## History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author(s)</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.1</td>
<td>July 3, 2012</td>
<td>Wenzhong Liu</td>
<td>First Draft Created.</td>
</tr>
<tr>
<td>0.0.2</td>
<td>July 13, 2012</td>
<td>Bhavin Kharadi</td>
<td>Formatted and review comments added.</td>
</tr>
<tr>
<td>0.0.3</td>
<td>July 20, 2012</td>
<td>Wenzhong Liu</td>
<td></td>
</tr>
<tr>
<td>0.0.4</td>
<td>July 24, 2012</td>
<td>Wenzhong Liu</td>
<td>Add wiki link for XDS560v2-Pro.</td>
</tr>
<tr>
<td>0.0.5</td>
<td>Aug 29, 2012</td>
<td>Rinkes Dan</td>
<td>Updated for Processing Data Section</td>
</tr>
<tr>
<td>0.0.6</td>
<td>Aug 29, 2012</td>
<td>Bhavin Kharadi</td>
<td>Formatted.</td>
</tr>
</tbody>
</table>
CONTENTS

1 Introduction ............................................................................................... 6
  1.1 Purpose .............................................................................................. 6
  1.2 Scope ................................................................................................. 6
  1.3 Terms & Abbreviations .......................................................................... 6
  1.4 References .......................................................................................... 6
2 Overview ................................................................................................... 7
3 Using External Trace Receiver for DSP Trace .................................................. 8
4 Using ETB for DSP trace .............................................................................. 9
  4.1 Select and Configure Trace Receiver ....................................................... 9
  4.2 Define a Trace Job for Controlling Trace Start or Trace Stop .................... 11
    4.2.1 Insert Trace Control Points ............................................................ 12
    4.2.2 Define Trace Jobs via CCS ............................................................. 12
  4.3 Run Application & Display Trace Results................................................. 16
5 Use ETB+EDMA for DSP Trace .................................................................... 19
6 Generating Instructions Per Cycle from Trace Data ....................................... 20
  6.1 Capturing the Trace Data .................................................................... 20
  6.2 Exporting the Data ............................................................................. 20
    6.2.1 Required Fields ............................................................................ 21
  6.3 Processing the Data ........................................................................... 22
    6.3.1 Script Package ............................................................................. 22
    6.3.2 Executing the Script ..................................................................... 22
    6.3.3 Results ....................................................................................... 22
FIGURES

Figure 1 Connect “XDS560v2-Pro” and “XDS560v2” to a Target Board ..................... 8
Figure 2 Trace System Control Window .................................................................... 9
Figure 3 Trace Receiver Selection Window ............................................................. 10
Figure 4 ETB Configuration Window ....................................................................... 10
Figure 5 XDS560v2-Pro Configuration Window ..................................................... 11
Figure 6 Trace Job Creation .................................................................................. 12
Figure 7 Configure trace job(s) to control trace start/stop – the default. ............... 13
Figure 8 Configure trace job(s) to control trace start/stop – PC based. ................. 14
Figure 9 Enable trace job(s) before start trace. .................................................... 15
Figure 10 Configure trace job(s) to control trace start/stop – Mark based ............. 16
Figure 11 Trace Results Display ............................................................................ 17
Figure 12 Configuration Options of Trace Display ............................................... 18
Figure 13 Load 'ALL' data .................................................................................... 20
Figure 14 Export 'All' Data ................................................................................... 21
Figure 15 Sample Output ....................................................................................... 22
TABLES

No table of figures entries found.
1 Introduction

1.1 Purpose

Purpose of this document is to explain in detail the process of generating trace and then using those trace data to generate the Instructions per cycle number for an application. These instructions per cycle number for an application are used to generate the power estimates using power spreadsheet provided on the Keystone family device product folder pages.

1.2 Scope

The scope of this document is limited to explain steps for generating IPC numbers.

1.3 Terms & Abbreviations

<table>
<thead>
<tr>
<th>Term/Abbreviations</th>
<th>Full Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETB</td>
<td>Embedded Trace Buffer</td>
</tr>
<tr>
<td>IPC</td>
<td>Instructions Per Cycle</td>
</tr>
<tr>
<td>CCS</td>
<td>Code Composer Studio</td>
</tr>
</tbody>
</table>

1.4 References
2 Overview

CPU trace technology is an important means for debugging and optimizing application codes. TI’s KeyStone devices have the trace capabilities built in, and TI’s CCS fully supports this important debug tool.

