Hands-on labs:
How to use Trace

Requirements

- Three labs demonstrate the capabilities of using multiple Trace features on C66x devices:
- **Lab 1:** PC Trace use cases and customization
- **Lab 2:** Hotspot analysis with Function, Stall and Cache profiler
- **Lab 3:** Getting started with non-intrusive system trace (STM), SoC profiling and customization

LAB conventions

Before starting, it is important to review some lab conventions that will ease your work…

- **Lab steps are in black and numbered for easier reference**
  1. …
  2. …

- **Explanations, notes, warnings are written in blue**
  - Warnings are shown with
  - Information is marked with
  - Tips and answers are marked with
  - Questions are marked with

Requirements

- Software:
  - Code Composer Studio v5.4.0.00091
    Download from http://processors.wiki.ti.com/index.php/Download_CCS
  - BIOS-MCSDK 02.01.02.06 + patch version 02.01.02.06P01
    Download from http://software-dl.ti.com/sdoemb/sdoemb_public_sw/bios_mcsdk/latest/index_FDS.html
  - UIA 1.01.01.14
    Download from http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/uia/index.html

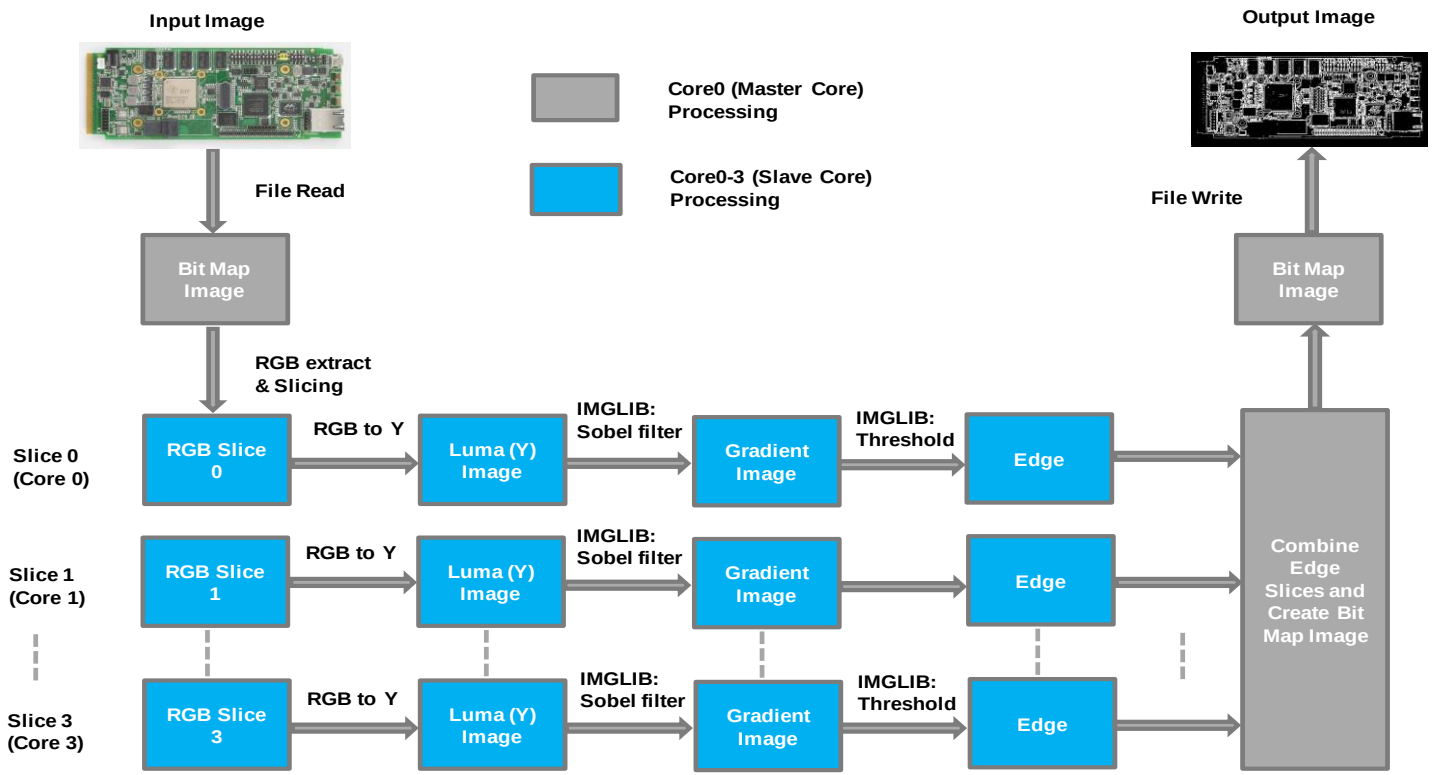
- Hardware
  - C6678 EVM (TMDXEVM6678L or LE, but this last one requires removing the built-in Mezzanine card)
  - Spectrum Digital XDS560v2 PRO TRACE Emulator.
    - XDS560v2 STM Emulators can be used for Labs 1 and 3
  - Ethernet cable connected between EVM and host

Installation and setup

1. Download and install the software into the directory C:\ti in the following order:
   - Code Composer Studio 5.4.0.00091
   - BIOS MCSDK v02.01.02.06
   - BIOS MCSDK Patch v02.01.02.06.patch01
   - Unzip the UIA v1.01.01.14 (there is no installer for this component)
2. Start Code Composer Studio
3. Select a workspace when requested
4. Wait until the *Add Discovered Products* window comes up
5. Leave all MCSDK related components selected
6. ***Unselect*** all NDK versions ***except*** **ndk_2_21_02_43**
7. Select *OK*
8. If warning pops-up, select *OK*
9. Say *Yes* to restarting CCS when requested
10. After CCS restarts, if requested to add other versions of NDK, select *Cancel*
11. Close *TI Resource Explorer* window which comes up by default

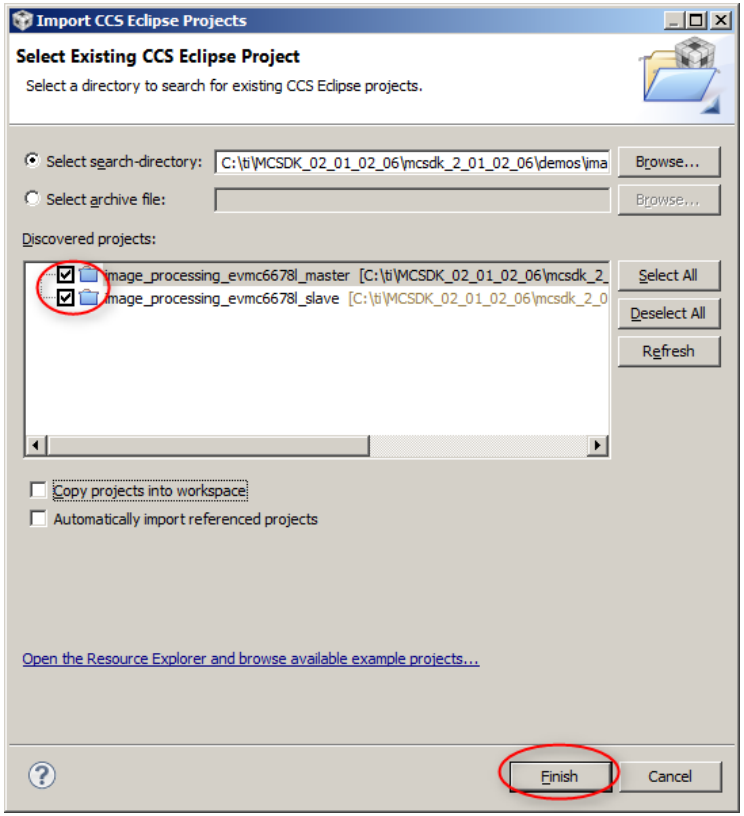
MCSDK Image processing demo

- The labs in this workshop use the Image Processing demo from MCSDK http://processors.wiki.ti.com/index.php/MCSDK_Image_Processing_Demonstration_Guide
- This application is an image processing system using a simple multicore framework. It receives an image via Ethernet, runs TI image processing routines (*imagelib*) on multiple cores to perform filtering and edge detection, and sends the image back via Ethernet.



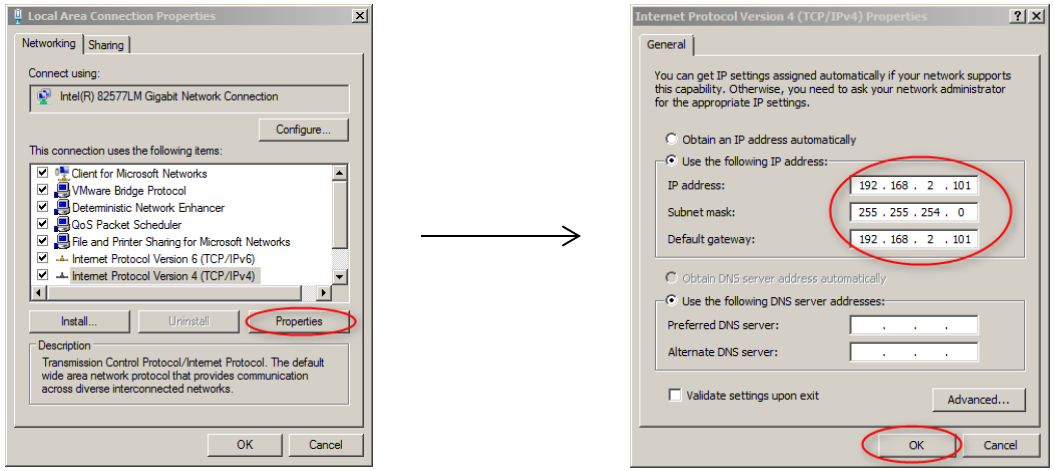
Importing and building the demo project

1. From CCS main menu, select *Project → Import Existing CCS Eclipse Project*
2. Browse to folder: C:\ti\mcsdk_2_01_02_06\demos\image_processing\ipc\evmc6678l
3. Select *OK*
4. Select the following projects:
   i. **image_processing_evmc6678l_slave**
   ii. **image_processing_evmc6678l_master**
5. Make sure the checkbox *Copy projects into workspace* is not enabled
6. Select *Finish*
7. To select the correct UIA version, right-click on the project name, then select *Properties.* In the left menu, select *General* and in the right box select the tab *RTSC.* Scroll down until you see *System Analyzer (UIA Target)* and select version **1.1.1.14**. Click *OK*. Repeat for the other project.
8. In *Project Explorer* view, right-click on each project imported and select *Build Project*



Running the demo (i)

1. Make sure the DIP switch settings in the C6678 EVM are the same as shown below:
   - SW3[1:4]: OFF, ON, ON, ON SW4[1:4]: ON, ON, ON, ON
   - SW5[1:4]: ON, ON, ON, ON SW6[1:4]: ON, ON, ON, ON SW9[1:2]: ON, OFF
2. Connect the C6678 EVM to the host PC using the Ethernet cable
3. Connect XDS560v2 PRO TRACE Emulator to the EVM and PC (via USB). Power up the EVM and the Emulator (leave around 1-½ minute for the Emulator to boot)
4. Change the host PC network settings to use static IP address **192.168.2.101** and Subnet mask **255.255.254.0**
   - Go to *Control Panel* → *Network and Sharing Center* → *Change Adapter Settings*
   - Right click on *Local Area Network*, highlight *Internet Protocol Version 4* and change the *Properties* as shown below:

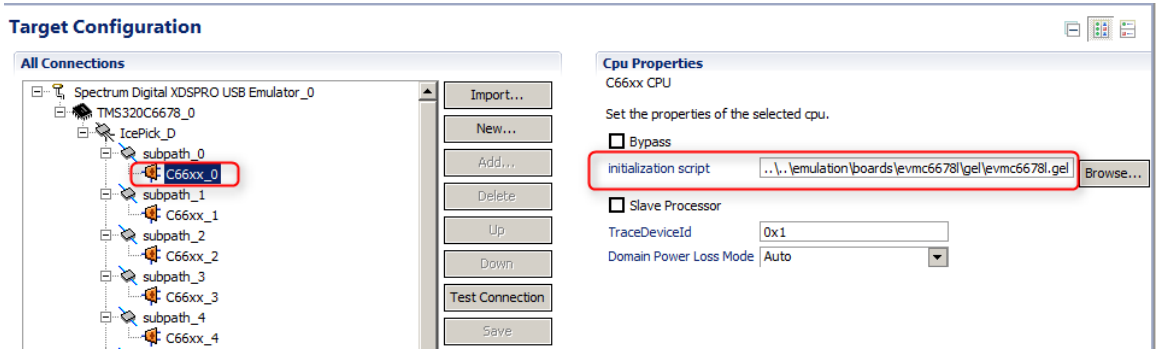


If you prefer to use DHCP instead of static IP, SW9 position2 on the EVM should be ON, and the IP address used by the demo is shown in CCS console after the program is run

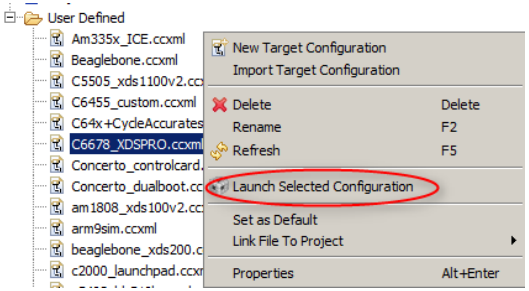
Running the demo (ii)

In order to connect to the board, a Target Configuration File must be created. This file contains the information about the JTAG emulator and the device to be connected.

1. Go to *File* → *New* → *Target Configuration File*
2. Type file name as *C6678_XDSPRO* and click *Finish*
3. For *Connection* select *Spectrum Digital XDSPRO USB Emulator*
4. Type *C6678* in the *Board or Device* filter field and select *TMS320C6678*
5. Click on *Advanced* tab at the bottom of the view
6. Select **C66xx_0** and in the *initialization script* box click on browse to add the file <evm6678l.gel> located at: C:\ti\ccsv5\ccs_base\emulation\boards\evmc6678l\gel

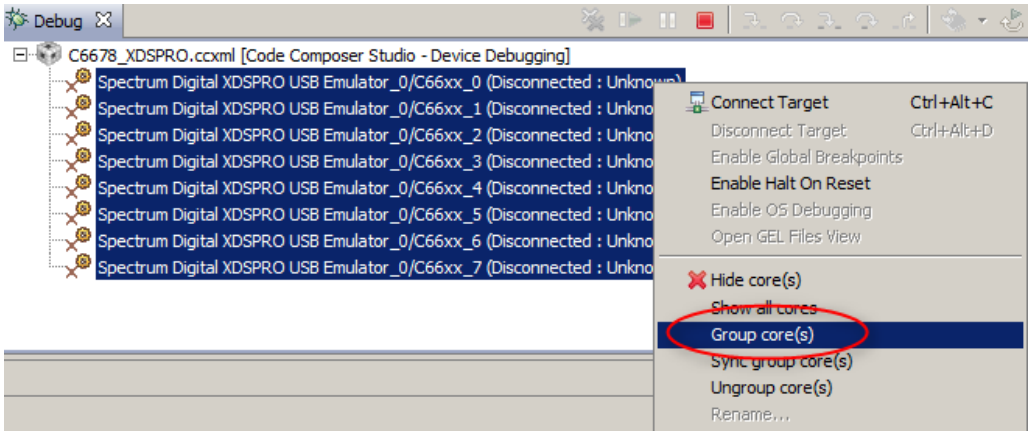


7. Click on *Save*
8. Go to menu *View* → *Target Configurations*
9. Under *User Defined,* select the configuration just created
10. Launch the debug session by right-clicking on the file and choosing *Launch Selected Configuration*

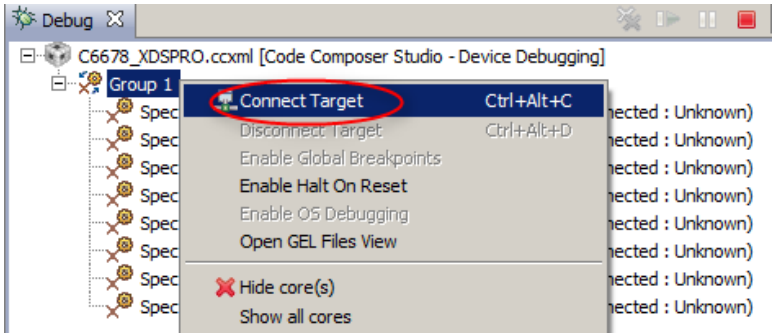


Running the demo (iii)

