Qualcomm® Hexagon™ Profiler

User Guide

80-N2040-10 Rev. E

May 23, 2022

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

Contents

Figures

1 Introduction

This document describes the Qualcomm® Hexagon™ profiler, which is the primary profiling tool for the Hexagon processor. The profiler displays information about the execution history of a program written for a Hexagon processor. Use it to identify any processor stalls in a program that can potentially be avoided.

This document is intended for experienced C programmers with assembly language experience.

1.1 Conventions

Computer text, code names, and code samples appear in a different font, for example, `printf("Hello world\n")`.

The following notation is used to define command syntax:

- Square brackets enclose optional items, for example, `[label]`.
- **Bold** indicates literal symbols, for example, [**comment**].
- The vertical bar character, `|`, indicates a choice of items.
- Parentheses enclose a choice of items for example, (**add**|**del**).
- An ellipsis, `...`, follows items that can appear more than once.

1.2 Technical assistance

For assistance or clarification on information in this document, submit a case to Qualcomm Technologies, Inc. (QTI) at https://createpoint.qti.qualcomm.com/.

If you do not have access to CreatePoint, register for access or send email to qualcomm.support@qti.qualcomm.com.

2 Use the profiler

The Hexagon profiler can profile all Hexagon processor applications: standalone, RTOS-based, single-threaded, or multi-threaded. The profiler performs *post-mortem* processing of the target application; that is, it is used after the target application has completed executing.

Before an application can be profiled, the profile data file must first be created for it. As shown in Figure 2-1, the Hexagon profiler obtains its profiling information from a profile data file that the Hexagon simulator generates when simulating the target application. When you input this data file into the profiler, the profiler translates it into an HTML file that you can then view in a web browser.

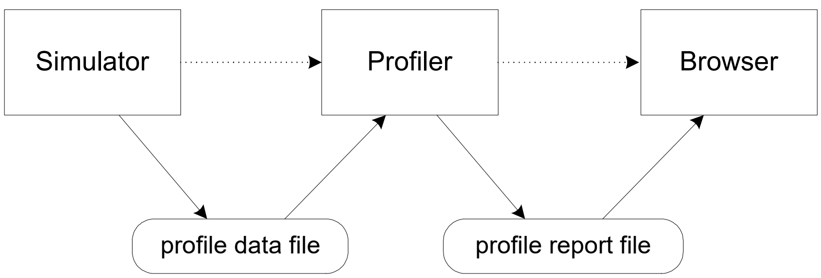


Figure 2-1 Using the Hexagon profiler

2.1 Create the profile data file

When you run an application in the simulator, the simulator generates the profile data file while it executes the target application.

Simulate the application on the command line with the `--packet_analyze` option. This option instructs the simulator to generate a packet statistics file in JSON format, which is used as the profile data file.

For more information on packet statistics files, see the *Qualcomm Hexagon Simulator User Guide*.

2.2 Run the Hexagon profiler

After the simulation is finished, input the generated profile data file into the Hexagon profiler. The profiler will then generate the profile information report in HTML format.

NOTE: The Hexagon profiler operates as a command-line utility.

2.2.1 Command

From the command line, enter the following:

```
hexagon-profiler [options...]
```

2.2.2 Options

```
--dockMenu
--elf=<file>[:reloc][,<file>[:reloc],...]
--help
--hideJumpToFuncs
--highlightJumps
--json=<file>
--noChromeMagnifyFix
--noUnderlineJumps
-o <file>
--packet_analyze
--show_0x
--showLeadingZeros
--showPacketBraces
--showSelectedType
--tools_dir=<dir>
-v
```

Option details

--dockMenu

Start with the menu already open.

--elf=<file>**[:reloc][,**<file>**[:reloc],...]**

Input one or more Hexagon ELF/OBJ/LIB files for disassembly, with optional relocation offsets to match their memory locations when the Hexagon code was run by `hexagon-sim`.

The Hexagon application binary file (such as `app.elf`) that was profiled.