There are three ways to use trace for KeyStone devices:

1. Use an external trace receiver (XDS560v2-Pro) – this can trace large amount of data.

2. Use internal ETB trace buffer – there is one dedicated ETB buffer (4kBytes) for each DSP core on KeyStone devices.

3. Use ETB+EDMA for large amounts of trace data – overcome the limitation of the ETB buffer size. But user needs to integrate the application code with TI’s cToolsLib – AETLib, ETBLib, and DSPTraceLib.

This user guide shows how to use DSP trace for application profiling, step by step.
3 Using External Trace Receiver for DSP Trace

There are two kinds of trace receivers that exist today - XDS560T and XDS56v2-Pro. The XDS560v2-Pro is the latest one, and is recommended for Keystone devices.

Following are the picture of the XDS560v2-Pro:

![Figure 1 Connect “XDS560v2-Pro” and “XDS560v2” to a Target Board](image)

Here is the wiki link to the latest status of XDS560v2-Pro products, and user guide of it. [http://ap-fpDSP-swapps.dal.design.ti.com/index.php/XDS_Pro_Trace#Getting_Started](http://ap-fpDSP-swapps.dal.design.ti.com/index.php/XDS_Pro_Trace#Getting_Started)

Other than the difference in hardware settings, the procedures of using ETB and using external Trace Receiver via CCS are very similar. So, we cover them together in next section - “Using ETB for DSP trace”, and show the differences in the Figure 3, Figure 4, and
4 Using ETB for DSP trace

4.1 Select and Configure Trace Receiver

Before enable trace for a specific trace job, we need to select and configure the trace receiver. This should be done after CCS has connected to a target to be traced.

1. Open trace control window: Tools->Trace Control.

The receiver configuration window will pop-up with one tag associated to one of connected targets (here, we have only one target – C66xx_0).

![Figure 2 Trace System Control Window](image)

2. Change the receiver if it is needed, by clicking on the option - “Receiver”. Then, select the receiver in following window, and click on OK.
3. Assuming the ETB buffer is selected. Following are typical settings to use:

4. Assuming the Pro Trace is selected. Following is a typical settings:
5. Click OK. CCS will start to setup the device as well as the receiver. This process might take some time for clock setting, and signal training, etc.

### 4.2 Define a Trace Job for Controlling Trace Start or Trace Stop

The simplest way to control a trace job is to trace everything blindly by setting a trace job as "Trace On" via CCS. In this way, whenever the target is running, CCS will enable the trace function and start to capture the traced data.

A better tuned trace session will only turn on/off trace at desired location of application. Typically, we can define a trace job for controlling trace start or stop:

1. Based on a PC value.
2. Based on the execution of a special instruction – the MARK instruction.
4.2.1 Insert Trace Control Points

Other than get an absolute PC value from disassembled code, user can use named variables such as a sub-routine's name. A better way is to define some global labels and use them as following:

1. Define a macro:
   
a. `#define INSERT_LABLE(x) asm("\t.global "x); asm(x": nop");`

2. Insert global labels in application code where they are needed, such as:
   
a. `INSERT_LABEL("TraceStart");`
b. `INSERT_LABLE("TraceStop");`

Then the named labels - TraceStart and TraceStop, can be used to define trace jobs.

For “mark” instructions, they can be inserted in any place in your C code, as:

```c
_mark(n);
```

Here, n = 0, 1, 2, or 3.

4.2.2 Define Trace Jobs via CCS

Following examples assumes that:

1. Target has been connected, and trace receiver has been selected and configured.
2. Application code has been loaded with following control points being inserted:
   a. Two global labels – TraceStart and TraceStop
   b. DSP instructions - mark1 and mark2 (can be placed in multiple places).

Here are the steps to configure trace jobs and enable them via CCS:

1. Open Breakpoints view: View->Breakpoints.
2. Click on the target to be traced.
3. Create two trace jobs (dummy): Click on the New Breakpoint button, and select Trace (twice):
4. Configure the trace job(s) - right click on each of trace jobs, then select Properties, you will see following default setting (after expand some items):

![Figure 7 Configure trace job(s) to control trace start/stop – the default.]

5. Change the Actions from "Trace On" to "Start Trace". Make sure settings - Trigger Type is "PC", and Location Type is "Point".
7. In "what to trace", select program address and timing.
8. Optionally, we change the Name as “TraceStart” (default is Trace).
9. Click on OK. The configuration for TraceStart job is done.
Figure 8 Configure trace job(s) to control trace start/stop – PC based.