1. In the *Debug* view, group all the 8 C66x cores into one single group
   - Select all cores in Debug view, right-click and choose *Group Core(s)*



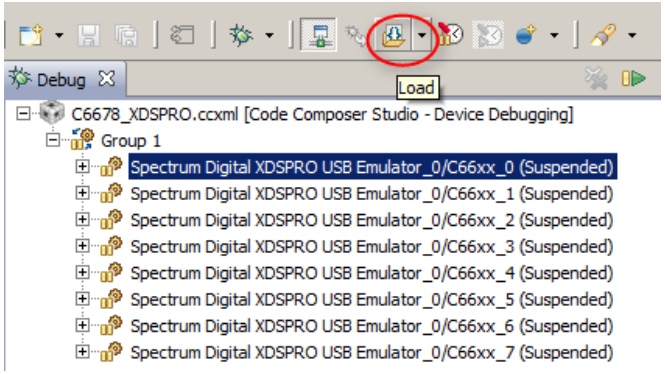
2. Connect to the cores
   - Select Group1, right-click and choose *Connect Target*



Running the demo (iv)

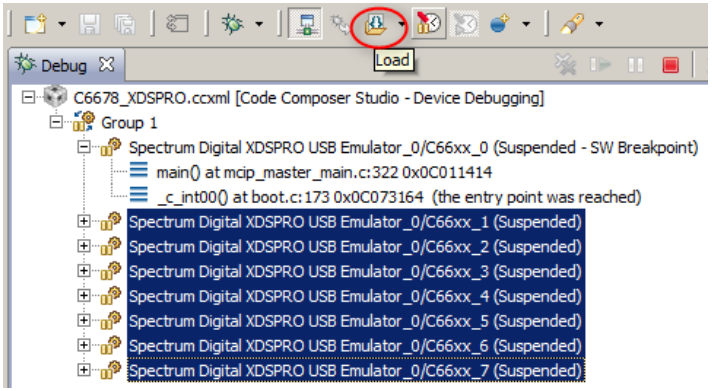
1. Load **image_processing_evmc6678l_master.out** on core0

   - Select **C66xx_0**, click on *Load Program* icon, select *Browse Project* and select the *Debug* build of the master project: (image_processing_evm6678l_master/Debug)



2. Load **image_processing_evmc6678l_slave.out** on cores1 through 7

   - Select **C66xx_1** through **C66xx_7** using the shift + left mouse button, click on *Load Program icon,* select *Browse Project* and select the *Debug* build of the slave project: (image_processing_evm6678l_slave/Debug)

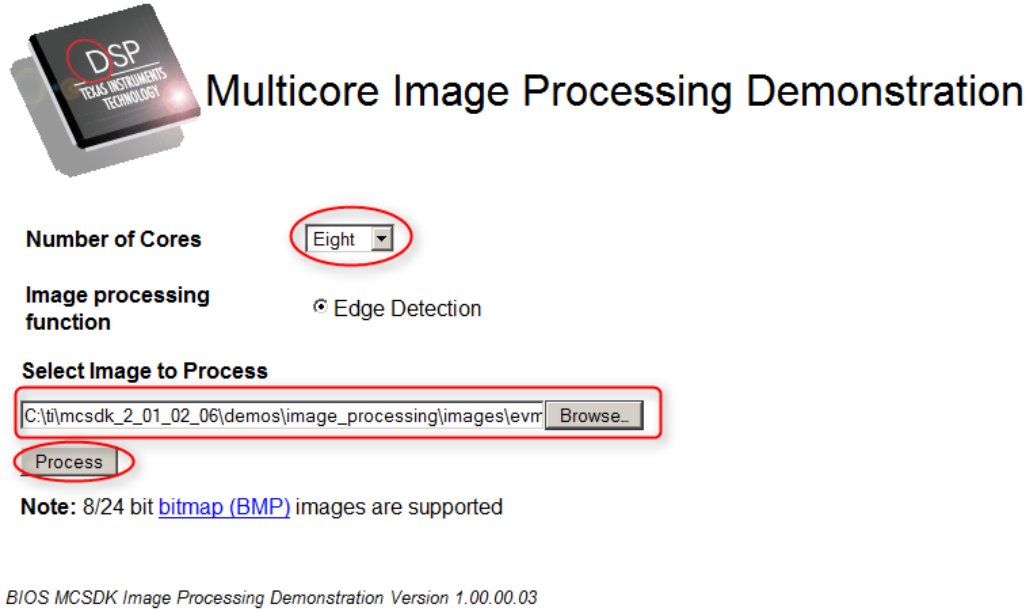


3. Run all cores

   - Select *Group 1* and click on *Resume* icon

Running the demo (v)

1. Open a web browser and type in **192.168.2.100** (EVM's IP address) in the address box
   If the C6678 EVM is configured to use DHCP, use the IP address printed out in the CCS console view when program is run
2. The "Multicore Image Processing Demonstration" webpage will be shown
3. In the drop-down option *Number of Cores,* set it as *Eight* to use all DSP cores
4. At the box *Select Image to Process*, load the file <evmc6678l_1920x1080_5_93MB.bmp> located at C:\ti\mcsdk_2_01_02_06\demos\image_processing\images
5. Click on the button *Process*



Running the demo (vi)

The output of the image processing demo includes several details of the image and the processing time, as well as a copy of the input image and the processed output with filtering and edge detection (scroll down on the page to see all details).

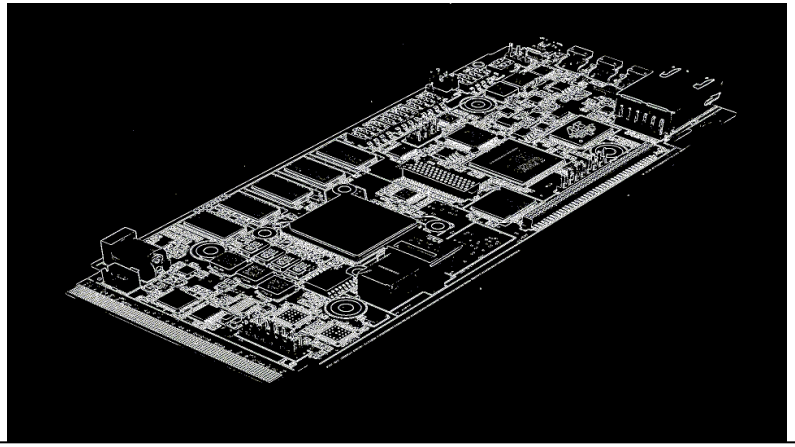
Multicore Image Processing Demonstration - Output

Return to Main Page

Image Processing Function	Edge Detection
Image Dimension (in pixels)	1920x1080
Input Image Size (in bytes)	6220854
Number of Cores Used	8
Processing Time	26.900ms

Input Image

Output Image



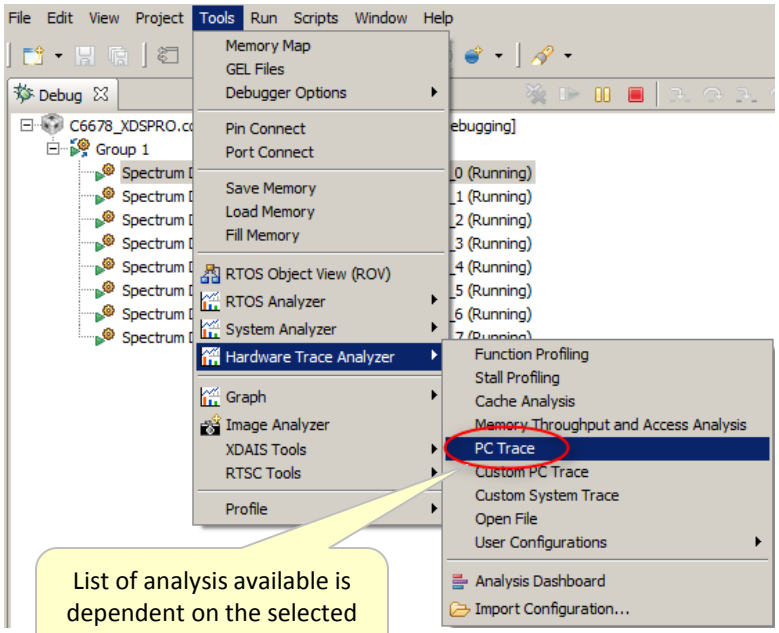
LAB 1:
PC Trace Use Cases and customization

Lab 1: Exercise summary

- **Key Objectives**
  - Collect and analyze program execution using PC Trace
  - Perform analysis of collected data using Function execution graph and Program address graph
  - Customize trace configuration
  - Save configuration and export/import saved configurations
- **Tools and Concepts Covered**
  - Trace configuration
  - Trace Viewer
  - Analysis from within Trace Viewer
  - Navigation within analysis views
  - Export/import of trace configurations

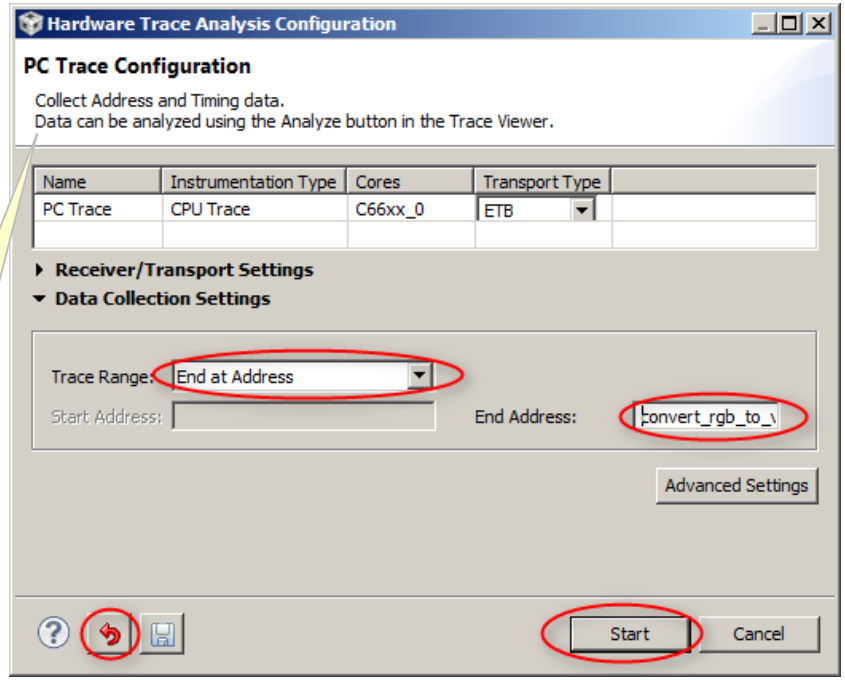
Trace program execution (i)

1. If Image Processing Demo is not already running, then complete Steps i to iv of Running the demo
2. Select **C66xx_0** in the *Debug* view
3. In CCS menu select *Tools* → *Hardware Trace Analyzer* → *PC Trace* to start PC Trace
4. Click  to clear any saved/cached settings from previous run
5. Set *Trace Range* = *End at Address*
6. Set *End Address* = *convert_rgb_to_y*
7. Select *Start* to open *Trace Viewer*



List of analysis available is dependent on the selected core(s) and their state

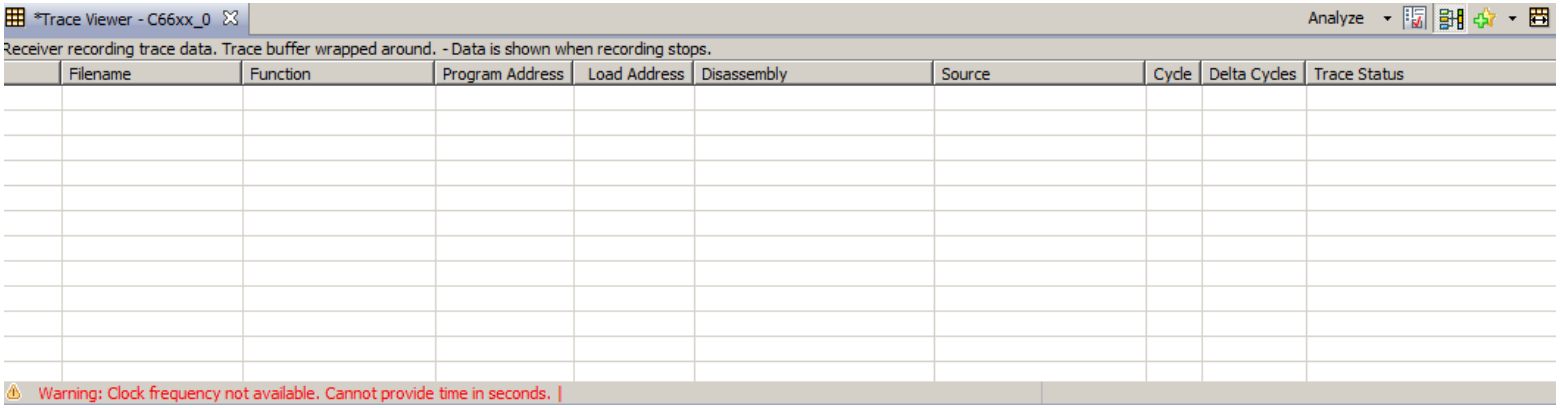
Description of default configuration



Trace program execution (ii)

Trace Viewer view opens and status shows that buffer is already wrapped but data will only be shown when recording ends

In this case recording will end either when **convert_rgb_to_y()** is executed or **C66xx_0** is halted



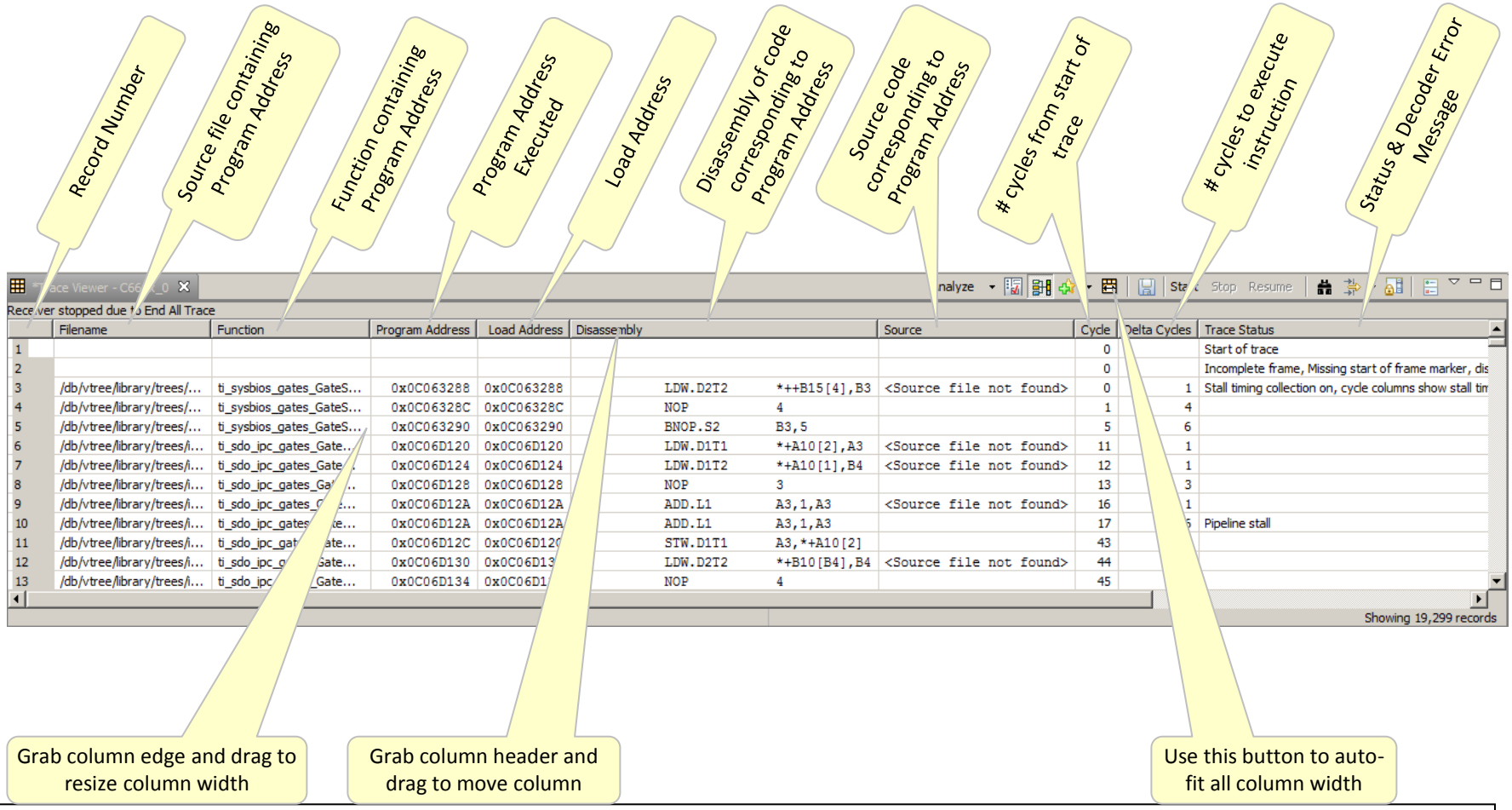
There is a warning about clock frequency. The warning can be ignored for now. Clock frequency will be obtained when data collection stops