This file must be the same file that was executed on the simulator to create the profile data file. The file is in ELF format.

--help

Display the command usage information and exit.

`--hideJumpToFuncs`

Hide the function names of jump and call targets.

`--highlightJumps`

Highlight address links in the disassembly.

`--json=``<file>`

Input the packet statistics JSON `<file>` that was generated when the Hexagon simulator ran the ELF file: `hexagon-sim --timing --packet_analyze <file>`.

`--noChromeMagnifyFix`

Use Chrome 54+ as-is, with its new magnification.

`--noUnderlineJumps`

Hide address link underlines in the disassembly.

`-o` `<file>`

Place the output into `<file>`.

The profiler creates this file in HTML format, with the conventional extension, `.html`.

`--packet_analyze`

Produce an HTML file that shows packet analysis.

`--show_0x`

Show the hex indicator, `0x`, in the address columns.

`--showLeadingZeros`

Show the leading zeros in the address columns.

`--showPacketBraces`

Show braces around each packet in the disassembly.

`--showSelectedType`

Show only the type of stall or event selected.

`--tools_dir=``<dir>`

Find `hexagon-llvm-objdump` and other tools in `<dir>`.

`-v`

Enable Verbose mode.

2.2.3 Example

```
hexagon-profiler --packet_analyze --elf=app.elf --json=app.json
-o app.html
```

In this example, the `hexagon-profiler` reads both the `app.elf` executable binary file and the `app.json` data file created by the simulator, and then it generates the `app.html` file.

2.3 View the profile report file

When viewed in a web browser, the HTML report file presents an interactive document (user interface) that allows you to select and display the following profile information:

- Total number of cycles executed
- Total number of stall cycles
- Highest cycle or stall counts (by function or instruction packet)
- Commit and stall statistics (by function or instruction packet)
- PMU event counts (by event type or instruction packet)
- Annotated disassembly of instruction packets
- Assembly instruction counts

Additional information, such as derived statistics and HVX commit and stall statistics, might be available depending on the processor version and features. For more details, see Chapter 3.

3 Hexagon profiler user interface

Use a web browser to view the report file that contains the profile information. The report file is in HTML format and presents an interactive document that allows you to select and display various types of profile information. This interactive document is called the *Hexagon profiler user interface*.

