ClearSpeed Visual Profiler

CSX System - Example 1

Viewing hardware thread trace information generated from the CSX700 monitor port

Advance
Accelerate
Optimize

ClearSpeed Visual Profiler

CSX System - Example 1

Introduction

This is the first example in the set of examples demonstrating the use of the visual profiler for investigating csx system activity. In this example csvprof will be used to show how the hardware thread activity of the CSX700 can be visualized. The hardware thread activity is captured from the CSX700 via the monitor port hardware on the device and the processed into a csvprof file by the csgdb debugger. The test code is a very simple assembler file which contains some code using multiple threads. The example assumes that the user is familiar with the use of csvprof from the GUI overview workshop.

Running the example

The assembler file and csgdb script file used for this workshop can be found in the directory

```
csvprof_examples/csx_system/example1
```

This example requires the use of the ClearSpeed SDK in order to assemble the example executable file.

The command line for generating the workshop file is as follows

```
cscn –o example1.csx –g threads.is
```

This will generate the executable file `example1.csx`

There is also a csgdb script for this example to allow commands to be given to the debugger in a non interactive mode.

The contents of the script file are as follows

```
connect
cstrace event icache on
cstrace event thread on
cstrace event proc on
tbreak _start
r
cstrace enable
c
cstrace dump example1
q
```

This script file contains the following commands that are executed in sequence. First the debugger connects to the CSX700 device followed by

Advance
Accelerate
Optimize

some commands to configure the event tracing. The events to trace are configured as icache, thread and processor. A temporary break point is then set at _start in the csx file and the code is started running. When the breakpoint at _start is hit the debugger will enable the trace port and continue execution. Once the code has terminated the debugger will dump out the file `example1.cst` for use with the visual profiler and then quit.

The following command can now be used to generate the trace data.

```
csgdb –x example1.gdb example1.csx
```

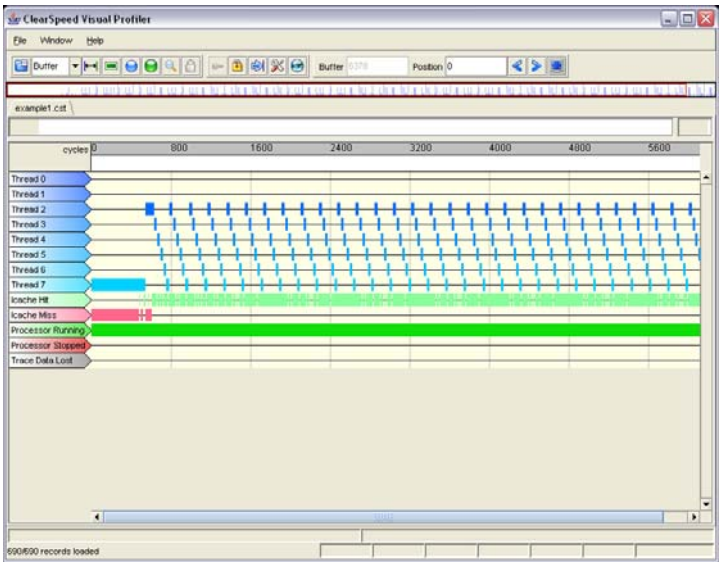
1. Starting the csvprof GUI

The csvprof GUI is started from the command line using the following command. The example trace file is passed on the command line.

```
csvprof example1.cst
```

 # Start up the visual profiler with the above command line.

The GUI should start up and appear as below.



The visual profiler has started up and you can see the events that have been traced during this session listed down the left hand side of the GUI. In this example we have all the hardware threads traced plus the Icache and the processor (which denotes if the processor is running or stopped). It is clear from the GUI that there is a repeating pattern of thread switches starting with thread 2 and moving between all the threads in order to thread 7. This then loops a number of times before terminating.

There are several things we can notice in the GUI at this point.