8. Complete step v of Running the demo to run the process image function

9. Wait for *Trace Viewer* view to show all collected data

Analyze trace results

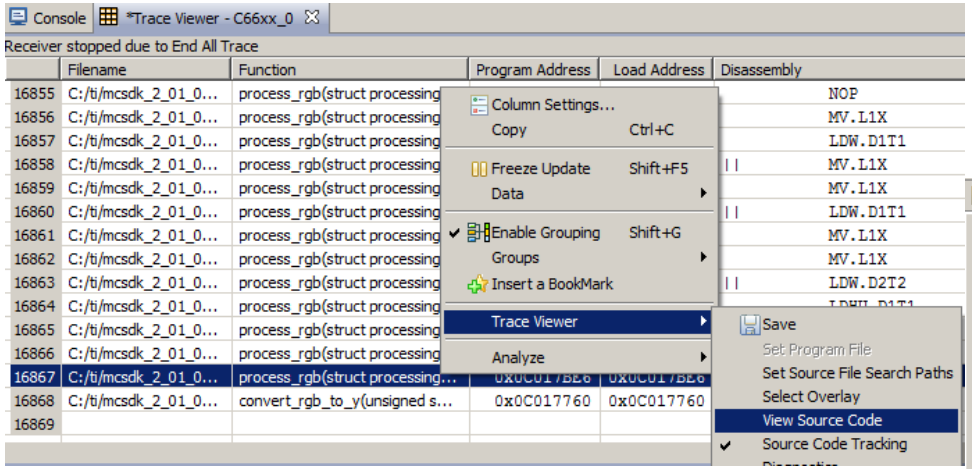
1. Grab and drag column borders to resize as needed
2. Grab and drag column headers to reposition columns as required
3. *Trace Viewer* shows Program Addresses executed leading up to **convert_rgb_to_y()**

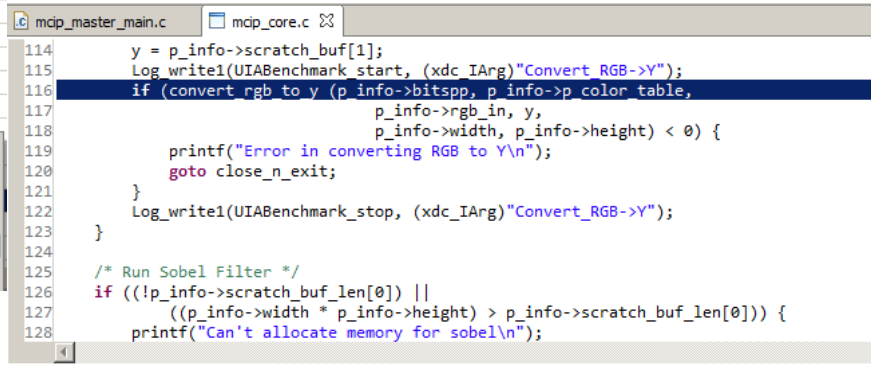


	Filename	Function	Program Address	Load Address	Disassembly		Source	Cycle	Delta Cycles	Trace Status
1								0		Start of trace
2								0		Incomplete frame, Missing start of frame marker, dis
3	/db/vtree/library/trees/...	ti_sysbios_gates_GateS...	0x0C063288	0x0C063288	LDW.D2T2	*++B15[4],B3	<Source file not found>	0	1	Stall timing collection on, cycle columns show stall tim
4	/db/vtree/library/trees/...	ti_sysbios_gates_GateS...	0x0C06328C	0x0C06328C	NOP	4		1	4	
5	/db/vtree/library/trees/...	ti_sysbios_gates_GateS...	0x0C063290	0x0C063290	BNOP.S2	B3,5		5	6	
6	/db/vtree/library/trees/i...	ti_sdo_ipc_gates_Gate...	0x0C06D120	0x0C06D120	LDW.D1T1	*+A10[2],A3	<Source file not found>	11	1	
7	/db/vtree/library/trees/i...	ti_sdo_ipc_gates_Gate...	0x0C06D124	0x0C06D124	LDW.D1T2	*+A10[1],B4	<Source file not found>	12	1	
8	/db/vtree/library/trees/i...	ti_sdo_ipc_gates_Ga...	0x0C06D128	0x0C06D128	NOP	3		13	3	
9	/db/vtree/library/trees/i...	ti_sdo_ipc_gates_G...te...	0x0C06D12A	0x0C06D12A	ADD.L1	A3,1,A3	<Source file not found>	16	1	
10	/db/vtree/library/trees/i...	ti_sdo_ipc_gates...te...	0x0C06D12A	0x0C06D12A	ADD.L1	A3,1,A3		17	5	Pipeline stall
11	/db/vtree/library/trees/i...	ti_sdo_ipc_gat...ate...	0x0C06D12C	0x0C06D12C	STW.D1T1	A3,*+A10[2]		43		
12	/db/vtree/library/trees/i...	ti_sdo_ipc_g...Gate...	0x0C06D130	0x0C06D13	LDW.D2T2	*+B10[B4],B4	<Source file not found>	44		
13	/db/vtree/library/trees/i...	ti_sdo_ipc...Gate...	0x0C06D134	0x0C06D1	NOP	4		45		

View source code

1. In *Trace Viewer* view, click on the record before **convert_rgb_to_y** (second to last)
2. Right-click-context-menu select *Trace Viewer* → *View Source Code*
3. The file <mcip_core.c> will open at line 116 showing source code corresponding to the program address in the selected record

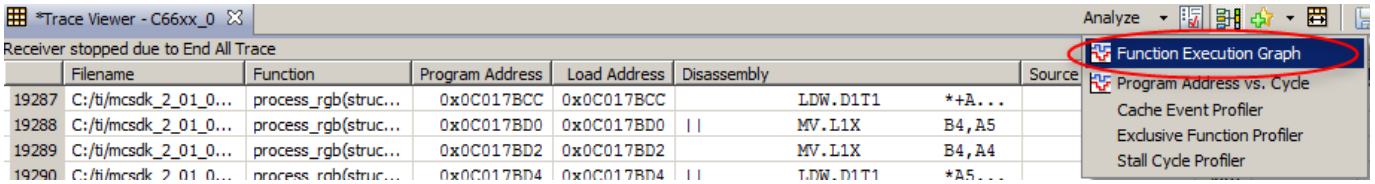




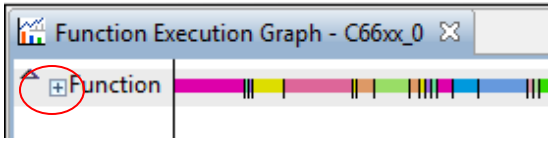
4. Scroll down in *Trace Viewer* to the record containing **convert_rgb_to_y** and notice the function **convert_rgb_to_y()** is highlighted in the source file

Function execution graph (i)

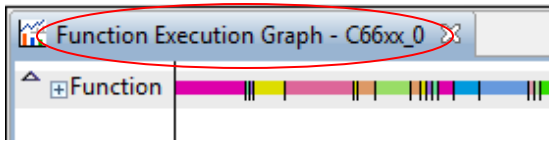
1. From *Trace Viewer* toolbar, select *Analyze* → *Function Execution Graph*



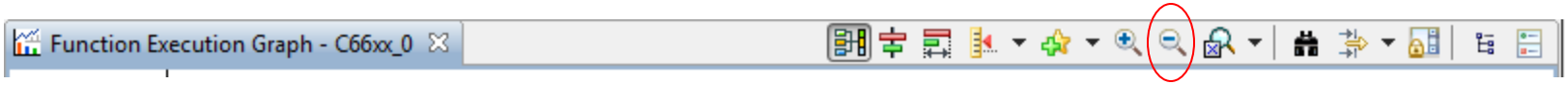
2. Click on the **+** next to *Function* on y-axis to expand graph



3. Double-click on view title to expand the view to full-screen

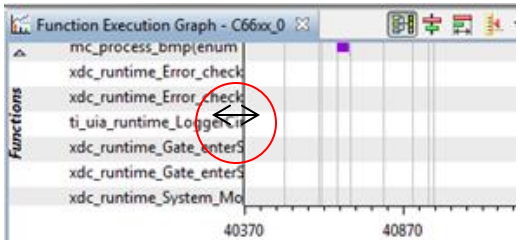


4. Click multiple times on the Zoom out button in the graph toolbar to see entire execution

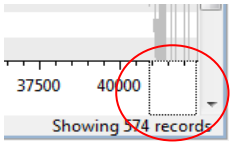


Function execution graph (ii)

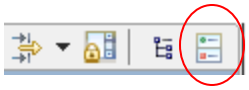
5. Grab y-axis with mouse and drag to see more of the name of the functions



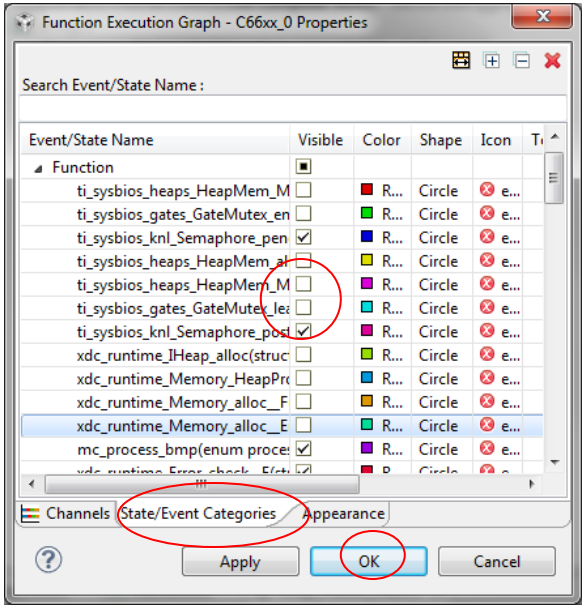
6. Place mouse just below the x-axis and expand that last bit of the graph (Alt + left mouse key) to zoom into selected region



7. (Optional) From *Function Execution Graph* toolbar select the Display Properties icon

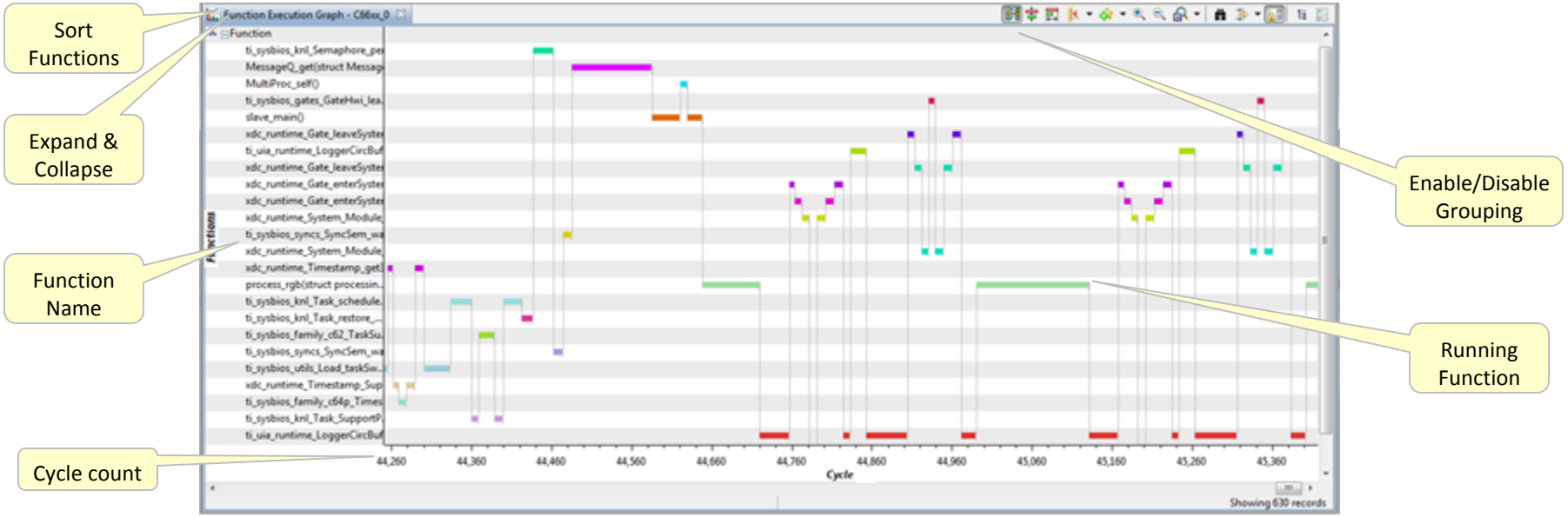


8. (Optional) In the properties view, click on *State/Event Categories* tab, expand *Function* and uncheck the Visible box for functions that are not of interest. Click *OK* when finished.



This will fit more of the graph in view

Function execution graph (iii)



The graph shows what function is executing and not function entry/exit

At the end of the graph **process_rgb()** makes some UIA log calls and then **convert_rgb_to_y()**

9. Double click on view title to collapse full-screen view
10. Click anywhere in graph to automatically scroll *Trace Viewer* to same *cycle* position
11. Click  on Graph toolbar to disable grouping
12. Now click anywhere in graph and note that *Trace Viewer* is no longer scrolled

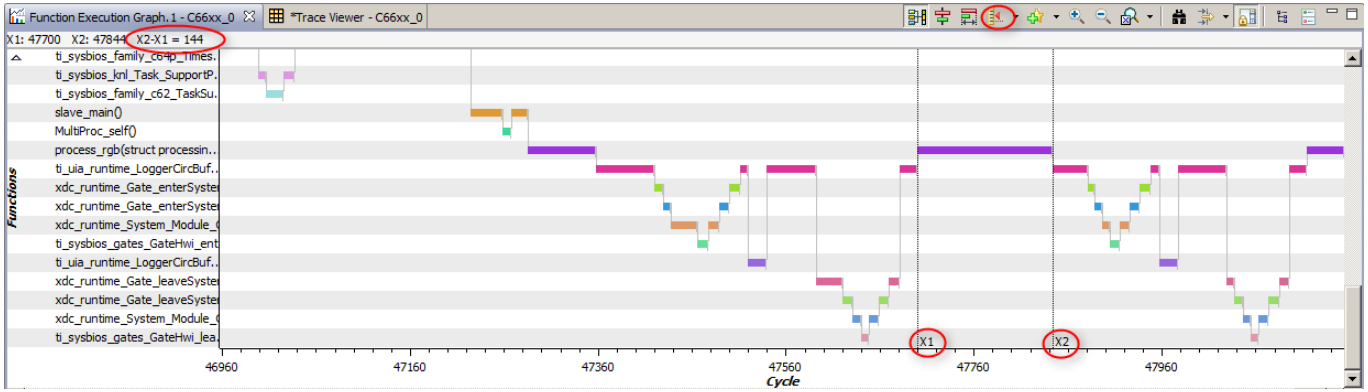
Function execution graph (iv)

14. Click on  in *Function Execution Graph* toolbar then click at the beginning of an instance of **process_rgb()** in the graph. This inserts a measurement marker 1 (X1)

15. Repeat step 1 but this time click at the end of same instance of **process_rgb()**. This inserts measurement marker 2 (X2)