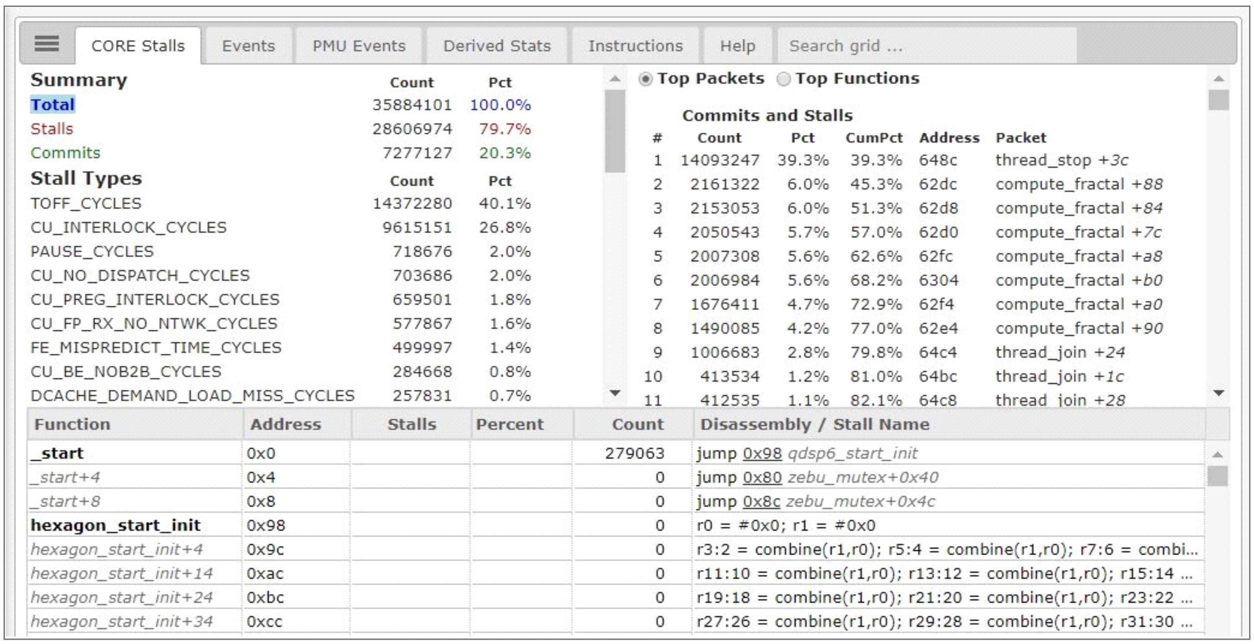


Figure 3-1 Hexagon profiler user interface

3.1 User controls

At the top of the user interface are a menu button (on the left) and profile information tabs. Depending on which tab is selected, the page displays the corresponding profile information. The default view is the CORE Stalls tab (Section 3.2).

3.1.1 Profile information tabs

The user interface displays several tabs:

Figure 3-2 Tabs

Tab	Profile Information	See
CORE Stalls	Total number of cycles and stalls executed Highest cycle or stall counts (by function or instruction packet) Commit and stall statistics (by function or instruction packet)	Section 3.2
	A **Search grid** tab appears at the far right of the tabs	Section 3.8
Events	Event count statistics (by function or instruction package) Disassembled listing of instruction packets	Section 3.3
	A **Search grid** tab appears at the far right of the tabs	Section 3.8
PMU Events	PMU event counts	Section 3.4
Derived Stats	Commonly used PMU events calculations	Section 3.5
Instructions	Hexagon assembly instructions that were profiled, with counts	Section 3.6
Help	Basic information about what was profiled	Section 3.7

The following tab appears only if the Hexagon ELF file contained HVX instructions.

Tab	Profile Information	See
HVX Stalls	Cycle counts and cycle count statistics for HVX packets	Section 3.9

3.1.2 Menu of disassembly configuration options

The menu button displays a Disassembly Panel, which lists configuration options for disassembly information to be displayed on the CORE Stalls and Events tabs. The options vary according to each tab.

Figure 3-3 Menu icon

These options change the way the following profile information is displayed.

