ClearSpeed Visual Profiler

CSX System - Example 4

Visualizing PIO transactions and the transfer of data to and from the PE array

Advance
Accelerate
Optimize

ClearSpeed Visual Profiler

CSX System - Example 4

Introduction

When optimizing an application it is just as important to understand the data movement and transfer performance. This example introduces the capture and viewing of trace events related to transferring data between the mono and poly processors using the Programmed I/O (PIO) subsystem. The events for the PIO mechanism are enabled and captured via the CSX700 monitor port via csgdb and then output in a form that can be understood by the visual profiler. The source code for this example is a simple piece of Cn code which transfers data to and from the poly processor. As with the other examples in this section it assumes the user has been through the workshop provided describing the features of the visual profiler GUI in the GUI overview section.

Running the example

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

```
csvprof_examples/gui_overview/example4
```

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 example4.csx –g memcpy.cn –O3
```

This will generate the executable file `example4.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 pio on
cstrace event branch on
cstrace event icache on
cstrace event monomem on
cstrace event proc on
cstrace event polymem on
cstrace polyfilter 1;
tbreak _start
r
```


Advance
Accelerate
Optimize

```
cstrace enable
c
cstrace dump example4
q
```

The script file in this example starts by connecting to the device. In this example the events are enabled individually rather than via a group and for this trace session we enable pio, branch, icache, monomem (mono processor load / store), proc and polymem (poly processor load / store). These events will allow us to see the events which are generated by moving data between the mono and poly processors. Once these are set up the next command changes the filter on the poly processor events so they do not toggle too often. The CSX700 monitor port has limited bandwidth, and events which toggle every cycle can often use all the bandwidth it has. This is a limitation of the hardware. The script then continues by setting a temporary breakpoint, running to this point, enabling the trace port, continuing to the end of the code, dumping the trace out and then quitting the debugger. The trace file is now generated and can be viewed using the visual profiler.

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

```
csgdb –x example4.gdb example4.csx
```

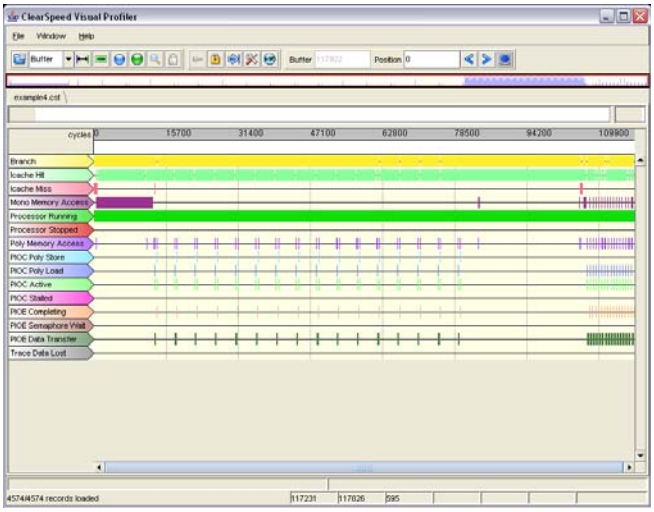
1. Starting the csvprof GUI

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

```
csvprof example4.cst
```

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

The GUI should start up and appear as below.