16. Look at top left corner of graph to see number of cycle between X1 and X2

17. While holding Shift button, use mouse to select and drag X2. Notice change in the number at top left corner of graph



18. Double-click on X2 to remove

19. From context menu select *Remove All Measurement Marks* to remove remaining markers (in this case only X1)

Program address graph

1. Go to *Trace Viewer* view
2. From *Trace Viewer* toolbar, select *Analyze* → *Program Address vs. Cycle*
3. Click on the graph zoom out button ( ) multiple times to see entire range of program addresses executed

4. (Optional) Using Alt + left mouse button, zoom into a selected region

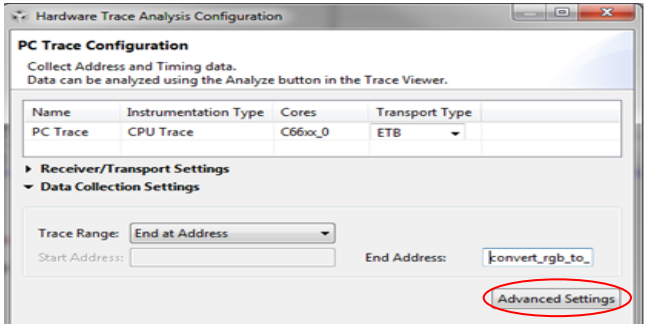
5. (Optional) Use the zoom reset button ( ) on graph toolbar to restore original zoom

Customize for data tracing

Data Tracing shows the actual data being transferred to/from the CPU

1. In *Trace Viewer*, click on the *Analysis Properties* icon to change its settings

2. In the *Hardware Trace Analysis Configuration* dialog, select *Advanced Settings*

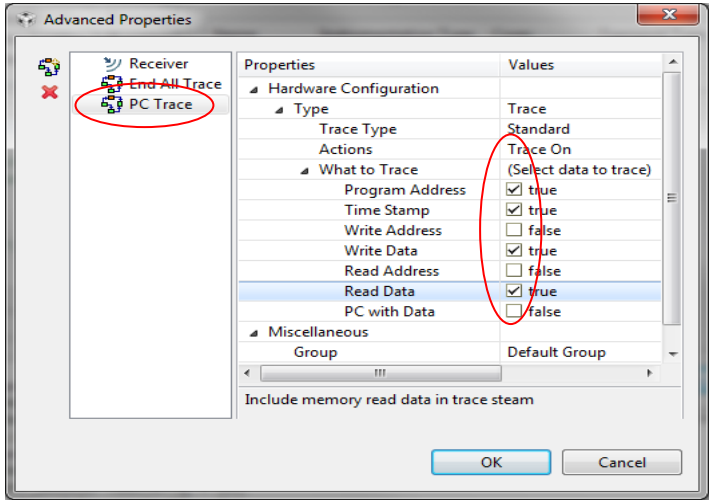


3. Select *PC Trace* in the left column

4. Expand *Type* and *What to Trace* in the right pane

5. Enable tracing of *Write Data* and *Read Data* by enabling their check boxes

This would add additional data tracing options to the program trace

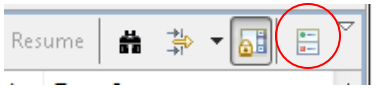


6. Select *OK*

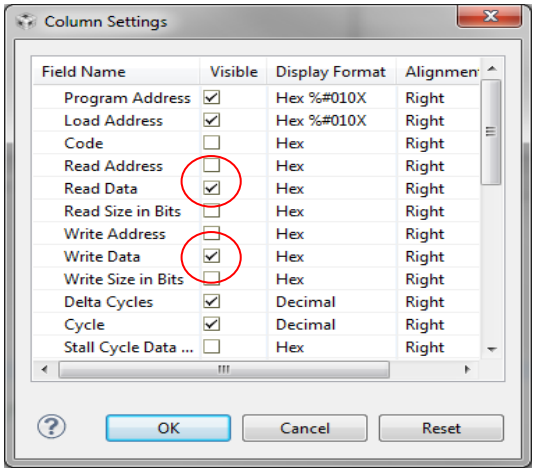
7. Select *Apply* in *Hardware Trace Analysis Configuration*

Trace data access

8. Re-run the demo by completing step v of Running the demo to process image
9. Wait for *Trace Viewer* to update with collected data
10. Note that *Trace Viewer* does not show *Data Read* and *Data Write* columns by default
11. In *Trace Viewer* toolbar click on *Column Settings* button



12. In *Column Settings* dialog, enable visibility of *Read Data* and *Write Data*



13. Select *OK*
14. In *Trace Viewer* resize and reposition columns as required
15. Scroll through *Trace Viewer* to see what data was read/written

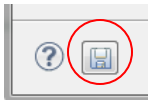
Saving the configuration

1. Click on *Analysis Properties* button in *Trace Viewer* toolbar

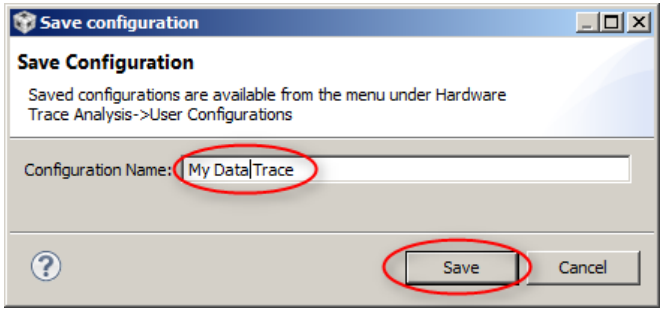
   This reopens the configuration dialog. Here properties can be modified and re-applied to the analysis

   Now we will not modify properties, instead we'll save current configuration for future reuse

2. Press the *Save* button at the bottom of the configuration dialog

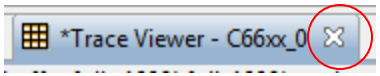


3. In *Save Configuration* dialog enter *My Data Trace* for *Configuration Name* then press *Save*

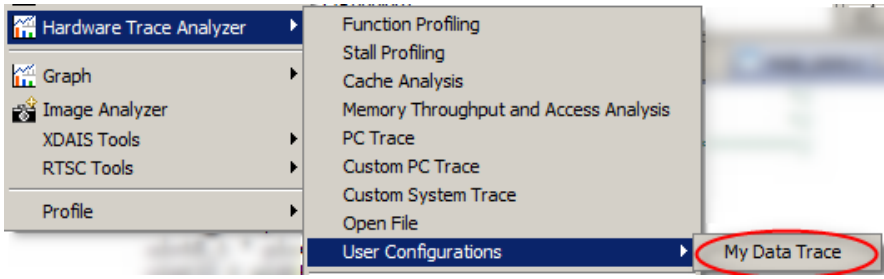


4. Press *Cancel* to exit *Hardware Trace Analysis Configuration* dialog

5. Close the *Trace Viewer*

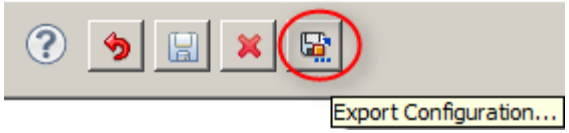


6. Go to *Tools->Hardware Trace Analysis->User Configurations* and note that *My Data Trace* is now available for reuse



Sharing the configuration

1. Create a C:\temp directory on your hard disk
2. Select *Tools* → *Hardware Trace Analysis* → *User Configurations* → *My Data Trace*
3. Click *Export Configuration* button at bottom of configuration dialog



4. Browse to the directory C:\temp, select *Save*
5. Click on *Delete* button at bottom of configuration dialog to delete this saved analysis

6. Go to *Tools* → *Hardware Trace Analyzer*. Note that Saved Analysis no longer exists

⚠ It seems CCSv5.4 has a bug, as the saved analysis is still visible

7. Select *Tools* → *Hardware Trace Analyzer* → *Import Configuration*
8. Browse to the directory C:\temp and select *File Name* as <*My Data Trace.zip*>, click *Open*
9. Go to *Tools* → *Hardware Trace Analyzer* → *User Configurations* and see that an entry named *My Data Trace* now exists

Lab 1: Summary

- Usage of DSP trace for real-time tracing of program execution
- Trace Viewer allows for further analysis to process collected data
- Usage of Function Execution Graph provides a bird's eye view of program execution
- Program Address Graph shows what program addresses are executed
- Views have numerous features to help navigate the large volume of data that may be collected
- Trace can be used to monitor what data addresses and values are accessed
- Trace configurations can be customized using Advanced Settings
- Configurations can be saved for reuse
- Saved configurations can be exported/imported

LAB 2:
Hotspot Analysis with Function, Stall and Cache Profiler

Lab 2: Exercise summary

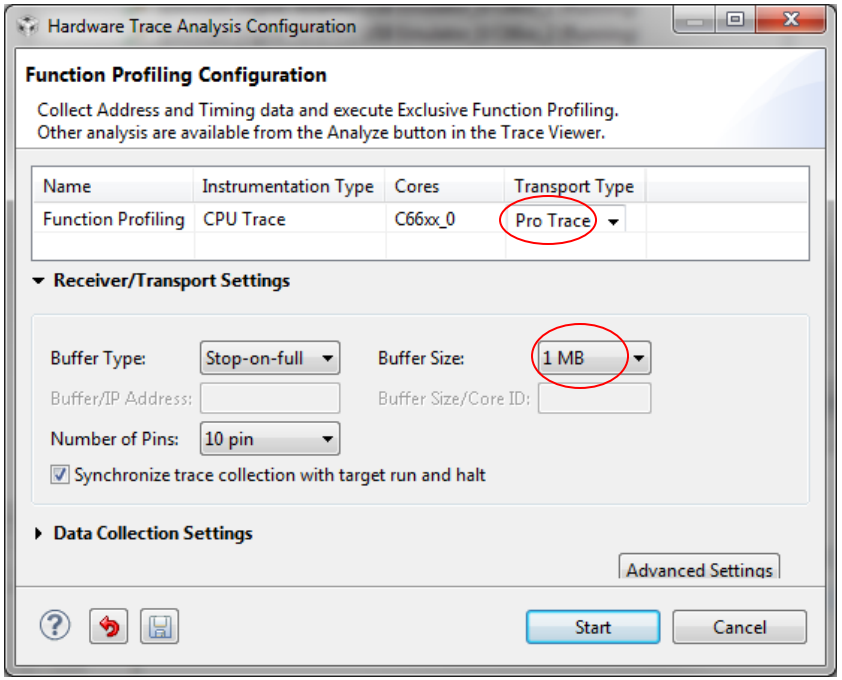
- **Key Objectives**
  - Run Function Profiler to analyze which functions take up most time
  - Run Stall Profiler to analyze where stalls are occurring
  - Run and analyze cache analysis results
  - Export and Import analysis data

- **Tools and Concepts Covered**
  - Function Profiler
  - Stall Profiler
  - Cache Analysis
  - Analysis from within Trace Viewer
  - Navigation within analysis views
  - Using files to analyze data

Running function profiler (i)

The function profiler allows inspecting the time spent by every function in the system

1. If Image Processing Demo is not already running, complete steps i to iv of Running the demo
2. Select **C66xx_0** in *Debug* view
3. Open Function Profiler from CCS menu *Tools* → *Hardware Trace Analyzer* → *Function Profiling*
4. Click to reset to original settings
5. Change *Transport Type* to *Pro Trace* and *Buffer Size* to *1MB*



Running function profiler (ii)

6. Click *Data Collection Settings* to expand

7. For *Trace Range*, select *Start and Stop at Addresses*

Stop Address will not end trace, it will just stop collection until *Start* is encountered again

8. Set *Start Address* = *IMG_sobel_3x3_8* and *End Address* = *MultiProc_self*

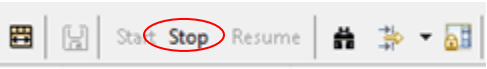
9. Select *Start*

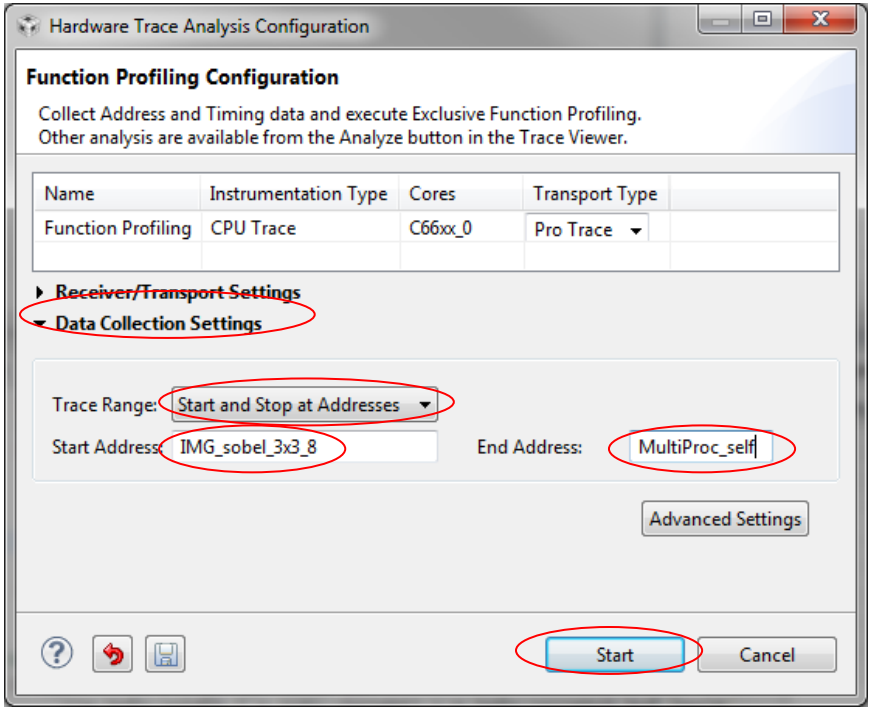
10. Complete step v of Running the demo

11. Wait for Demo to complete

12. Go to the *Trace Viewer* view

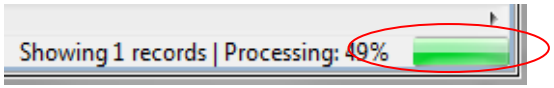
13. Press *Stop* in *Trace Viewer* toolbar





Analyzing function profile results (i)

1. Wait for *Trace Viewer* and *Exclusive Function Profiler* processing to complete



2. Go to *Exclusive Function Profiler* view
3. Resize column width of *Exclusive Function Profiler* view as needed
4. Click on *CPU Cycles Total* column header twice to sort data in descending order
5. Note that assembly routines are shown as:
   unknown_<address of next function>_<address of previous function – 1>_<name of next function>
6. Note 2 functions **IMG_thr_le2min_8()** and **IMG_sobel_3x3_8()** are taking ~99% of time

Exclusive Function Profiler - C66xx_0 | *Trace Viewer - C66xx_0

	Function	Calls	CPU Cycles Min	CPU Cycles Max	CPU Cycles Total	CPU Cycles Avg	CPU Cycles Percent
1	unknown_0xc06fca0_0xc06fd5f_IMG_thr_le2min_8()	1			1,297,028	1,297,028.00	66.64
2	unknown_0xc04cb80_0xc04cddf_IMG_sobel_3x3_8()	1	2975	2975	642,689	642,689.00	33.02
3	process_rgb(struct processing_info *)	1			1,873	1,873.00	0.10
4	ti_sysbios_knl_Clock_workFunc__E(unsigned int, u...	2	207	207	1,073	536.50	0.06
5	unknown_0xc0e0dc0_0xc0e0ddf_ti_sysbios_family...	2	24	24	728	364.00	0.04
6	ti_uia_runtime_LoggerCircBuf_write1__E(struct ti_...	4	117	201	616	154.00	0.03
7	ti_sysbios_family_c64p_Hwi_dispatchC__I(int)	2	269	269	535	267.50	0.03
8	ti_sysbios_knl_Swi_restoreHwi__E(unsigned int)	2	150	150	280	140.00	0.01
9	ti_sysbios_knl_Swi_post__E(struct ti_sysbios_knl_...	2	130	130	260	130.00	0.01
10	ti_sysbios_family_c64p_Hwi_dispatchAlways()	2	96	96	214	107.00	0.01
11	ti_sysbios_knl_Clock_doTick__I(unsigned int)	2	97	97	194	97.00	0.01
12	ti_sysbios_knl_Clock_logTick__E()	2	73	73	146	73.00	0.01
13	xdc_runtime_Gate_enterSystem__F()	4	20	44	126	31.50	0.01
14	xdc_runtime_System_Module_GateProxy_leave__...	4	20	20	80	20.00	0.00