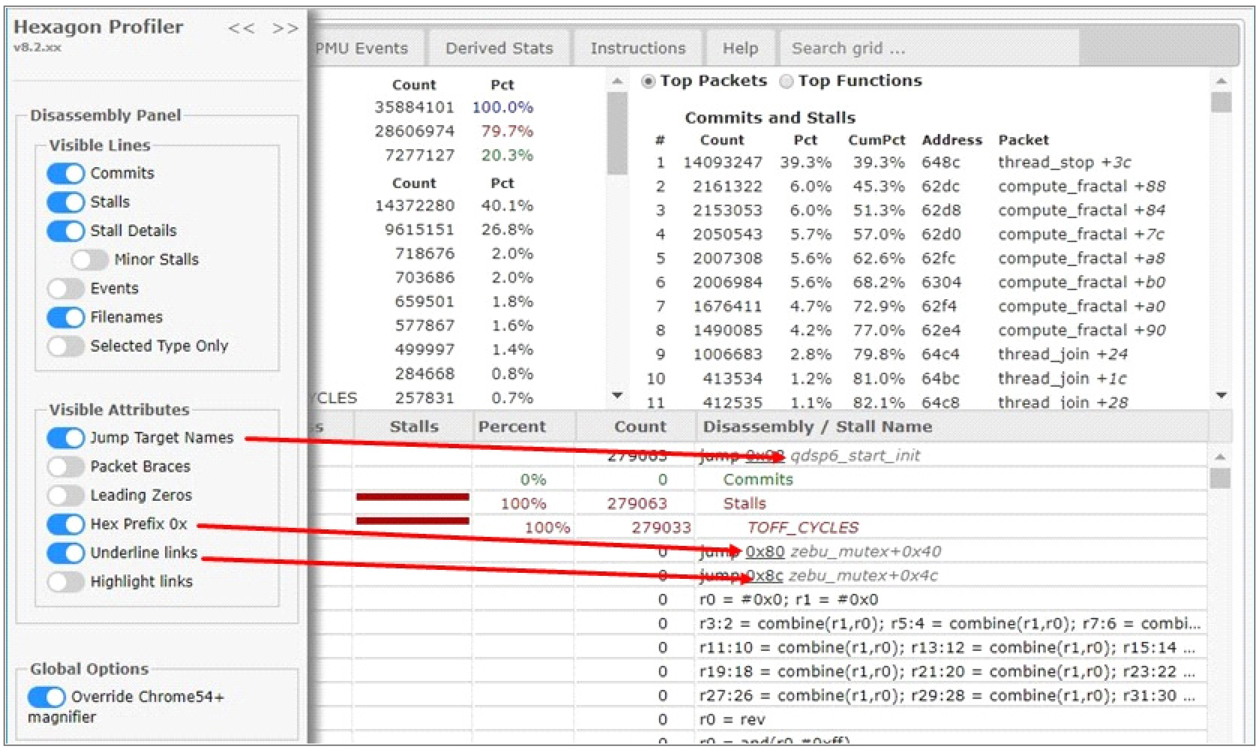


Figure 3-4 Disassembly Panel configuration options

Option	Profile information
Commits Stalls	Cycle counts and statistics showing how often an instruction packet executed without stalling (commit) versus how often it stalled during execution (stall).
Stall Details	Same as Commits/Stalls but includes cycle counts and statistics for each stall type that occurred to an instruction packet.
Minor Stalls	Same as Stall Details but includes the stall types that caused less than one percent of the total instruction packet stalls (called *minor stalls*).
Events	Enables display of the count and type of each event that occurred with an instruction packet.
Filenames	Enables display of the source file name and line number for each line of disassembly, when available.

Option	Profile information
Selected Type Only	Limits the disassembly display to the currently-selected Event or Stall type.
Jump Target Names	Displays the symbol name after the jump-to address in the Disassembly column.
Packet Braces	Adds curly braces around instruction packets in the Disassembly column.
Leading Zeros	Adds leading zeros to address values in the Address column.
Hex Prefix 0x	Adds **0x** to the beginning of address values in the Address column.
Underline links	Underlines all hyperlinked addresses in the Disassembly column.
Highlight links	Yellow highlights all hyperlinked addresses in the Disassembly column.
Global Options	Might appear depending on your browser. For example, the Override Chrome Magnifier option appears only for the Chrome browser.

3.2 CORE Stalls tab

When you open a profile report file in a web browser, the CORE Stalls tab displays by default.

