ClearSpeed Visual Profiler

CSX System - Example 2

Understanding program flow using the branch trace facility of the CSX700 monitor port

Advance
Accelerate
Optimize

ClearSpeed Visual Profiler

CSX System - Example 2

Introduction

This csvprof example is designed to introduce the concept of viewing processor branch and Icache events to the user. From running the simple case in this example it is possible to see how the visual profiler allows a user to follow program flow visually. The branch trace is captured using the CSX700 monitor port and then post processed into a file to be used with the visual profiler by csgdb. As csgdb understands the relationship between program counter addresses and source code lines it is possible to provide a view of program flow at the source code level. The test code in this example is a simple set of nested function calls which then return provided as Cn code. As with the previous example the assumption is made that the user is familiar with the visual profiler GUI functionality as described in the GUI overview workshop.

Running the example

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

```
csvprof_examples/csx_system/example2
```

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 example2.csx –g mono.cn
```

This will generate the executable file `example2.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 mono on
cstrace event thread off
tbreak _start
r
cstrace enable
c
cstrace dump example2
```


Advance
Accelerate
Optimize

```
q
```

The csgdb script allows the user to run a non interactive collection of commands through the debugger which define the trace collection session. The commands are very similar to the previous example with the first command connecting to the CSX device. The events that are enabled in this instance are defined by the next 2 lines. First the collection of events related to the mono processor part of the CSX700 processor are enabled then the events related to hardware threads are disabled as they play no part in this example so there is no point explicitly tracing them. This allows a user to reduce the number of trace events collected and as a consequence the memory footprint of the trace data. The final commands in the file simply set a temporary breakpoint at the symbol _start, run to the breakpoint, enable the trace, continue the code executing and then dump the trace and exit the debugger when the debugger stops due to code termination.

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

```
csgdb –x example2.gdb example2.csx
```

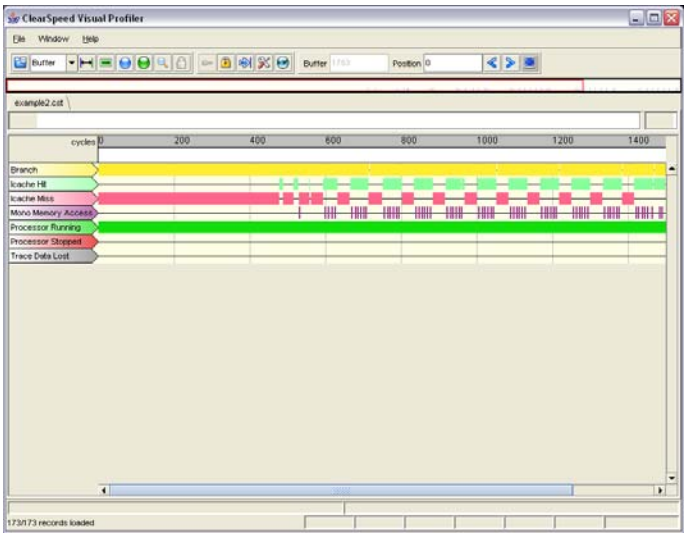
1. Starting the csvprof GUI

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

```
csvprof example2.cst
```

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

The GUI should start up and appear as below.



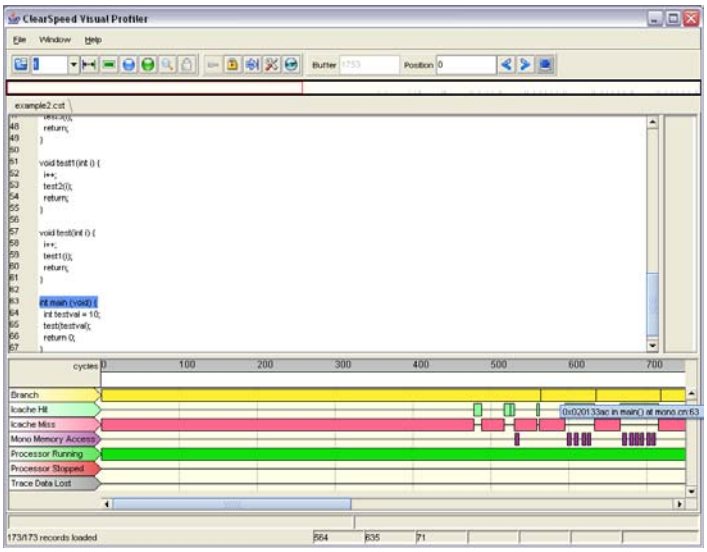
The visual profile GUI has now launched and has started up displaying the events that were enabled during the trace session run under csgdb. In this