Analyzing function profile results (ii)

6. Scroll to right on *Exclusive Function Profiler* Table

7. Observe that ~96% (1.2M cycles) of **IMG_thr_le2min_8()** time was a result of pipeline stalls

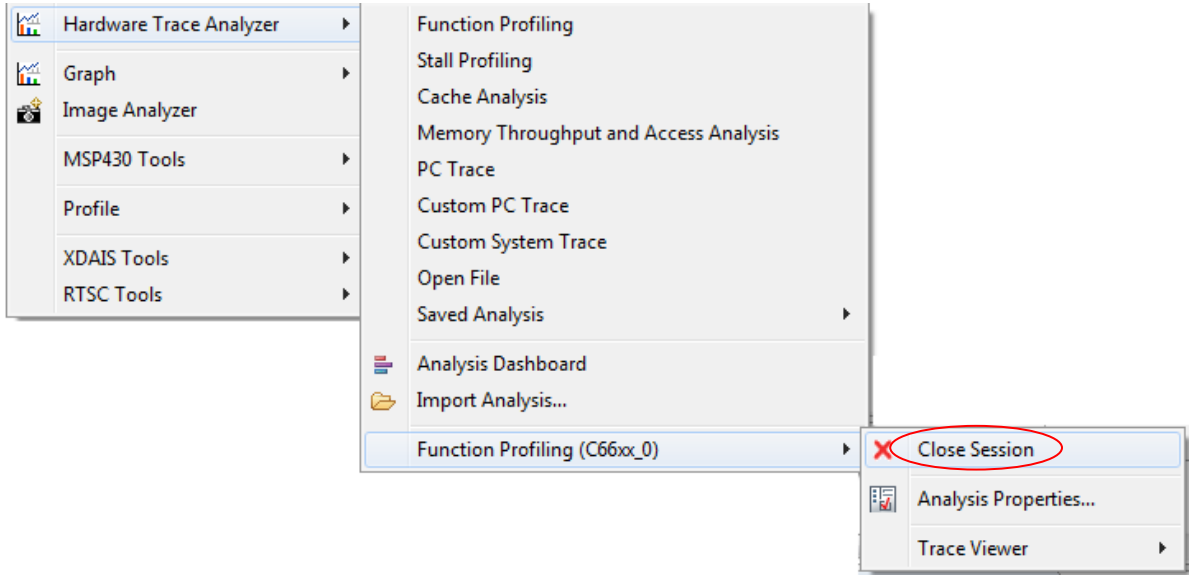
Exclusive Function Profiler - C66xx_0 | *Trace Viewer - C66xx_0

	Calls	CPU Cycles Min	CPU Cycles Max	CPU C...	CPU Cycles Avg	CPU Cycles...	Pipeline ...	Pipeline St...	Pipeline Stalls Total	Pipeline Stalls Avg	Percent of Total Pipeline Stalls	Function Pipeline Stalls Percent
6fd5f_IMG_thr_le2min_8()	1			1,297,028	1,297,028.00	66.64			1,247,677	1,247,677.00	82.87	96.20
14cddf_IMG_sobel_3x3_8()	1	2975	2975	642,689	642,689.00	33.02	1731	1731	253,831	253,831.00	16.86	39.50
sing_info *)	1			1,873	1,873.00	0.10			1,564	1,564.00	0.10	83.50
Func__E(unsigned int, unsi...	2	207	207	1,073	536.50	0.06	60	60	779	389.50	0.05	72.60
le0ddf_ti_sysbios_family_c...	2	24	24	728	364.00	0.04	13	13	706	353.00	0.05	96.98
Buf_write1__E(struct ti_uia...	4	117	201	616	154.00	0.03	0	84	148	37.00	0.01	24.03
ti_dispatchC__I(int)	2	269	269	535	267.50	0.03	139	139	275	137.50	0.02	51.40
Hwi__E(unsigned int)	2	150	150	280	140.00	0.01	27	27	34	17.00	0.00	12.14
E(struct ti_sysbios_knl_Swi...	2	130	130	260	130.00	0.01	73	73	146	73.00	0.01	56.15

Running stall profiler (i)

The stall profiler allows analyzing the time the system spends waiting for resources (peripherals) and memory

1. Close the current running Function Profiler by selecting menu *Tools → Hardware Trace Analyzer → Function Profiling (C66xx_0) → Close Session*

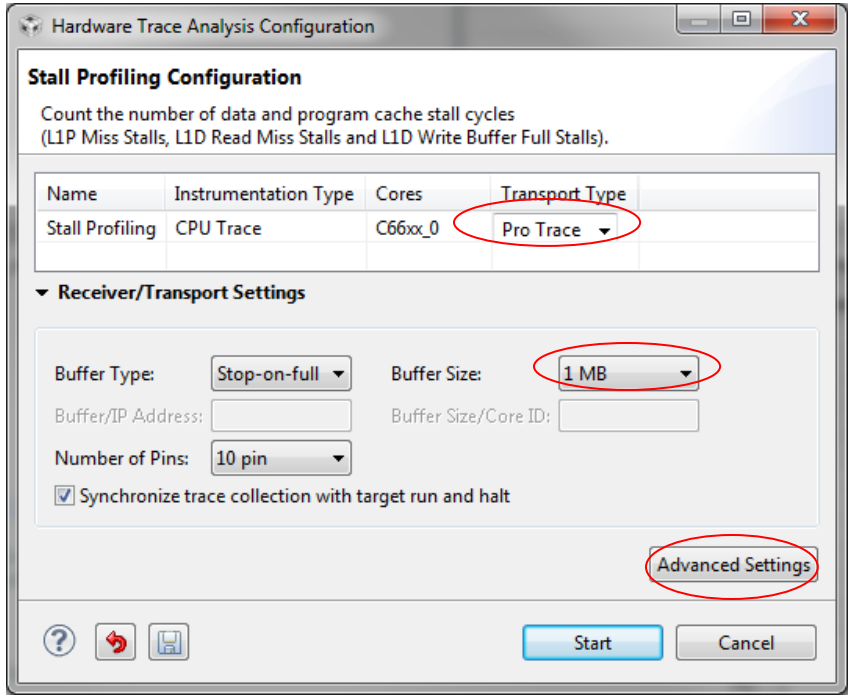


2. Open Stall Profiler from menu *Tools → Hardware Trace Analyzer → Stall Profiling*

Running stall profiler (ii)

3. Click  to reset to original settings

4. Change *Transport Type* to *Pro Trace* and *Buffer Size* to *1MB*



5. Click on *Advanced Settings* to setup Start/Stop condition

Start/Stop support will be added to the configuration dialog in the next CCS release

Running stall profiler (iii)

6. Select *Pipeline Stall Analysis* trigger in left pane of *Advanced Properties* dialog

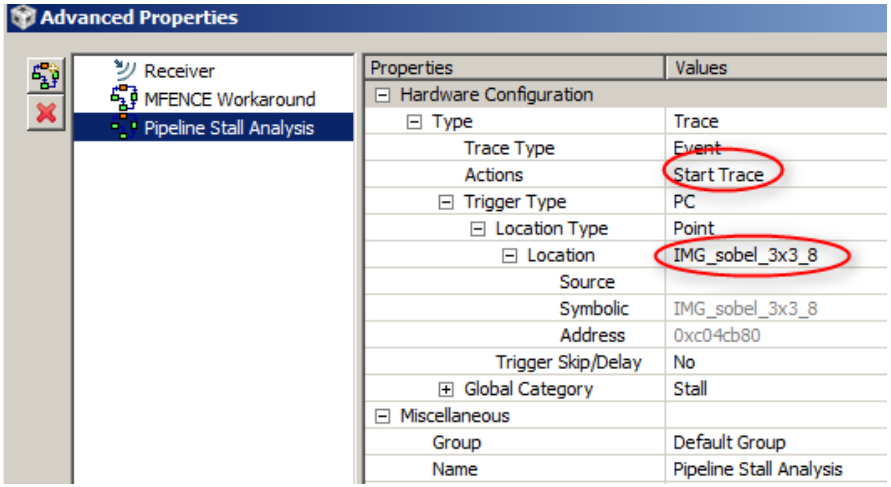
7. In the right pane:
   - expand *Type* in left column and change *Actions* to *Start Trace*
   - expand *Trigger Type* → *Location Type* → *Location* in left column and *change Location* to *IMG_sobel_3x3_8*

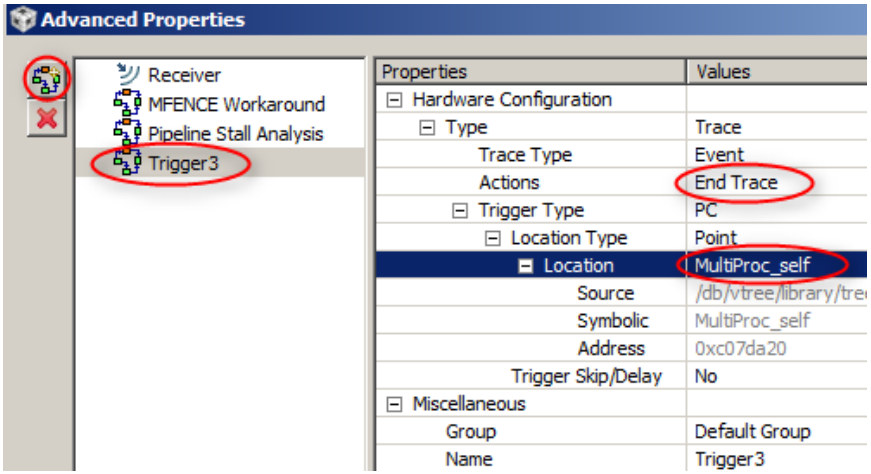
8. Expand *Global Category* to see what events are collected by default

9. Click on in the left pane to add another trigger (default name can be changed)

10. Select *Trigger3* in the left column and change *Actions* to *End Trace* and *Location* to *MultiProc_self*.

11. Click somewhere else in property view to allow symbol to be evaluated. Click *OK*.





Running stall profiler (iv)

12. Select *Start* in *Hardware Trace Analysis Configuration* dialog

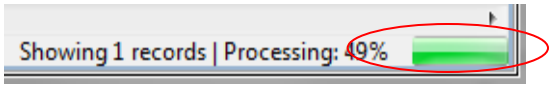
13. Complete step v of Running the demo

14. Wait for Demo to complete

15. Go to *Trace Viewer* view

16. Press *Stop* in *Trace Viewer* toolbar

17. Wait for *Trace Viewer* and *Stall Cycle Profiler* processing to complete



18. Go to *Stall Cycle Profiler* view

Analyzing stall profiler results

1. Observe that *Stall Cycle Data 2 Total* shows ~1.46M stall cycles for **IMG_thr_le2min_8()**

Stall Cycle Profiler - C66xx_0 | *Trace Viewer - C66xx_0

	Function	Stall Cycle Data 1 Total	Stall Cycle Data 2 Total	Stall Cycle Data 3 Total	Stall Cycle Data 4 Total
1	unknown_0xc06fca0_0xc06fd5f_IMG_thr_le2min_8()	168	1459249	0	0
2	unknown_0xc04cb80_0xc04cddf_IMG_sobel_3x3_8()	24	235873	0	31
3	ti_sysbios_knl_Clock_workFunc__E(unsigned int, unsi...	21	1640	0	9
4	process_rgb(struct processing_info *)	246	945	0	1
5	ti_sysbios_family_c64p_Hwi_dispatchC__I(int)	60	410	0	18
6	ti_sysbios_knl_Swi_post__E(struct ti_sysbios_knl_Swi...	18	198	0	3
7	ti_sysbios_family_c64p_Hwi_dispatchAlways()	48	114	22	16

2. By expanding the events to see what is collected by default for Event 2, it can be observed that the stalls are a result of L1D Read Misses

Advanced Properties

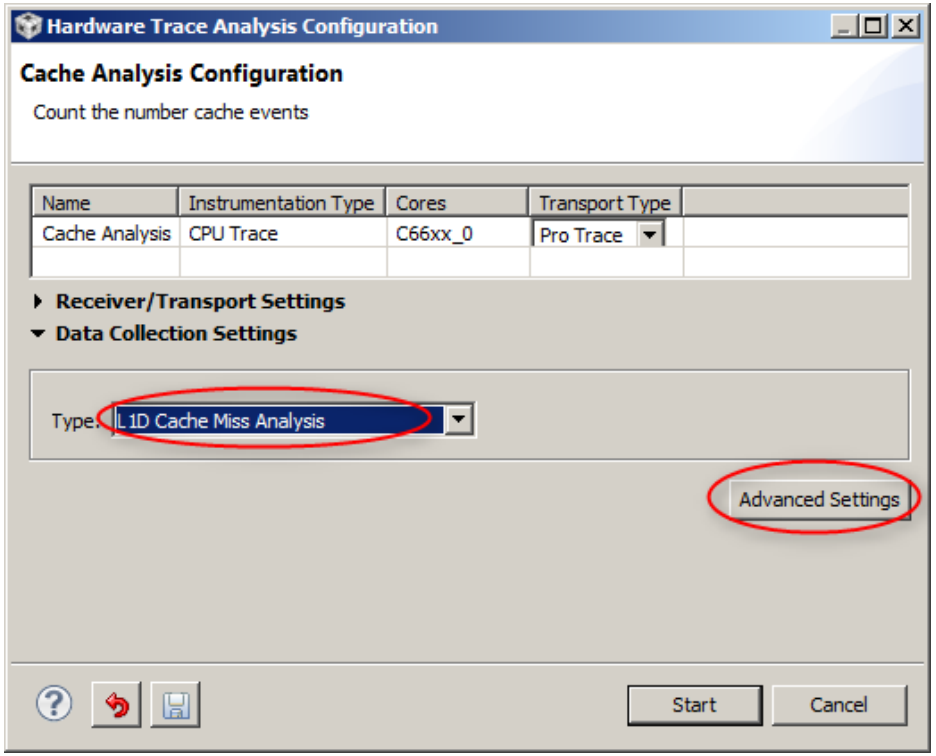
- Receiver
- MFENCE Workaround
- Pipeline Stall Analysis
- Trigger3

Properties	Values
Event 1	
CPU Stalls	false
L1P Stalls	true
L1D Stalls	false
Event 2	
CPU Stalls	false
L1P Stalls	false
L1D Stalls	true
L1D Bank Conflict	false
L1D Read Miss	true
L1D Write Buf Full	false
L1D Tag Update Buf I	false
L1D DMA Conflict	false
L1D Snoop Conflict	false
L1D Coherence Op C	false
L1D Tag Update on V	false
L1D Other	false

Running cache analysis (i)

As the name says, the cache analysis allows for a detailed view of the multiple cache operations in the system: hits, misses, flushes, etc.

