Event Analysis with Trace

{{#switchcategory:MSP430=<McuHitboxHeader/>|C2000=<McuHitboxHeader/>|Stellaris=<McuHitboxHeader/>|TMS570=<McuHitboxHeader/>|MCU=<McuHitboxHeader/>|MAVRK=<MAVRKHitboxHeader/>|<HitboxHeader/>}}

Contents

Event Analysis

Overview

Event Analysis with Trace is a method of analyzing the occurrences of events throughout the execution of an application. Specifically, we are looking for a correlation between events and specific program addresses. One type of event that is commonly of interest in this a cache miss event. With the data captured by trace and the processing script, we can identify the program addresses that most commonly cause cache misses. We can than take that data and try to improve the application performance by taking steps to avoid these misses and the associated stall cycles.

Methodology

Event Analysis data is effectively captured by storing the Program Address whenever the event (or set of events) that we have chosen occurs. The captured data is the analyzed to find all program addresses where this/these event(s) occurred and , more importantly, to specify which program addresses routinely cause these events. The program addresses are then binned with their associated function and the function level results are provided.

⚠ **Warning** Be aware of the limitation of trace being unable to execute trace triggers when in the delay slot of a branch or within an SPLOOP. Assume a case where, within an SPLOOP, we write data values to memory. Assume that the sploop executes 100 times, and every write causes an L1D write miss. Because of the SPLOOP we will only capture a single event, even though all 100 iterations would have caused an event to occur. The misses for this function will be significantly under-represented. Be aware that this scenario is not an ideal solution if we're trying to detect misses within an SPLOOP'ed function.

Limitations

- As mentioned above, in the warning box, events within SPLOOPS and delay slots of branches may be underrepresented in the output.
- The methodology behind this scheme is to store PC samples when an selected set of events occurs. (i.e. this is Standard Trace, not Event Trace). As such, the output only contains the PC values that were captured, and no information about the events. So, if multiple events are chosen for analysis, the OR'ed combination of these events is used to trigger trace. But from the trace output, there is no way to tie the specific PC sample back to the specific event. So, we know that for each sample, one of the events that we chose occurred, but there is no way to determine which one. If this type of functionality is needed, a better alternative is to only use a single event at a time, or to use Event Trace.