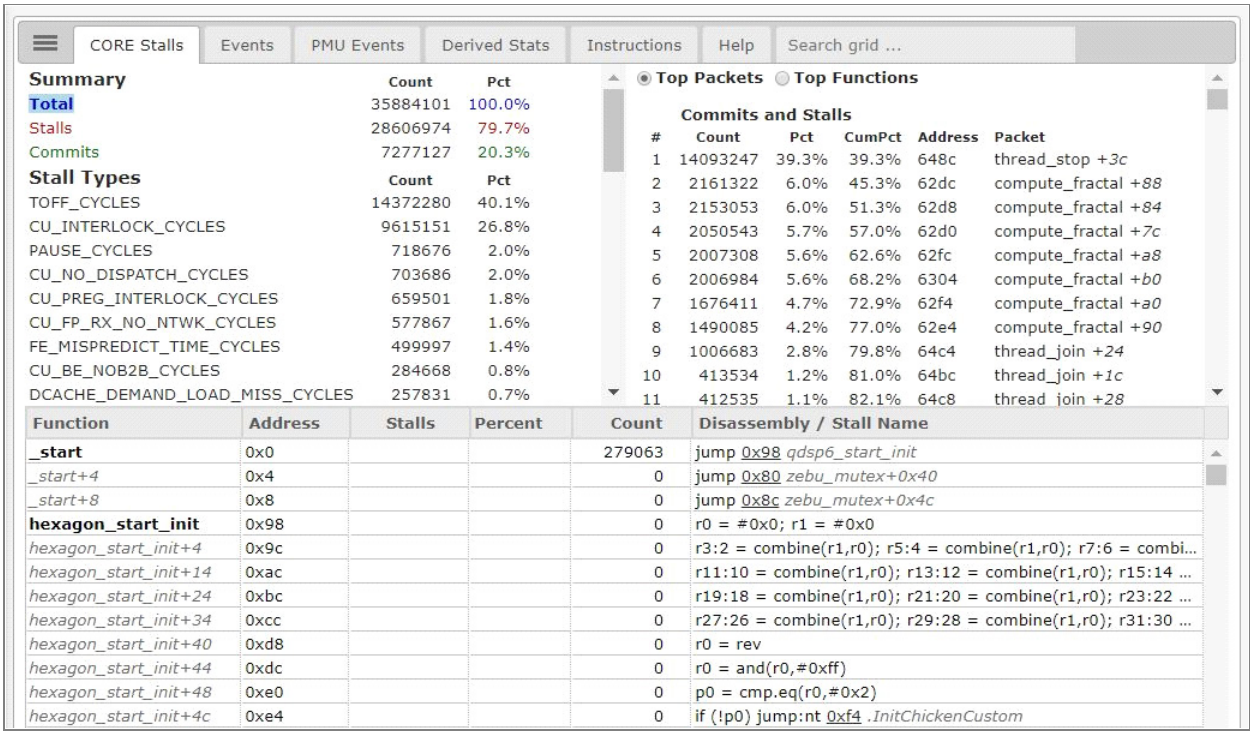


Figure 3-5 CORE Stalls tab

This tab displays the cycle counts and cycle count statistics for instruction packets:

- The top left pane lists the total cycles and stall cycles for the program, followed by a list of the stall cycles categorized by stall type.

  Clicking a hyperlinked item (Total, Stalls, Commits, or a stall type) in the top left pane changes the top right pane so it lists the instruction packets with the highest cycle counts for the selected item.

- The top right pane lists the highest cycle or stall counts (by function or instruction packet).

  Clicking the PC address of an instruction packet changes the bottom pane to show a disassembled listing of the specified instruction packet and the packets that follow it.

- The bottom pane provides a disassembled listing of instruction packets along with detailed Commit/Stall cycle counts and statistics for each by function or instruction packet.

- A Search grid tab also appears. It allows you to quickly search for a function (symbol) (see Section 3.8).

The profile information displayed in these panes is controlled by the Disassembly Panel options and the hyperlinks that appear in the top and bottom panes.