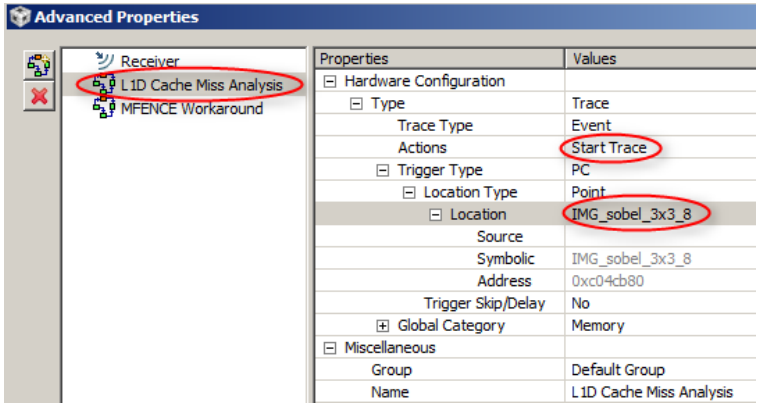
1. Open Cache Analyzer from CCS menu *Tools* → *Hardware Trace Analyzer* → *Cache Analysis*
2. Close Stall Profiler when requested
3. Click to reset to original settings
4. Change Transport/Receiver Type to *Pro Trace* and *Buffer Size* to*1MB*
5. Expand *Data Collection Settings* and for *Type* select *LID Cache Miss Analysis*
6. Click on *Advanced Settings*



Running cache analysis (ii)

7. Select *L1D Cache Miss Analysis* trigger in left pane

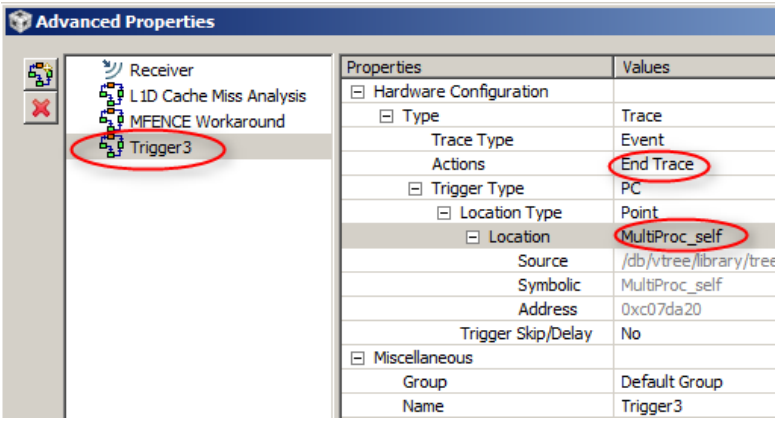
8. In right pane:
   - expand *Type* and change *Actions* to *Start Trace*
   - expand *Trigger Type* → *Location Type* → *Location* in left column and change *Location* to *IMG_sobel_3x3_8*



9. Expand *Global Category* to see what events are collected by default

10. Click on  in the left pane to add another trigger (default name can be changed)

11. Select *Trigger3* in the left pane and change *Actions* to *End Trace* and *Location* to *MultiProc_self*



Running cache analysis (iii)

12. Select *OK* in *Advanced Properties* dialog

13. Select *Start* in *Hardware Trace Analysis Configuration* dialog

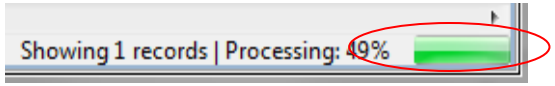
14. Complete step v of Running the demo

15. Wait for Demo to complete

16. Go to *Trace Viewer* view

17. Press *Stop* in *Trace Viewer* toolbar

15. Wait for *Trace Viewer* and *Cache Event Profiler* processing to complete



16. Go to *Cache Event Profiler* view

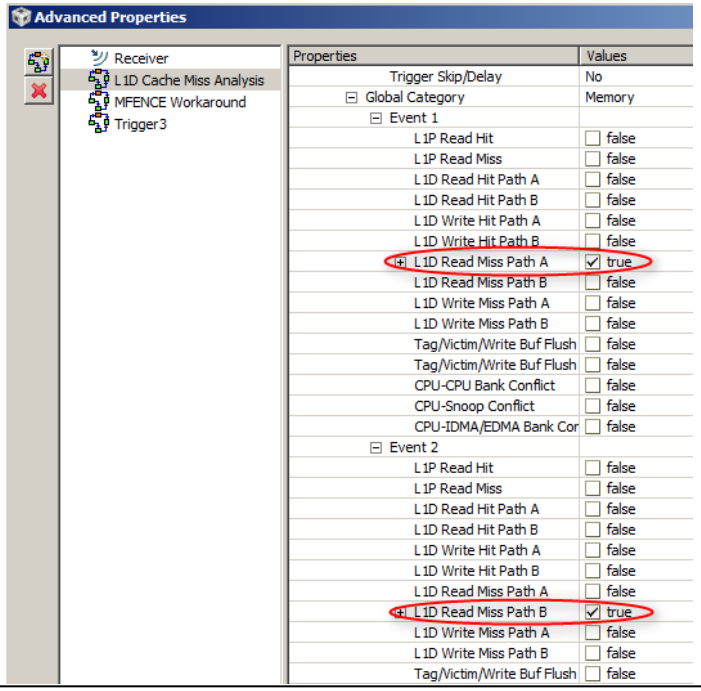
Analyzing cache results

1. Observe that the 1.46M L1D Read Miss cycles of **IMG_thr_le2min_8()** we had noted earlier is resulting from 8220 cache misses

Cache Event Profiler - C66xx_0 | *Trace Viewer - C66xx_0

	Function	Memory Event 1 Total	Memory Event 2 Total	Memory Event 3 Total	Memory Event 4 Total
1	unknown_0xc06fca0_0xc06fd5f_IMG_thr_le2min_8()	4110	4110	0	0
2	ti_sysbios_family_c64p_Hwi_dispatchC__I(int)	15	18	4	0
3	ti_sysbios_knl_Clock_workFunc__E(unsigned int, unsig...	0	15	0	0
4	ti_sysbios_knl_Clock_logTick__E()	0	6	0	0
5	ti_sysbios_family_c64p_Hwi_dispatchAlways()	6	6	2	0
6	process_rgb(struct processing_info *)	5	4	0	0
7	unknown_0xc0e0dc0_0xc0e0ddf_ti_sysbios_family_c6...	0	3	0	0

The reason for counting Memory Event 1 and Memory Event 2 is because Event 1 is set up to collect L1D Read Miss Path A and Event 2 is set up to collect L1D Read Miss Path B as seen in this screenshot



Using files to view data across analysis

Data collected by the Trace utilities can be saved for later analysis or to compare with data collected from other debug sessions

1. Open Function Profiler from *Tools* → *Hardware Trace Analyzer* → *Function Profiling*
2. Select *Close Cache Analysis* when requested
3. Verify that *Start Address* is still set to *IMG_sobel_3x3_8* and *End Address* is still set to *MultiProc_self*
4. Select *Start*
5. Complete step v of Running the demo
6. Wait for Demo to complete
7. Go to *Trace Viewer* view
8. Press *Stop* in *Trace Viewer* toolbar

Using files to view data across analysis

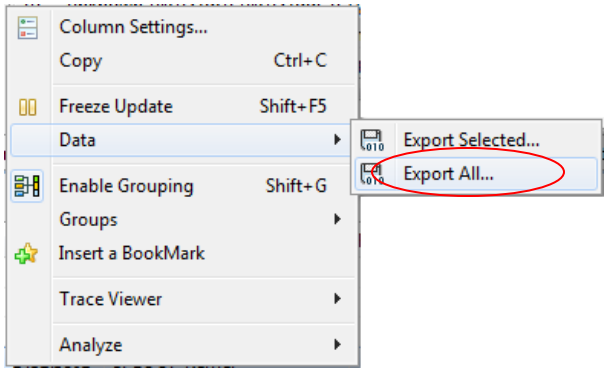
9. Wait for *Trace Viewer* and *Exclusive Function Profiler* processing to complete
10. Create a C:\temp folder on your hard disk if not already present
11. Select *Save* in *Trace Viewer* toolbar

12. Browse to C:\temp, specify *File Name as mytrace* and select *Save*
13. Click *Start* in *Trace Viewer* toolbar to restart tracing

14. Repeats steps 5 to 9 above to profile the application again
15. Select *Open File* from *Tools* → *Hardware Trace Analyzer* → *Open File*
16. Browse to C:\temp, select *File Name* <mytrace.tdf> and select *Open*
17. In *Trace Viewer – mytrace.tdf* view toolbar, select *Analyze* → *Exclusive Function Profiler*
18. Now current profile result can be visually compared with previously saved result

Exporting data

1. In *Exclusive Function Profiler – C66xx_0* view right-click-context menu, select *Data* → *Export All…* to export all records



2. (Optional) In the *Export Data* dialog add or remove columns to export
3. (Optional) In the *Export Data* dialog use the *Move* button to rearrange order in which columns are to be exported
4. Click on *Browse* and browse to C:\temp folder, specify *File Name* as *myexporttrace* and select *Save*
5. Select *OK* to export all records
6. Data is exported in CSV format which can be consumed by CCS and other tools such as Excel

Importing data

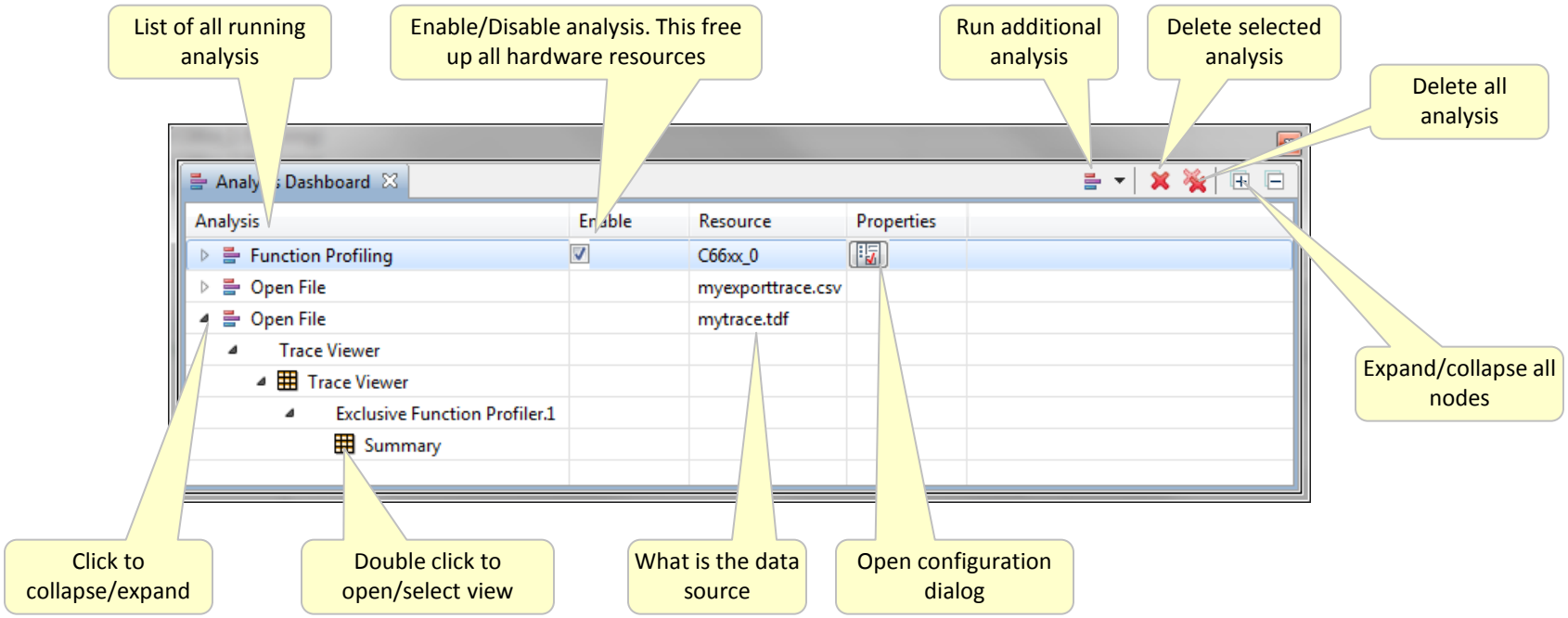
1. Select *Open File* from *Tools* → *Hardware Trace Analyzer* → *Open File*
2. At Bottom Right corner of *Open Trace File* dialog select *CSV trace data file (*.csv)*

3. Browse to C:\temp, select *File Name* <myexporttrace.csv>, select *Open*
4. The data from the csv file is now visible in the *Trace Viewer – myexporttrace.csv* view

Using analysis dashboard

The dashboard is a comprehensive list of all the Trace jobs set up at a given debug session

1. Select *Open File* from *Tools* → *Hardware Trace Analyzer* → *Analysis Dashboard*
2. Observe features of Dashboard shown below



3. Select Remove All to remove all running analysis

Lab 2: Summary

- DSP Trace can be used to profile hotspots in application
- Hotspots can further be analyzed using stall and cache profiling
- Data can be saved to binary file to use for comparison with future results or to share with others
- Data can be exported/imported via CSV file
- Analysis Dashboard provides access to all analysis

LAB 3:
Getting started with non-intrusive system trace (STM), SoC profiling and customization

Lab 3: Exercise summary

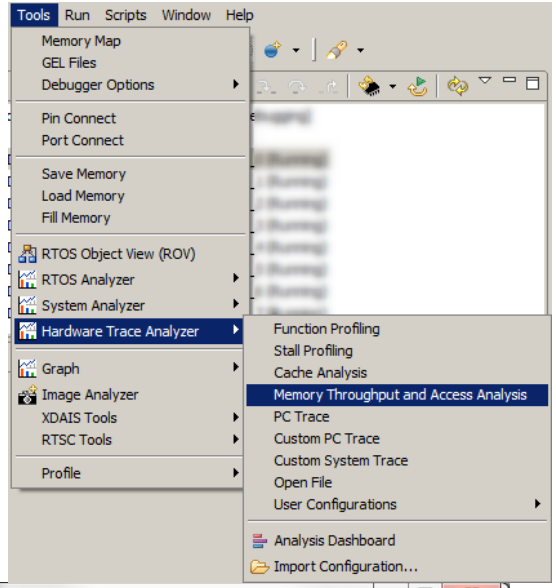
- **Key Objectives**
  - Perform Memory Throughput analysis
  - View core and system bandwidth utilization
  - View system latency
  - Customize STM SoC profiling for DDR and MSMC bandwidth and latency analysis

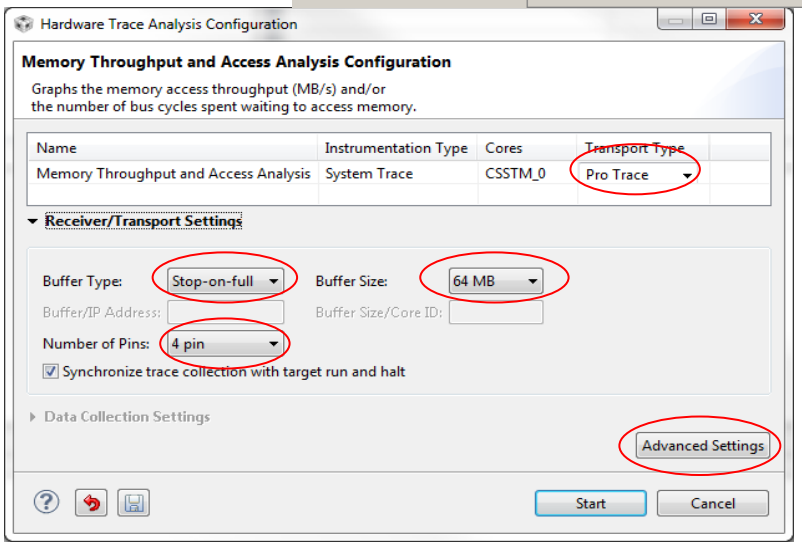
- **Tools and Concepts Covered**
  - External memory throughput analysis
  - Bus latency analysis
  - Internal memory controller (MSCM) throughput analysis

Setup memory throughput analysis (i)

As the name says, the memory throughput analysis tool allows measuring the data rate through various memory components of the system

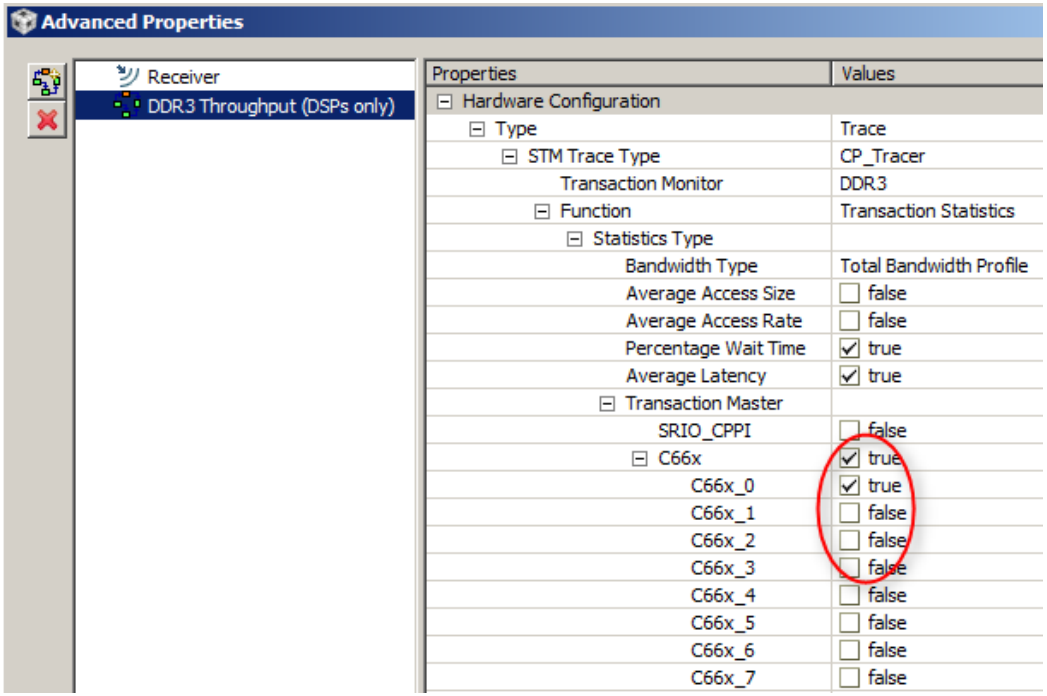
1. Continue from Lab 2. Make sure all cores are running in *Debug* view
2. Go to *Tools* → *Hardware Trace Analyzer* → *Memory Throughput and Access Analysis*
3. Select *Transport Type* as *Pro Trace*
4. Expand *Receiver/Transport Settings*
5. Select *Buffer Type* as *Stop-on-full*, *Buffer Size* as *64 MB* and *Number of Pins* as *4 pin*
6. Go to the *Advanced Settings*