Advance
Accelerate
Optimize

example we are looking at branch and icache activity so that we can understand how the program flow is displayed. The code for this example is a simple set of function calls and we can now look at this in the profiler. If we turn on the outline view in the GUI and move the trace view to the right we can get a view of the current trace session.

 # Set the scale to 1, resize the source code window and move the mouse over the first mark in the branch trace event, this should bring up a tool tip.

The GUI should now look as follows.



From this view we can see the branches more clearly. If you move your mouse over each of the branch events you can see which source code locations they belong to. The value that appears in the tool tip as you move over the different branch events shows the source code location at the point the branch was entered. It will always appear as the start of the code branched to. Moving the horizontal scroll bar will allow you to see the rest of the trace execution easily.

There are some things to note from this current view in the GUI.

1. We can see that the first branch is the runtime function _start which is the entry point to the code.
2. We then have branches going from main() through test() to test9() and then back down the call stack again.
3. There is a large icache miss during _start due to the initial fetch into the cache of the program code when the example is started.
4. There are icache misses on the way up the call stack as the calls to the functions are made backwards in memory and the functions are not cached.

Advance
Accelerate
Optimize

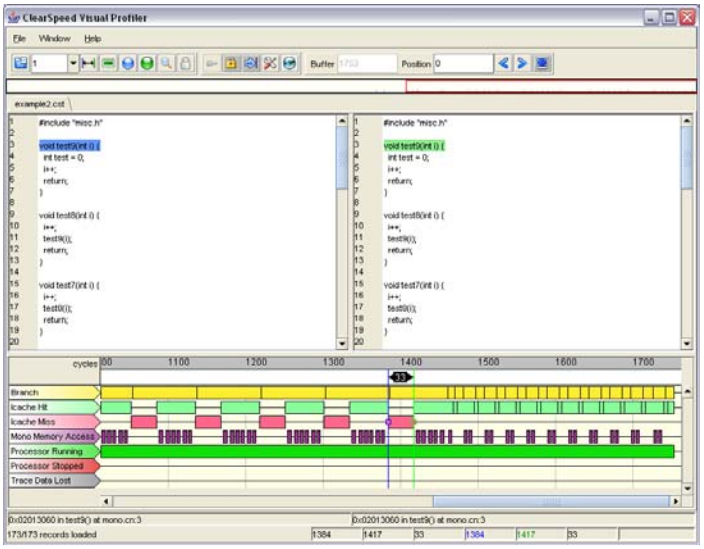
5. On the way down the call stack the branch overhead is seen when hitting the cache with the events marked on the icache hit (rather than miss) section.
6. The code executing from main()->test9() misses the cache and the return from test9()->main() hits the cache.

 # 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.

Individual event timing can be read from the status panel at the bottom of the GUI. For example if we want to find out how long a particular icache miss is for a function call we can time this in the visual profiler.

 # Move the horizontal scroll bar all the way to the right, set the lock to event and snap to event buttons from the control panel and set the left and right markers around the cache miss of function test9(). You may wish to drag the source code divider down to see more of the code.

The GUI should now look as follows.



You can see that the timing information for the cache miss has been provided by the visual profiler. All events can be timed in this way.

Feel free to experiment and when finished exit the GUI from the file menu.

Advance
Accelerate
Optimize

Conclusion

We have seen in this example that the visual profiler can reconstruct a visual representation of the flow of program counter branches within user code. It has also demonstrated that you can clearly see when the icache is responsible for performance issues when there is a cache miss. By simply moving the mouse over trace events the user can follow in their source code the calls that are made and can easily see what is happening between branch events. Once again the ability to collect accurate timing information from hardware events and the use of csgdb to capture trace information has been demonstrated.