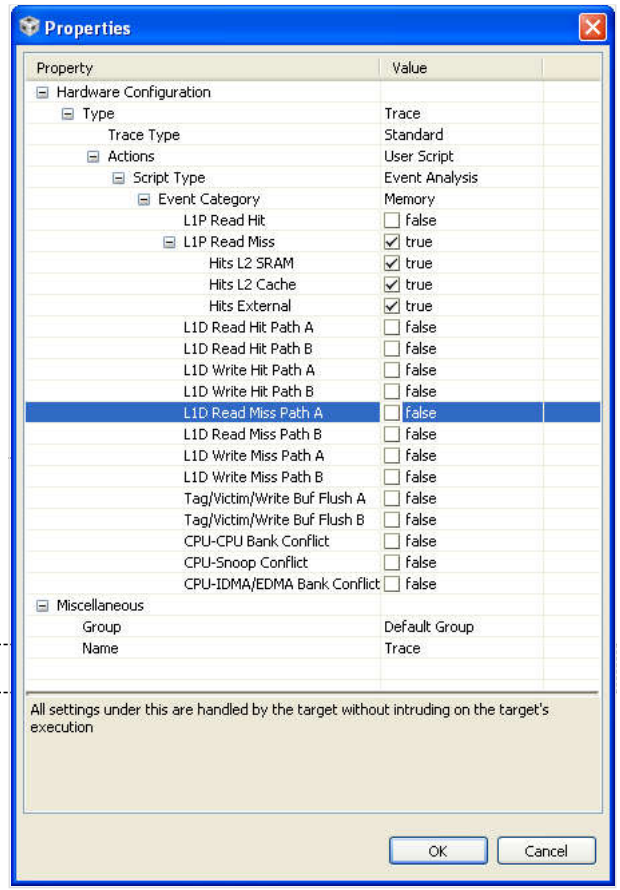
Capturing Event Analysis Data

Configure trace in the Trace Control menu as desired. Typically, "Stop on Buffer Full" mode is used with Event Analysis, but it can also be used with "Circular Buffer" mode. If using an XDS560T, configure the desired trace buffer size. A larger buffer will capture much more data, but the data will take a longer time to process. A smaller buffer won't get nearly as much of the application, but will be post processed very quickly. The compression of data when capturing via this method will not yield nearly as much data as in some other cases, so the penalty for using a very large trace buffer will be much smaller.

The Unified Breakpoint Manager (UBM) Plugin in Code Composer Studio can be easily configured to capture Pipeline Stall Analysis data

- From the Breakpoint window, create a new Trace job.
- Edit the properties of the job, and select Trace Type->Standard, Actions->User Script, and Script Type->Event Analysis as shown in the image.
- Expand the Script Type Option
- Choose the event category
- Expand the Event Category and select the event(s) that you wish to analyze.
- Click OK to save the configuration and ensure that the job is enabled in the breakpoint window.
- Run the application. You should see trace data being captured in the Trace Display Window.

In the example shown at right, we are analyzing all of the L1P Read Misses. In effect, the script will tell us on a function level basis which functions cause the most L1P misses. In some cases, if there are two functions that call each other that share the same cache line, they will continually evict each other from the cache. This is a good example of the type of issue that can be detected by this method. Linking these funcitons consecutively in memory can alleviate the cache conflicts.



We will use the Stand Alone trace decoder to get the trace data for our script. The Stand Alone decoder is shipped with CCSv4 and is located at the following location <CCS4_INSTALL_DIR>\ccsv4\emulation\analysis\bin\td.exe. It can be called from the command line. Use the command td.exe -help to get details on the switches to be passed to the stand alone decoder. CCSv4 will create a file called XDS560_RecTraceData.bin in the <CCS4_Install_Dir>\ccsv4\emulation\analysis\bin\logs directory automatically whenever trace is captured. This is a copy of the raw data trace file. We will pass this file to the trace decoder to be processed.

Generating the func_info file

The first step in processing the trace data is to create a func_info.csv file from your .out file using the utility ofd6x.exe. This is a utility that is shipped with the code generation tools. The file that we need to generate will contain a list of all of the functions and filenames, along with the start, end, and length of each function. The utility ofd.exe is also shipped with the trace script package. It is located in the utils directory. Create the func_info file with the following command line

```
ofd6x --func_info <.out file name> > <.csv file name>
```

The <.out file name> parameter specifies the location and name of the .out file. The <csv file name> specifies the name of the file to be generated. You can choose whatever name you wish, but it is recommended to use the .csv extension. This file only needs to be regenerated when the .out file changes. One option is to have CCS execute this command as a post-build step every time you build your application so you will always have an up to date func_info file.

Decoder Prerequisites

In order for the Pipeline Stall Analysis application to be able to process the data, the following is required of the output of td.exe.

- The following fields must be included in the output (Default is All fields, which is fine)
  - Program Address
  - Trace Status
- The format must be in CSV_NO_TPOS_QUOTE form (-format CSV_NO_TPOS_QUOTE **This is the default**)

Command Line

One of the following commands can be used to process the trace data. Note that full or relative paths to each file supplied must be provided if all files are not in the current directory.

td.exe -bin XDS560_RecTraceData.bin -app <out file> <trace_decoder_options> | perl trace_event_analysis -fxns_input=<func info .csv>

or

td.exe -bin XDS560_RecTraceData.bin -app <out file> <trace_decoder_options> | perl trace_event_analysis.pl -f=<func info .csv>

or

td.exe -bin XDS560_RecTraceData.bin -app <out file> <trace_decoder_options> | trace_event_analysis.exe -f=<func info .csv>