Setup memory throughput analysis (ii)

7. By default, DDR3 memory throughput will be captured
8. In the right pane, expand *Type* → *STM Trace Type* → *Function* → *Statistics Type* →*Transaction Master*
9. Under *Transaction Master* leave only *C66x_0* (core 0) enabled and disable all other masters
10. Click *OK*
11. Click on *Start* to setup the trace
12. Run the demo by completing step v of Running the demo



Analysis view

1. Go to *Trace Viewer* view and click Stop
2. Wait for processing to complete

Showing 2,730,000 records | Processing: 21%

3. DDR CP tracer messages are captured in the *Trace viewer* view

*Trace Viewer - CSSTM_0 | Memory Throughput - CSSTM_0 | Minimum Average Latency - CSSTM_0 | Analyze | Start | Stop | Resume

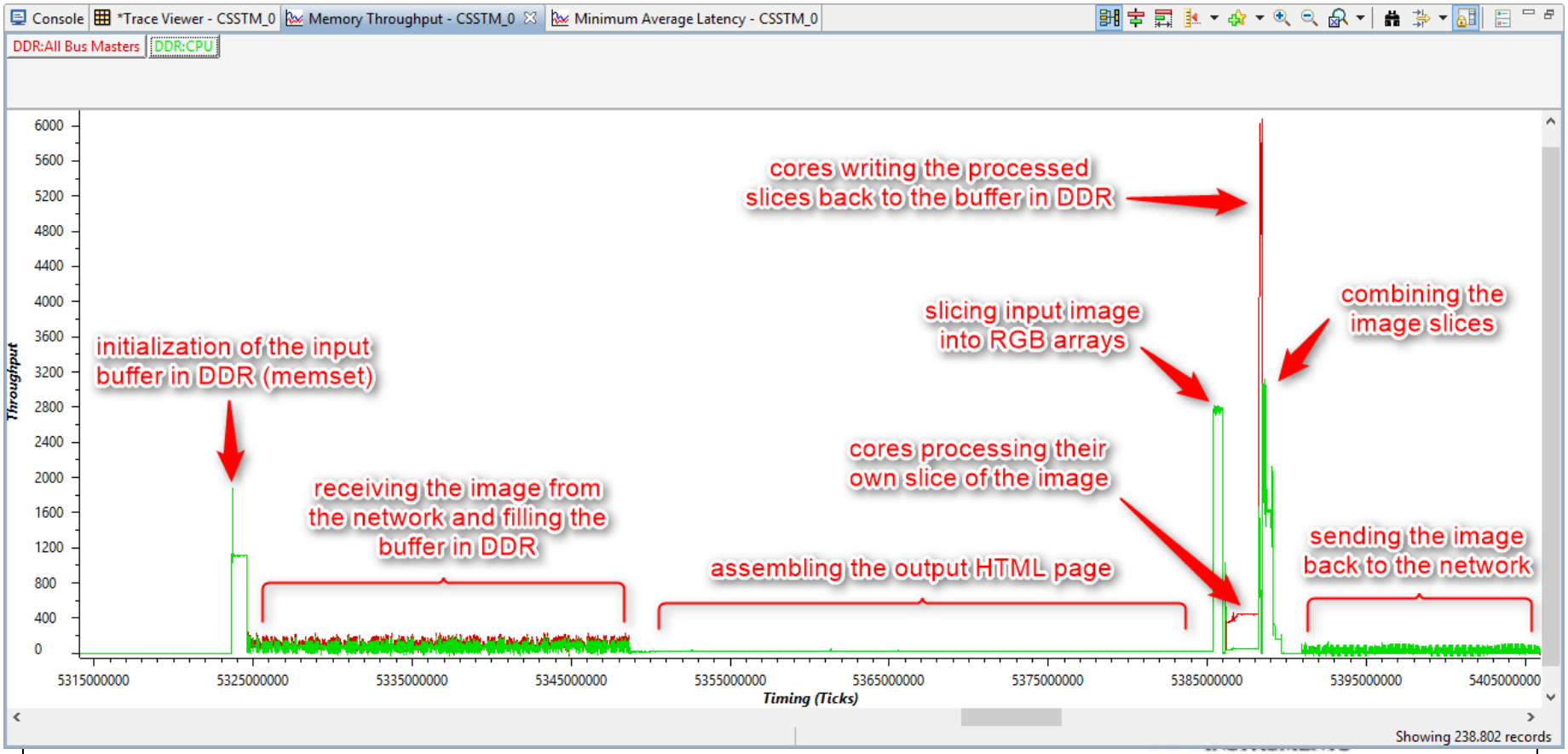
Stopped by buffer full : 100% full, 100% read

	Time	Micro Secs	Master Name	Data M...	Data	Class	Module	Domain	Trace Status	Receiver Status
1	0	0.0000							Start of trace	
2	51	0.1990	CPTracer		0x0	Bus Throughput(MBytes/s) = ThroughPutCounter1 / Sliding Time Window	CPT	DDR		
3	51	0.1990	CPTracer		0x0	Average Access Size(bytes/cycle) = ThroughPutCounter1 / Num Access Granted	CPT	DDR		
4	51	0.1990	CPTracer		0x0	Bus Utilization(Mil Transactions/s) = Num Access Granted / Sliding Time Window	CPT	DDR		
5	51	0.1990	CPTracer		0x0	Bus Contention(%) = Accumulated wait time / Sliding Time Window	CPT	DDR		
6	51	0.1990	CPTracer		0x0	Min. Average Latency (cycles / transaction) = Accumulated wait time / Num Access Granted	CPT	DDR		
7	51	0.1990	CPTracer		0x0	Selected bus throughput(MBytes/s) = ThroughPutCounter0 / Sliding Time Window	CPT	DDR		
8	17114	66.7863	CPTracer		0x0	Bus Throughput(MBytes/s) = ThroughPutCounter1 / Sliding Time Window	CPT	DDR		
9	17114	66.7863	CPTracer		0x0	Average Access Size(bytes/cycle) = ThroughPutCounter1 / Num Access Granted	CPT	DDR		
10	17114	66.7863	CPTracer		0x0	Bus Utilization(Mil Transactions/s) = Num Access Granted / Sliding Time Window	CPT	DDR		
11	17114	66.7863	CPTracer		0x0	Bus Contention(%) = Accumulated wait time / Sliding Time Window	CPT	DDR		
12	17114	66.7863	CPTracer		0x0	Min. Average Latency (cycles / transaction) = Accumulated wait time / Num Access Granted	CPT	DDR		
13	17114	66.7863	CPTracer		0x0	Selected bus throughput(MBytes/s) = ThroughPutCounter0 / Sliding Time Window	CPT	DDR		
14	34182	133.3932	CPTracer		0x0	Bus Throughput(MBytes/s) = ThroughPutCounter1 / Sliding Time Window	CPT	DDR		
15	34182	133.3932	CPTracer		0x0	Average Access Size(bytes/cycle) = ThroughPutCounter1 / Num Access Granted	CPT	DDR		
16	34182	133.3932	CPTracer		0x0	Bus Utilization(Mil Transactions/s) = Num Access Granted / Sliding Time Window	CPT	DDR		
17	34182	133.3932	CPTracer		0x0	Bus Contention(%) = Accumulated wait time / Sliding Time Window	CPT	DDR		

Showing 6,993,188 records

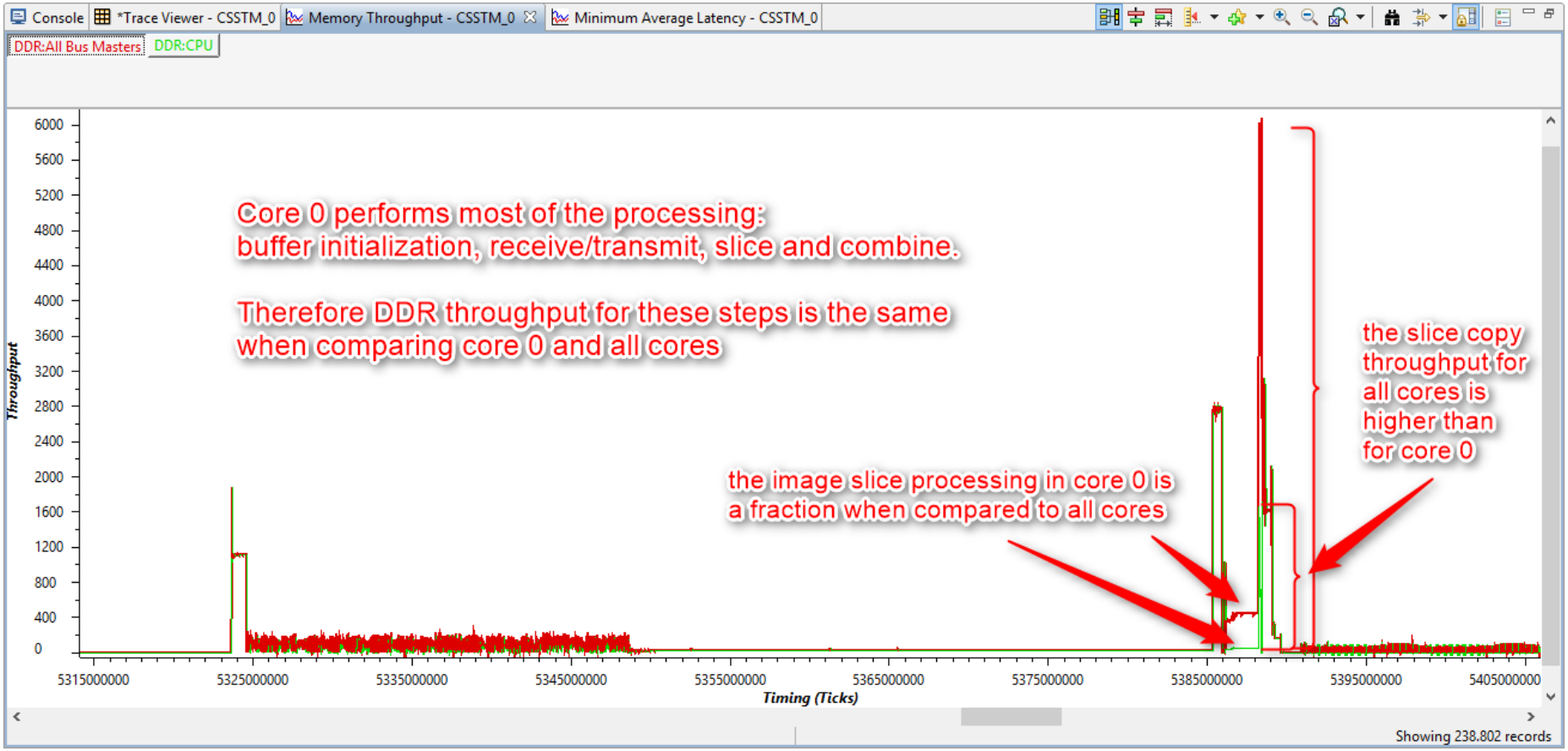
View core0 DDR3 bandwidth utilization (i)

1. Go to *Memory Throughput – CSSTM_0* view and select *DDR:CPU* tab
2. Click on zoom out icon several times (~10-12 times) until you see a view of the image processing portion
3. Zoom to the portion of the graph where the image is being processed using the left mouse button together with the Alt key.



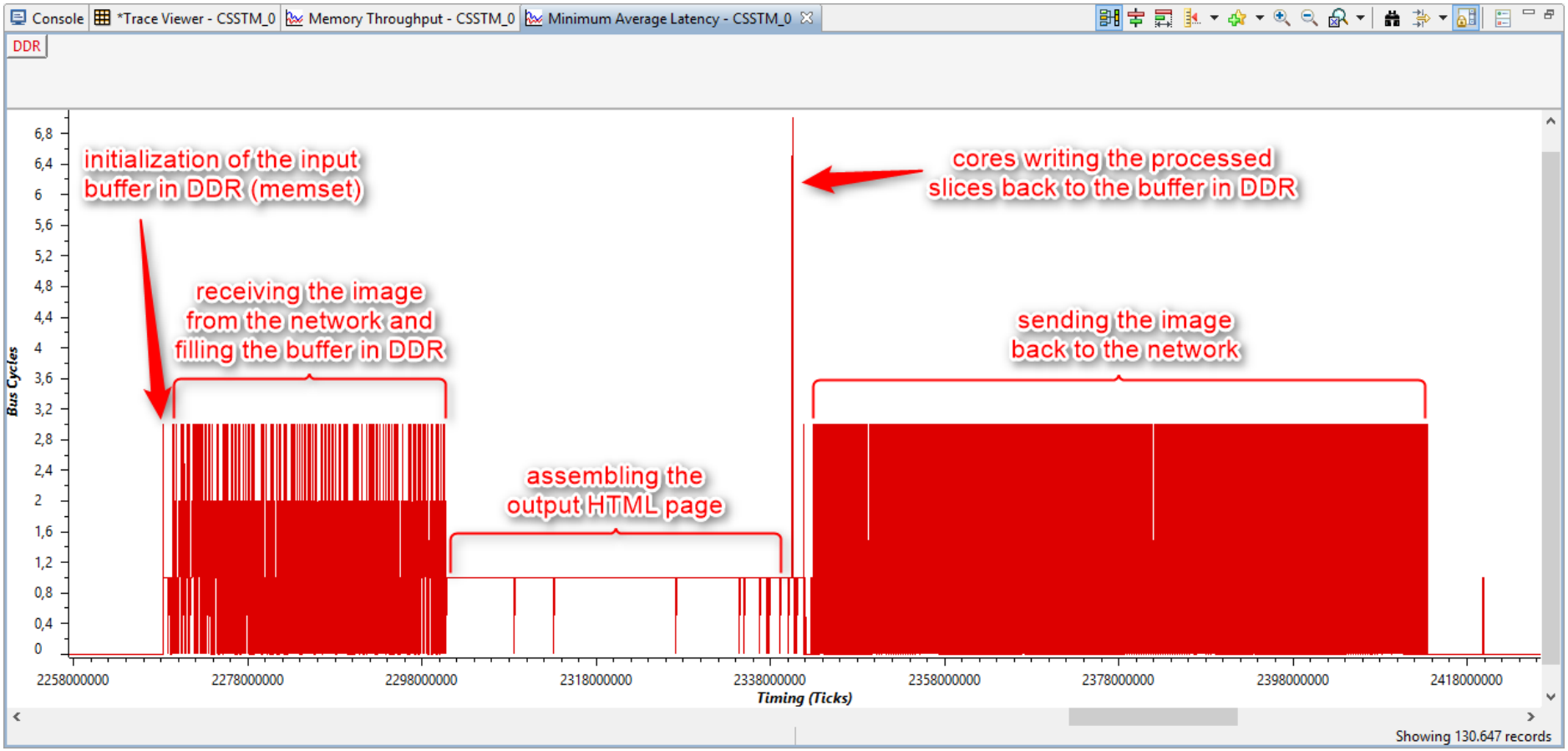
View core0 DDR3 bandwidth utilization (ii)

4. Now select *DDR:All Bus Masters* tab to highlight the throughput across all the cores.



View DDR3 latency

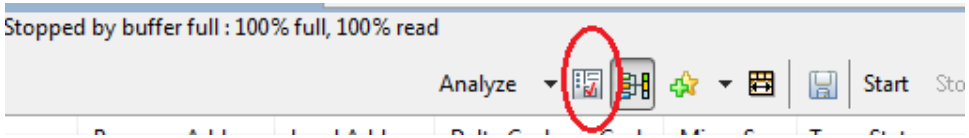
5. Select *Minimum Average Latency – CSSTM_0* view and zoom to the portion of the graph where the image is being processed