3.2.1 Display totals

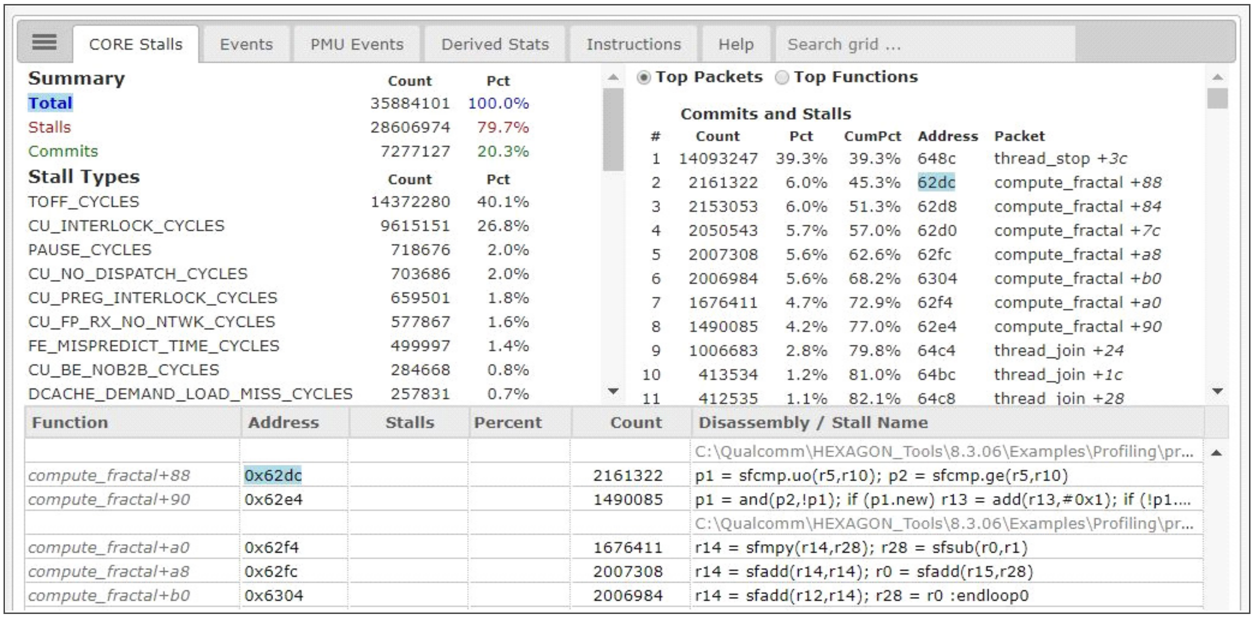


Figure 3-6 CORE Stalls tab: total commits and stalls

In the top left pane:

- Click the **Total** item to change the top right pane to display the total number of both commits and stalls executed in the program (this is the default view).
- Click either the **Stalls** or **Commits** item to change the top right pane to display only the respective stalls/commit statistics.

In the top right pane:

- Select **Top Functions** or **Top Packets**. The cycle count, percentage, and cumulative percentage for each instruction packet listed are displayed. The packets are identified by their PC addresses, which are hyperlinked.
- Click the PC address to change the bottom pane to display a disassembled listing of the program code, starting at the specified instruction packet.

  For example, the **0x62dc** address is the PC address of an instruction packet that was executed 2161322 times in the program, and it is contained in the `compute_fractal()` function.

The bottom pane displays the PC address, which contains the function and cycle count for each disassembled instruction packet.