Sample Results Data

The results data is output through stdout in comma separated value form. It's essentially a list of function/file names and the number of times the specific event occurs in each. A header line is prepended to the output, detailing the contents of each field. It can be supressed by using the --no_header option with the script.

```
Function,Filename,Count
--------,--------,-----
"task","slice.c",7995
"LNK_F_dataPump","src/lnk/rtdx.s62",7902
"LNK_F_getChanPtr","src/lnk/rtdx.s62",7711
"_KNL_check","src/knl/knl_chec.s62",5167
"_IDL_run","src/idl/idl_c.s62",5138
"RTA_F_dispatch","src/rta/rta.s62",5134
"KNL_run","src/knl/knl_run.c",2681
"_LOG_event","src/log/log.s62",2621
"__divu","./divu.asm",2571
"_STS_delta","src/sts/sts.s62",2571
"LNK_F_checkBufferCall","src/lnk/rtdx.s62",257
```

The results can be piped to a file or to another script for further processing.

Trace Script Download

The script package can be downloaded at the following location https://www-a.ti.com/downloads/sds_support/applications_packages/trace_csv_scripts/index.htm

Note that in order to use the stand alone trace decoder, you must have version 3.0.0 of the scripting package or later.

Trace Script Frequently Asked Questions

Generic Scripting FAQS

- Q: Why do I sometimes see "UNKNOWN", "UNKNOWN" in the output for functions/filenames
  - A: The function/filename symbols are determined from the output of the ofd6x.exe (Object File Dump utility), which generates a list of functions and filenames from the .out file, along with their staring and ending addresses. If "UNKNOWN" values are showing in the output, it's because trace captured program execution that had a program address outside the ranges specified by the OFD utility. Common causes might be code that has been dynamically loaded/allocated which wouldn't have associated information in the .out file. You can usually determine the exact cause by looking in the file generated by ofd6x.exe and the .map file and determining why the offending program address is not defined.

Event Analysis Specific FAQS

{{

1. switchcategory:MultiCore=

- For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum
- For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum

Please post only comments related to the article **Event Analysis with Trace** here.

Keystone=

- For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum
- For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum

Please post only comments related to the article **Event Analysis with Trace** here.

C2000=*For technical support on the C2000 please post your questions on The C2000 Forum. Please post only comments about the article **Event Analysis with Trace** here.*

DaVinci=*For technical support on DaVincoplease post your questions on The DaVinci Forum. Please post only comments about the article **Event Analysis with Trace** here.*

MSP430=*For technical support on MSP430 please post your questions on The MSP430 Forum. Please post only comments about the article **Event Analysis with Trace** here.*

OMAP35x=*For technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article **Event Analysis with Trace** here.*

OMAPL1=*For technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article **Event Analysis with Trace** here.*

MAVRK=*For technical support on MAVRK please post your questions on The MAVRK Toolbox Forum. Please post only comments about the article **Event Analysis with Trace** here.*

*For technical s please post yo questions at http://e2e.ti.cor Please post on comments abo article **Event Analysis with** here.*

}}

Links

Amplifiers & Linear
Audio
Broadband RF/IF & Digital Radio
Clocks & Timers
Data Converters
DLP & MEMS
High-Reliability
Interface
Logic
Power Management
Processors
- ARM Processors
- Digital Signal Processors (DSP)
- Microcontrollers (MCU)
- OMAP Applications Processors

Switches & Multiplexers
Temperature Sensors & Control ICs
Wireless Connectivity

{{#switchcategory:MSP430=<McuHitboxFooter/>|C2000=<McuHitboxFooter/>|Stellaris=<McuHitboxFooter/>|TMS570=<McuHitboxFooter/>|MCU=<McuHitboxFooter/>|MAVRK=<MAVRKHitboxFooter/>|<HitboxFooter/>}}