Advance
Accelerate
Optimize

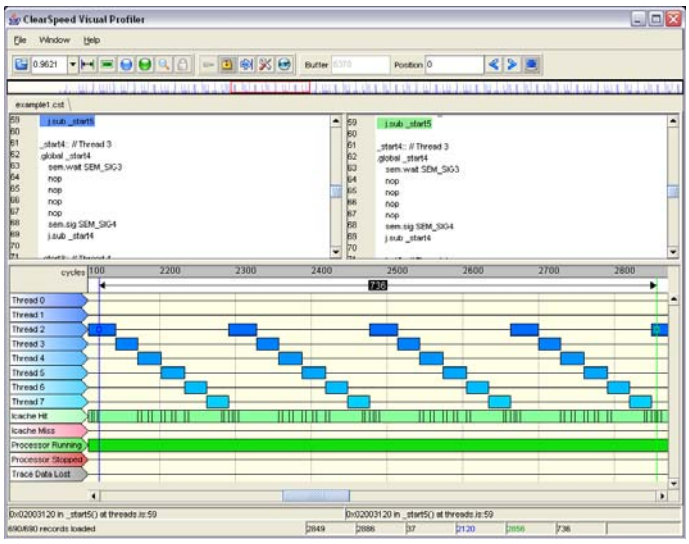
1. As you move the mouse over a particular event block the source code window displays the line the event corresponds to.
2. Moving the mouse over particular events displays the event timing info in the status bar.
3. Selecting an event with the blue marker sets a blue mark in the trace file and the source code line associated is displayed in the left source code window.
4. Using the right mouse button will place the green marker in the trace file which also sets the green event source code line in the right source code window.
5. Once both markers are set, the GUI will show the difference in CPU cycles between the blue and green markers.
6. The buffer panel at the top of the GUI shows a global overview of record density in the whole trace file.

 # Experiment with the timeline view and see the above items occur, you may also want to drag the source code divider down to see more of the example code.

We can now set some buttons from the control panel to further analyze the trace file. In order to get accurate cycle time for particular events or sequences of events it is best to zoom to a scale in the timeline view which encompasses the events to be timed. In this example we may wish to look at the cycle time for a particular loop instance.

 # Set the lock to event button from the control panel and select two points in the trace data, one with the left mouse button and the other with the right. Allow a few iterations of the loop to be between the markers and press the fit to markers button.

The visual profiler should now show a zoomed in view of the thread activity as seen below.

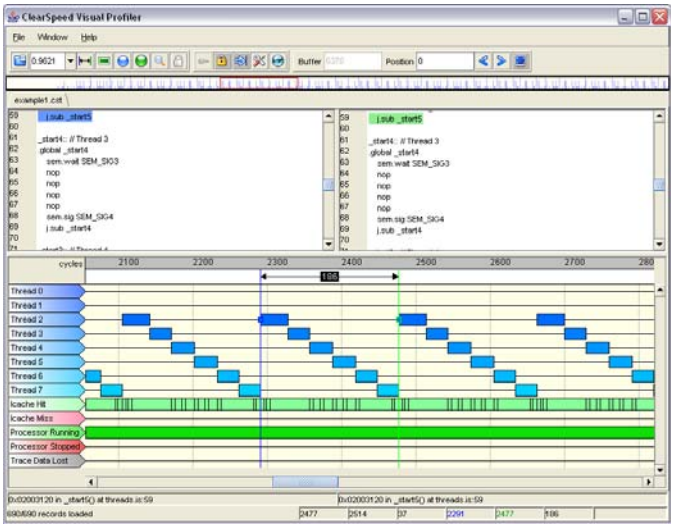


Advance
Accelerate
Optimize

You can now see the thread events are much larger in the timeline view after the zoom. We can now find out the cycle time for the loop from thread 2 back to thread 2 again. This is easy to do as we just have to set a marker at the start of 2 consecutive thread 2 events. In order to do this we can use the snap cursors to event boundaries option.

 # Set the snap cursors button and select two consecutive thread 2 start points using the left and the right mouse button.

You can now see that the time between the blue and green markers is an accurate cycle time of the loop as shown in the image below.



This demonstrates how you can use the visual profiler to time sections of code or times between particular events accurately.

For the remainder of this example feel free to experiment with the GUI and the options it has. Once finished exit the GUI from the file menu. This concludes the interactive part of this example.

Conclusion

In this example you have seen how a simple trace of hardware events maps back to software and how the visual profiler is able to provide you with a pattern for executing code. The ability to collect accurate timing in cycles from the hardware for events has been demonstrated and the use of hardware thread events has been introduced. An example of using csgdb to configure and capture trace information has been demonstrated also.

Advance
Accelerate
Optimize