3.2.2 Display commits and stalls

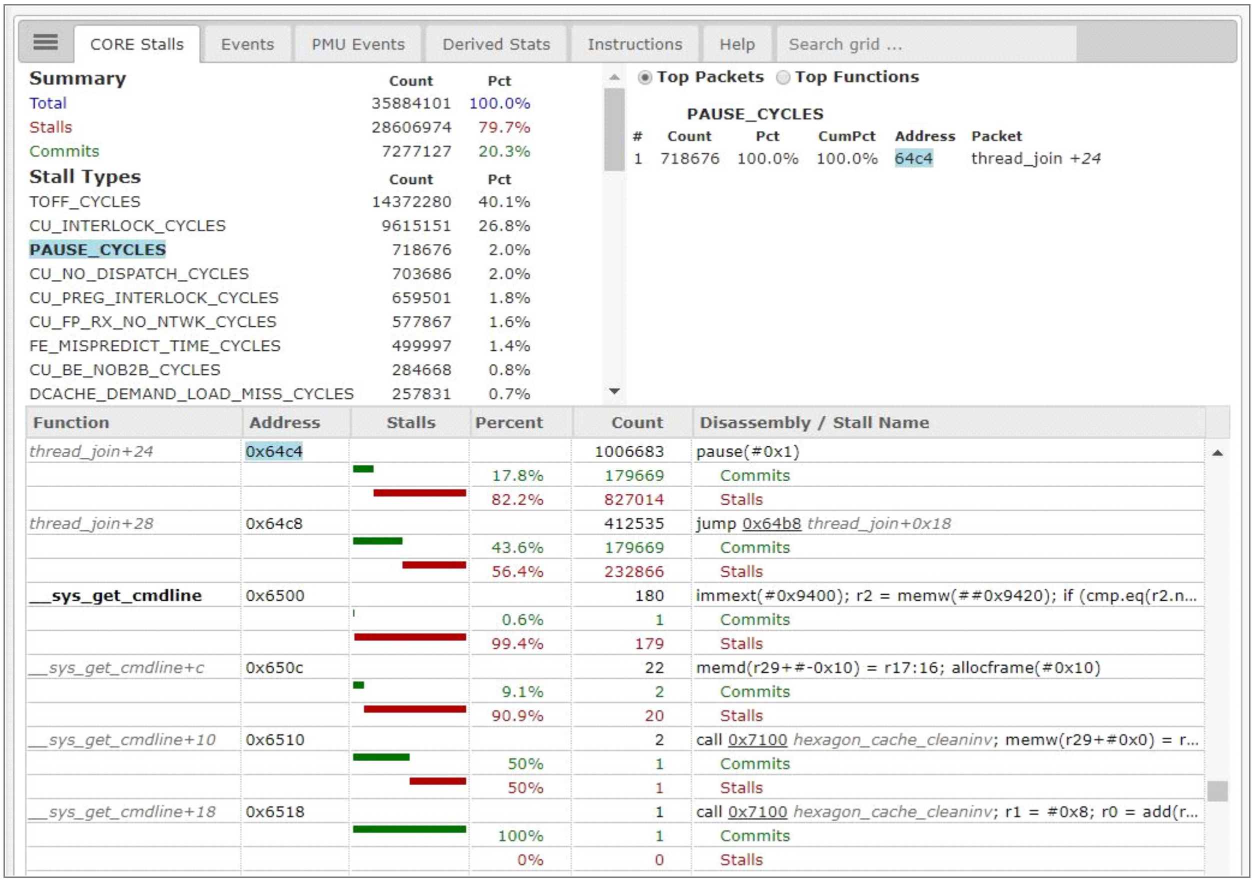


Figure 3-7 CORE Stalls tab: commits and stalls by stall type

In the top left pane, clicking a stall type changes the top right pane to display cycle counts and cycle count statistics that indicate how often an instruction packet executed without stalling (commit) versus how often it stalled during execution (stall).

To view commits or stalls, click the Menu button and select **Commits** or **Stalls** (or both) on the Disassembly Panel (see Section 3.1.2).

The page works like the Total display (see Section 3.2.1). Clicking on an item in the top left pane shows the corresponding information in the top right pane. Clicking an item in the top right pane shows the corresponding information in the bottom pane.

For example, clicking **PAUSE_CYCLES** shows the single instruction packet it specifies in the top right pane. Clicking that instruction address displays the following information in the bottom pane: PC address, which contains the function, commit and stall percentages, and cycle count for each disassembled instruction packet.

The commit and stall percentages are represented graphically, with green lines indicating the commit percentages and red lines indicating the stall percentages.

3.2.3 Display stall details

To view more stall details, click the Menu button and select **Stall Details** in the Disassembly Panel (see Section 3.1.2).