10. Similarly to configure trace stop job, with following difference:
   a. Change “Trace On” to “End Trace”.
   b. Enter “TraceStop” to the Location option.
   c. In “what to stop”, select program address and timing.
   d. Optionally, change the name to “TraceStop”.

11. Enable both trace jobs by clicking on the marks.
12. Then, you can start to run your application.

The trace will be started whenever, the PC is running over location "TraceStart", and will stop whenever, run over location "TraceStop".

The above example is using PC values to configure trace start/stop. Similar steps are applicable to using MARK instructions, with following difference:

1. Use trigger type "Event"
2. Use Event Category "System"
3. Mark the “MARK INSTRUCTION” under “System”.
4. Mark the “MARK 1” for trace start (assuming MARK1 is for your trace start).
5. Or, mark the “MARK 2” for trace stop (assuming MARK2 is for your trace stop).
4.3 Run Application & Display Trace Results

After configured trace receiver, defined and enabled trace jobs, user can start to run the application. Normally, CCS will trace the application as expected, and display the traced results (decoded) after target gets halted.

Here is the typical display of traced results:

![Breakpoint Properties](image)

<table>
<thead>
<tr>
<th>Properties</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Configuration</td>
<td>Trace</td>
</tr>
<tr>
<td>Type</td>
<td>Trace</td>
</tr>
<tr>
<td>Trace Type</td>
<td>Standard</td>
</tr>
<tr>
<td>Actions</td>
<td>End Trace</td>
</tr>
<tr>
<td>Trigger Type</td>
<td>Event</td>
</tr>
<tr>
<td>Trigger Skip/Delay</td>
<td>No</td>
</tr>
<tr>
<td>Event Category</td>
<td>System</td>
</tr>
<tr>
<td>CPU pipeline stall</td>
<td>false</td>
</tr>
<tr>
<td>CPU Interrupt/Exception</td>
<td>false</td>
</tr>
<tr>
<td>MARK Instruction</td>
<td>true</td>
</tr>
<tr>
<td>MARK 0</td>
<td>false</td>
</tr>
<tr>
<td>MARK 1</td>
<td>true</td>
</tr>
<tr>
<td>MARK 2</td>
<td>false</td>
</tr>
<tr>
<td>MARK 3</td>
<td>false</td>
</tr>
<tr>
<td>Interrupt Controller</td>
<td>false</td>
</tr>
<tr>
<td>External Pin Input</td>
<td>false</td>
</tr>
<tr>
<td>Memory Events</td>
<td>false</td>
</tr>
<tr>
<td>Emulation</td>
<td>false</td>
</tr>
<tr>
<td>Signal Type</td>
<td>Edge to High</td>
</tr>
<tr>
<td>What to End</td>
<td>(Select data to end)</td>
</tr>
<tr>
<td>Program Address</td>
<td>true</td>
</tr>
<tr>
<td>Time Stamp</td>
<td>true</td>
</tr>
<tr>
<td>Write Address</td>
<td>false</td>
</tr>
</tbody>
</table>

*Figure 10 Configure trace job(s) to control trace start/stop – Mark based*
Figure 11 Trace Results Display

Optionally, user can control what to display by right clicking on the display, and changing the column settings as following:
Figure 12 Configuration Options of Trace Display
5 Use ETB+EDMA for DSP Trace

Using ETB for trace is easy and simple. But the size of the trace buffer is limited - only 4k bytes for each DSP ETB, which limits its use for some applications.

To overcome this, KeyStone devices support using EDMA to drain data from ETB buffer to a much large system memory while the ETB is collecting new trace data.

Typically, this will need application code itself to setup ETB, DSP trace, as well as EDMA. Moreover, certain transportation mechanism is needed to export the traced data from system memory to a host for decoding and displaying.

TI has provided the libraries, called cToolsLibs, for such embedded debug use cases. All the libraries, including source codes and examples, can be downloaded from: https://gforge.ti.com/gf/project/ctoolslib/frs/. Here is a link to an article regarding to CtoolsLib - http://processors.wiki.ti.com/index.php/CToolsLib_Article.

The specific example for DSP trace with ETB+EDMA, is: Examples\C667x\DSPTrace_TIETB_EDMA_Ex_CorePacN.

This example uses AETLib to define two trace jobs – one for trace start and another for trace stop. It uses the ETB event – ETB Buffer Half Full, to trigger EDMA transfer to drain data in ETB buffer.

