```

6. Open the resulting trace.xml in Tracealyzer.

You may also note another another XML file named ARCHS.xml. This is not a trace, but a "target platform" file for Tracealyzer that describes some basic properties of the trace file structure. This is now included in the Tracealyzer for OpenVX package, so the local copy is not needed.

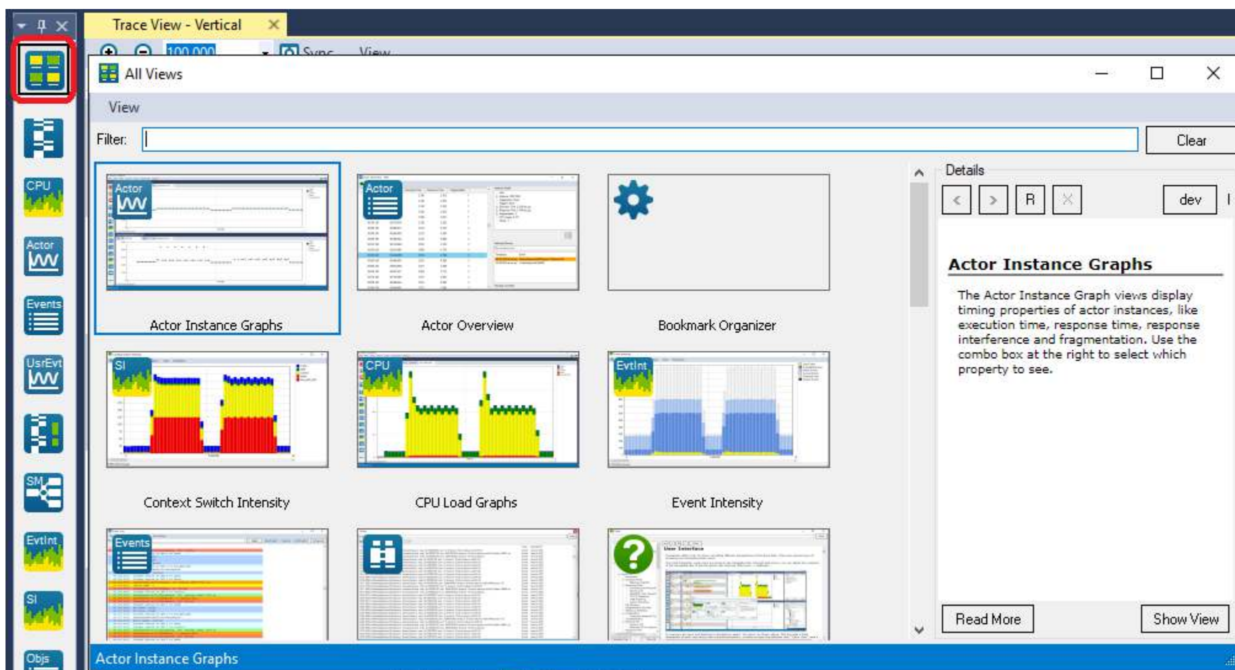
For further information, please refer to the EVTracer API described in the *Synopsys DesignWare EVRT Software User's Guide*.

Copyright © 2018, Percepio AB

Learning More

The best way to learn more about Tracealyzer is to try it yourself - there are many more features than mentioned here. We try to make them as intuitive as possible, but don't forget to check out the user manual. This is found under the Help menu. You may also press the F1 key to open the user manual section for the current view.

Also note the "All Views" option, found in the top of the Navigation bar. This lists all available views with a short summary, with buttons for opening the correponding user manual section and for opening the view.



To get started, visit https://percepio.com/download and sign up for an evaluation. An example trace in included, so you can begin exploring the capabilities of Tracealyzer directly.

In case you have any questions, don't hesitate to contact us at support@percepio.com.

Copyright © 2018, Percepio AB

https://percepio.com