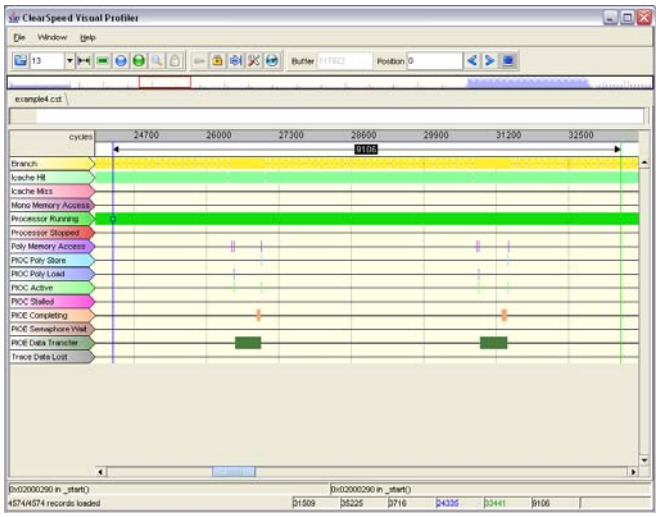
Advance
Accelerate
Optimize

You can see from the start up screen we have added the PIO event trace to the set of trace events. This allows you to see when data is being transferred to and from the poly array using the PIO mechanism. This is especially important when attempting to overlap I/O with compute. In this simple example we will only be dealing with synchronous I/O to explain the PIO trace events.

The code we have traced has a section of mono to poly data transfers at the start and then a section of poly to mono transfers at the end. The PIOC events are from the controller and these relate to the set up of a PIO transaction. The PIOE events are related to the actual transfer taking place.

 # Zoom in around a group of a 2 PIO transfers that make up the mono to poly transfer (the first set of PIO) to take a closer look.

The GUI should look as follows.



We can see now the particular events related to a single PIO transaction. Memory transfers are made up of many individual transactions when transferring data to and from the poly array.

In this example we can see that there are two parts of activity from the PIO controller and one piece of activity from the PIO engine that make up a complete transfer. This is then repeated multiple times to make up the whole data transfer operation. At the start of a transaction with data moving from mono to poly space the PIO controller shows it is active in the direction loading data from poly memory. This is the controller collecting the poly addresses for the data to be written into. The PIO engine is then active in the PIO Data Transfer event, this is the period when the data to be transferred is read from mono memory into the PIO buffer. The PIO Completing event describes the period when all of the mono to PIO buffer data transfer requests have issued but not yet completed. Once the PIO engine has finished, the PIO controller completes the transaction by storing the data from the PIO

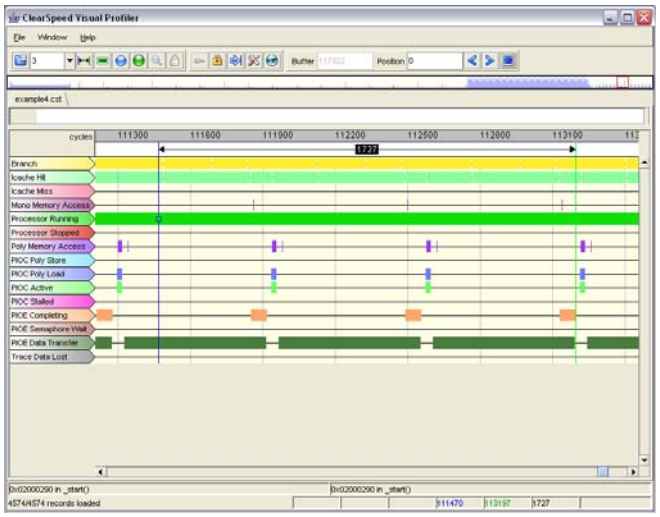
Advance
Accelerate
Optimize

buffer into poly memory. This is seen by the PIO controller activity in the store direction.

We can now move to the end of the trace to look at the activity in the other direction with the code that transfers data from poly to mono space.

 # Zoom out by using the buffer option in the scale box, set the left and right markers around a group of PIO activity towards the end of the trace and press fit to markers.

The visual profiler should now look as follows.



In this example it is visible that there is only activity in the PIO controller at the start of the transfer. The direction displayed is a load from poly memory to the PIO buffer. As this transfer is destined for mono memory there is no store after the engine completes.

This is the end of the example for PIO transactions. Feel free to experiment and when finished exit the GUI from the file menu.

Conclusion

In this example we can see that the visual profiler in conjunction with the debugger can be used to inspect the data activity between the mono and poly processors. This allows a user to get a good idea of how long data transfers last, where they start and how they relate to other code. Like all devices it is important when programming the CSX700 to optimize both the data transfer performance as well as the compute. The above example has demonstrated that the visual profiler is a very useful tool for optimizing data transfers. Again it has been shown how to use the debugger to control a trace capture session.

Advance
Accelerate
Optimize