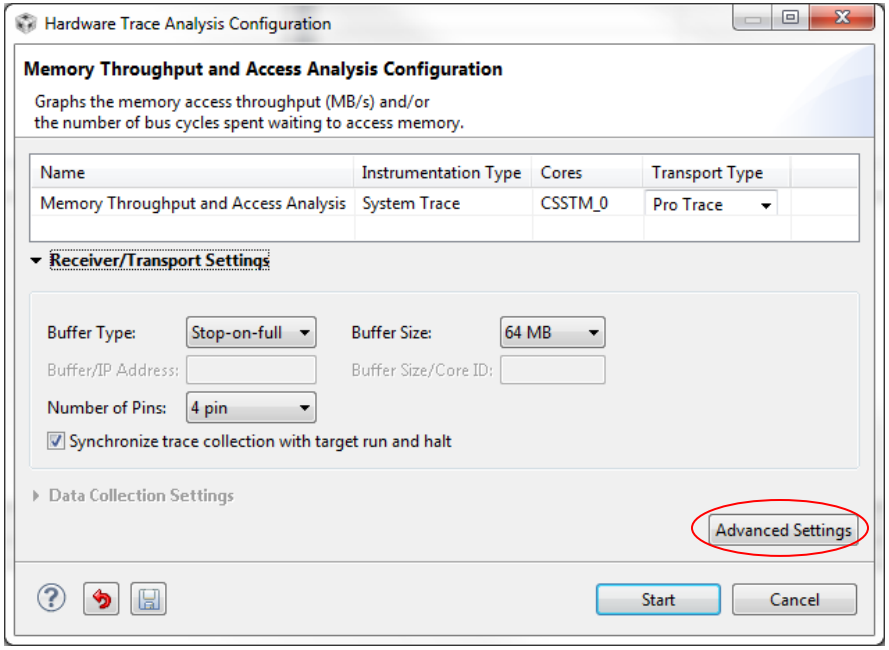
Customize for MSMC along with DDR (i)

The MSMC is the memory controller of the C6678 device, therefore it has visibility on all memory transactions and not only the external DDR

1. Go to *Trace Viewer* view
2. Click on *Analysis Properties* for bringing up setup configuration dialog box

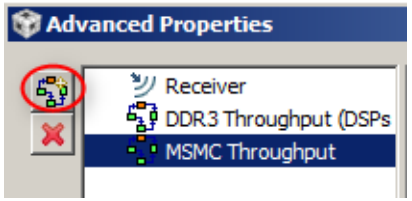


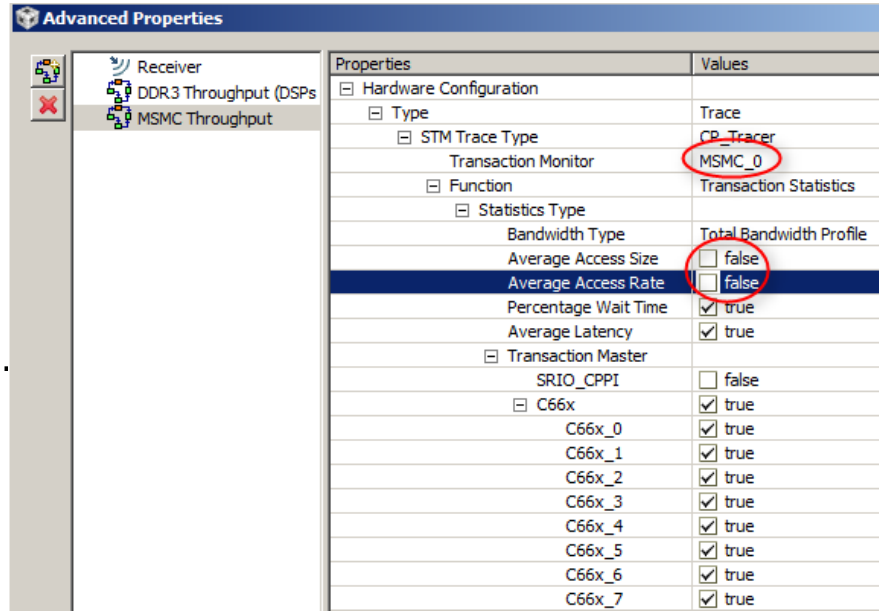
3. Go to the *Advanced Settings*



Customize for MSMC along with DDR (ii)

4. Notice that by default, DDR3 memory throughput will be captured
5. Now add a custom trigger for capturing MSMC memory throughput by clicking on New Trace Trigger icon
6. In the right pane, expand *Miscellaneous* and set *Name* to *MSMC Throughput*
7. Then select *Transaction monitor* as *MSMC_0* and *Average Access Size* and *Average Access Rate* as *false*
8. Click *OK*
9. Click on *Apply* to setup the trace
10. In *Debug* view, check that all cores are running. If they are not put them to run
11. Run the demo by completing step v of Running the demo





Analysis view

12. Go to *Trace Viewer* tab and click *Stop*
13. Wait for processing to complete
14. Observe that DDR and MSMC CP tracer messages are captured in the *Trace viewer*

*Trace Viewer - CSSTM_0 | Memory Throughput - CSSTM_0 | Minimum Average Latency - CSSTM_0 | Console | Analyze | Start | Stop | Resum

Stopped by user : 87% full, 87% read

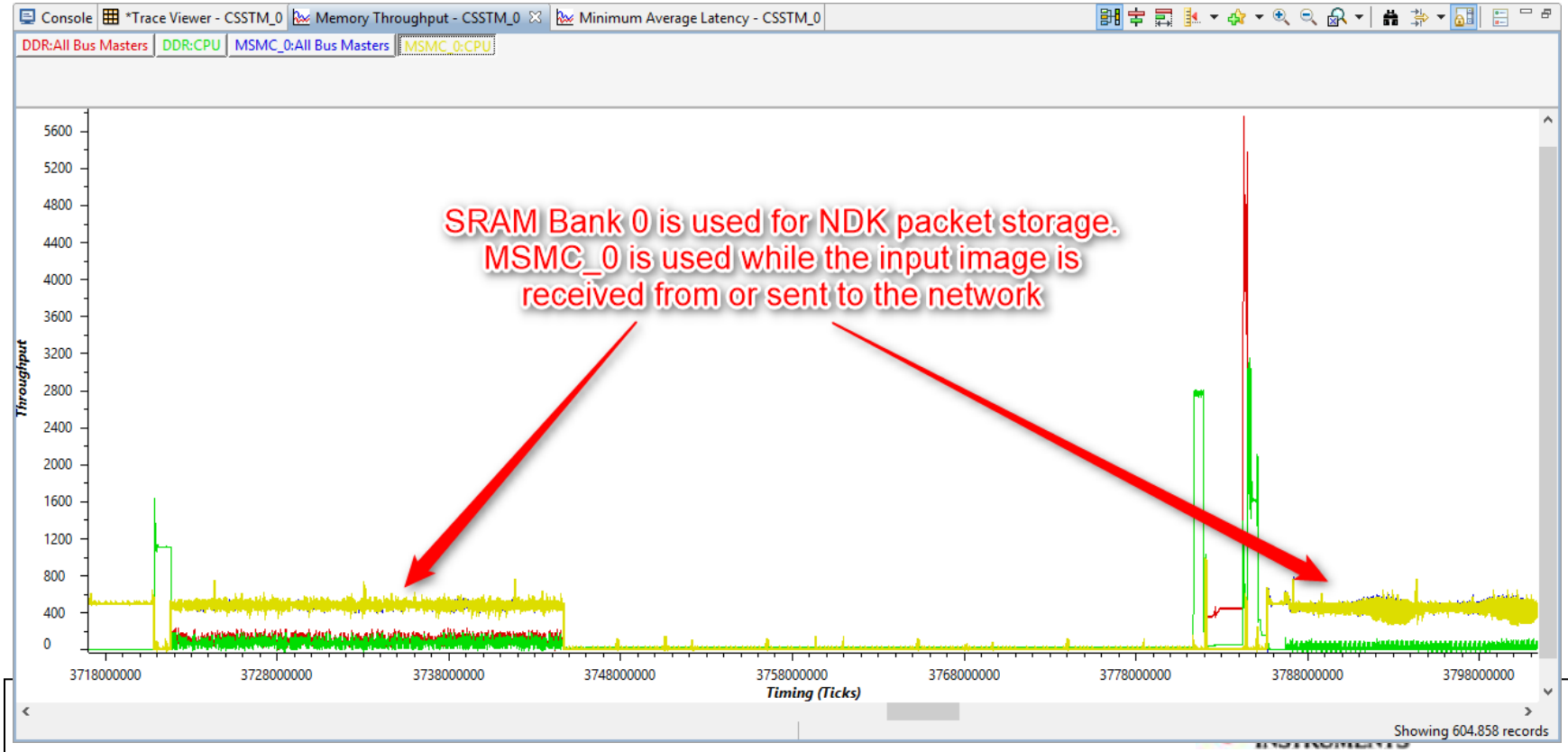
	Time	Micro Secs	Master Name	Data M...	Data	Class	Module	Domain
262...	38279929328	149385090.0605	CPTracer		0xA	Bus Utilization(Mil Transactions/s) = Num Access Granted / Sliding Time Window	CPT	MSMC_0
262...	38279929328	149385090.0605	CPTracer		0x2	Bus Contention(%) = Accumulated wait time / Sliding Time Window	CPT	MSMC_0
262...	38279929328	149385090.0605	CPTracer		0x1	Min. Average Latency (cycles / transaction) = Accumulated wait time / Num Access Granted	CPT	MSMC_0
262...	38279929328	149385090.0605	CPTracer		0x156	Selected bus throughput(MBytes/s) = ThroughPutCounter0 / Sliding Time Window	CPT	MSMC_0
262...	38279940536	149385133.7990	CPTracer		0x0	Bus Throughput(MBytes/s) = ThroughPutCounter1 / Sliding Time Window	CPT	DDR
262...	38279940536	149385133.7990	CPTracer		0x0	Average Access Size(bytes/cycle) = ThroughPutCounter1 / Num Access Granted	CPT	DDR
262...	38279940536	149385133.7990	CPTracer		0x0	Bus Utilization(Mil Transactions/s) = Num Access Granted / Sliding Time Window	CPT	DDR
262...	38279940536	149385133.7990	CPTracer		0x0	Bus Contention(%) = Accumulated wait time / Sliding Time Window	CPT	DDR
262...	38279940536	149385133.7990	CPTracer		0x0	Min. Average Latency (cycles / transaction) = Accumulated wait time / Num Access Granted	CPT	DDR
262...	38279940536	149385133.7990	CPTracer		0x0	Selected bus throughput(MBytes/s) = ThroughPutCounter0 / Sliding Time Window	CPT	DDR
262...	38279946395	149385156.6634	CPTracer		0x157	Bus Throughput(MBytes/s) = ThroughPutCounter1 / Sliding Time Window	CPT	MSMC_0
262...	38279946395	149385156.6634	CPTracer		0x20	Average Access Size(bytes/cycle) = ThroughPutCounter1 / Num Access Granted	CPT	MSMC_0
262...	38279946395	149385156.6634	CPTracer		0xA	Bus Utilization(Mil Transactions/s) = Num Access Granted / Sliding Time Window	CPT	MSMC_0

View MSMC_0 bandwidth for core 0

15. Select *Memory Throughput – CSSTM_0* view and select *MSMC_0:CPU* tab

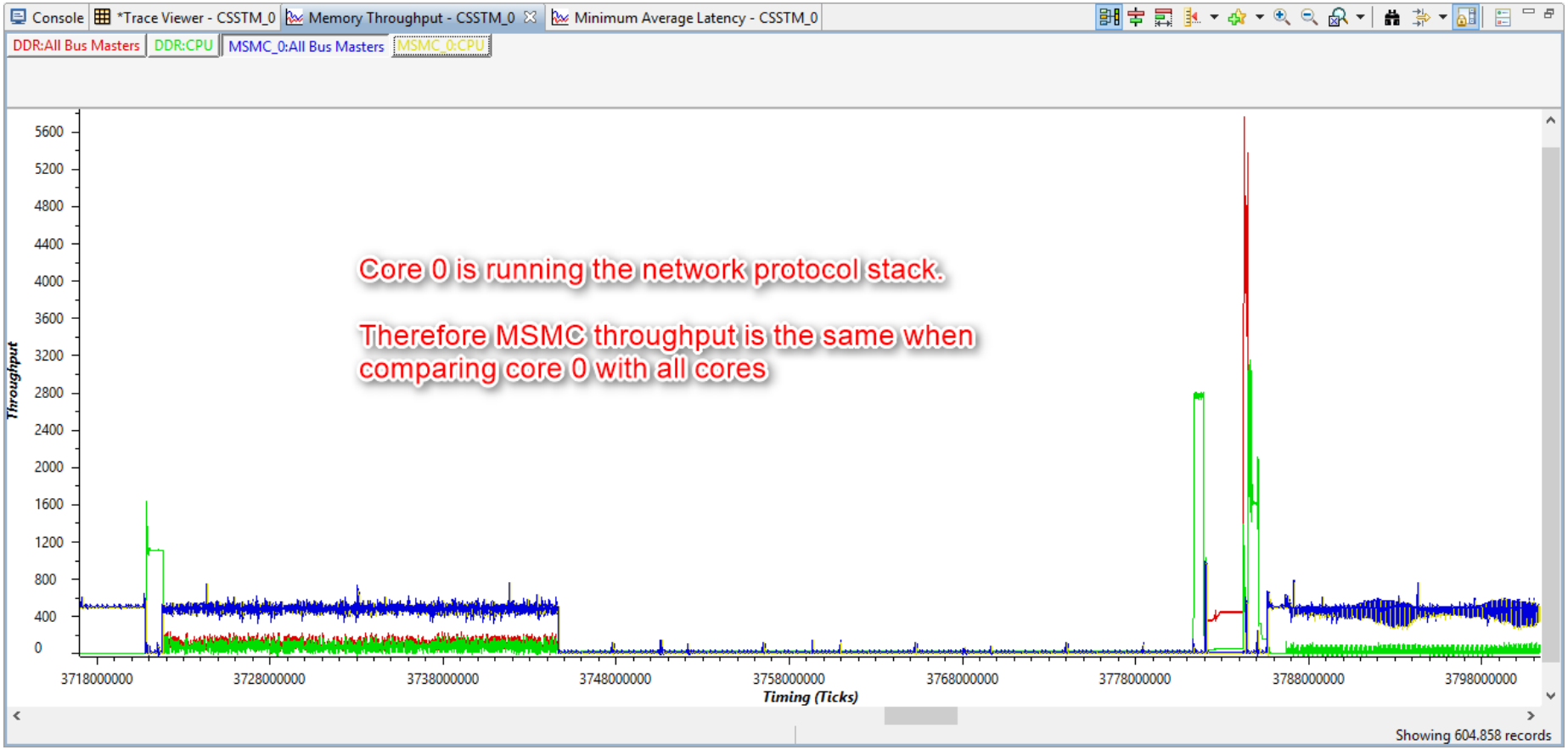
16. Click on zoom out icon several times (~10-12 times) until you see a view of the image processing portion

17. Zoom to the portion of the graph where the image is being processed (Alt key plus left mouse button). Note that MSMC information overlays the existing DDR data for comparison



View MSMC_0 bandwidth for all cores

18. Now select *MSMC_0:All Bus Masters*



Lab 3: Summary

- Using the MCSDK image processing demo, compared Core0's DDR3 bandwidth usage with the complete system's DDR3 bandwidth usage
- Using the MCSDK image processing demo, captured system's DDR3 latency
- Analyzed DDR3 memory performance and access analysis for MCSDK image processing demo
- Customized non-intrusive SoC profiling (memory performance and access analysis) job to add MSMC memory bandwidth measurement to the default DDR memory bandwidth job
- Using the MCSDK image processing demo, compared all cores (0-7) MSMC (bank0) bandwidth usage with the complete system's MSMC (bank0) bandwidth usage
- Similar to MSMC_0, bandwidth at any other memory end point (core0 L2, core1 L2…) can be measured

Questions?

Texas Instruments