In this example, even though, the application itself sets up the DSP trace, ETB, as well as EDMA, but, CCS is used for loading the code and for exporting traced data from system memory to a file on host (PC).
6 Generating Instructions Per Cycle from Trace Data

We can calculate the average number of instructions executed per cycle based on PC Trace Output captured throughout portions of the application. The trace data is captured and then exported to a comma separated value (.csv) file, which is then processed by a script that generates Instructions per cycle numbers.

6.1 Capturing the Trace Data

Capturing Data for calculating Instructions per cycle requires the same set of procedures for capturing trace data that are discussed in Section 4. The data required is simply PC trace data + timing. It is up to the individual user as to the region of the Trace data capture. For example, if it is desired to only calculate Instructions per Cycle for a single function, then trace capture should be started upon entry to a function and stopped upon the function exit. If the use case requires analysis of an entire application, then trace should be on continuously.

6.2 Exporting the Data

Once the data has been captured, it must be exported to a Comma Separated Value (.csv) file so that it can be analyzed via a script. Once the data has been captured into the trace display, it is important that you have the Trace analyzer load all of the samples prior to exporting. Do this by clicking the "All" button at the top of the Trace Analyzer window as shown in . Note that it may take a very long time (sometimes hours) for the all of the data to be loaded and decoded if it has been captured into a large external buffer.

![Figure 13 Load 'ALL' data](image-url)
Once all of the data has been loaded, we now export it to a .csv file. There are 4 required fields that must be included in the output. These are listed below. Any other fields are optional and will not be used in the processing of the data. Including only the required data in the output may decrease processing time as there will be less data for the script to process. The data which is included in the output is controlled by which data is displayed in the window. These columns can be selected by right clicking on in the table and choosing “Column Settings”. Fields which are checked are included, those that are not checked will not be included. The order of the columns does not matter.

### 6.2.1 Required Fields

There are four fields which must be included:

- Program Address
- Program Data (Code)
- Cycles
- Trace Status

You can export the data by right clicking in the table and choosing Data -> Export All, as shown in Figure 14 Export ‘All’ Data. Once you do this, a dialog box will open and allow you to export the data to the location and the file name of your choice. You can use whatever name, location, and extension that you like, but the .csv extension is recommended.

![Figure 14 Export ‘All’ Data](image)
6.3 Processing the Data

Once the data has been captured and exported to a .csv file, it can then be processed by the script to generate the Instructions per Cycle data.

6.3.1 Script Package

The script used to process this data will come bundled in a package called Trace CSV Scripts. These scripts are available here. The Instructions Per Cycle script will be included in future revisions System Analyzer.

In the mean time, you can read about and download a .zip file that contains the new script and the parser module from the wiki. See the Wiki page here for the link.

6.3.2 Executing the Script

The script requires a perl installation. This script was tested with ActiveState Perl 5.16.0 which can be downloaded for free from the ActiveState site http://www.activestate.com/activeperl

Once Perl is installed, ensure that the location of perl.exe is in the path. Then, from the command line you can execute the script with the following command. This can be done from within the directory where the perl script is located. It can be done from other directories as well, but the full path to the perl script must then be provided. A full path to the .csv file is also necessary if executing from a different directory than where the .csv file is located. Use the following command.

perl  trace_func_unit_efficiency.pl --trc_input <csv_file_name>

6.3.3 Results

The resulting computations will reported via STDOUT. Results will appear similar to those in Figure 15 Sample Output.

![Figure 15 Sample Output](image-url)
The following pieces of data are reported.

- **Execute Cycles** – This is a count of the total number of execute cycles encountered throughout the trace data. Execute cycles represent all cycles where the CPU was not stalled.

- **Stall Cycles** – This is a count of the total number of cycles throughout the trace data where the CPU is stalled.

- **Total Cycles** – This is the total number of cycles that are contained within the trace data. This is the sum of the Execute and Stall cycles.

- **Instructions Per Cycle with Stalls** – This is the Instructions per Cycle measurement, including stall cycles in the calculation. This is likely the preferred number for estimating power consumption because a stall does not consume any functional units.

- **Instructions Per Cycle neglecting Stalls** – This is the Instruction per Cycle, considering only execute cycles. This number is more relevant in determining the instruction scheduling efficiency.

- **Peak Efficiency** – This is a measurement that is taken throughout the processing, using the Instructions Per Cycle with Stalls measurement. It indicates the peak efficiency starting from the beginning and updating every 1000 cycles.