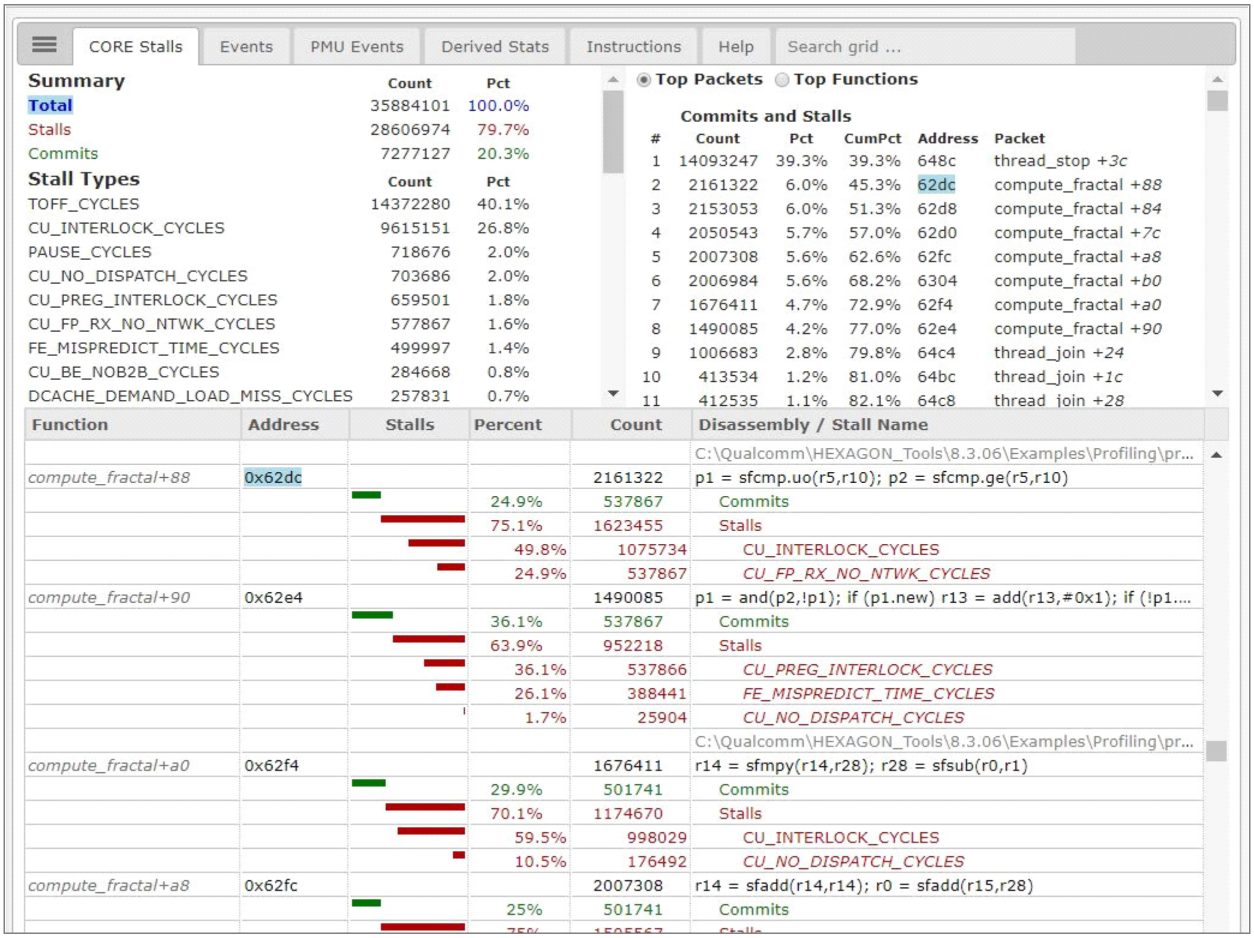


Figure 3-8 CORE Stalls tab: display stall details

This page works like the commits/stalls display (Section 3.2.2) but it also lists the separate cycle counts and statistics for each stall type that occurred to an instruction packet.

NOTE: Stall types with less than one percent of the total stalls are not listed; for more information see Section 3.2.4.

For example, the instruction packet at address **0x62dc** lists both the total number of stall cycles for this instruction packet (1623455) and how many of these stall cycles belong to the various stall types (CU_INTERLOCK_CYCLES, CU_FP_RX_NO_NTWK_CYCLES, and so on).

The bottom pane displays the PC address, which contains the function, commit, and stall percentages (both total and by stall type), and the cycle count for each disassembled instruction packet.

The commit and stall percentages are represented graphically, with green lines indicating the commit percentages and red lines indicating the stall percentages.

3.2.4 Display minor stalls

To view the stall types that caused less than one percent of the total stall cycles for an instruction packet (*minor stalls*), click the Menu button and select **Minor Stalls** in the Visible Lines portion of the Disassembly Panel (see Section 3.1.2).

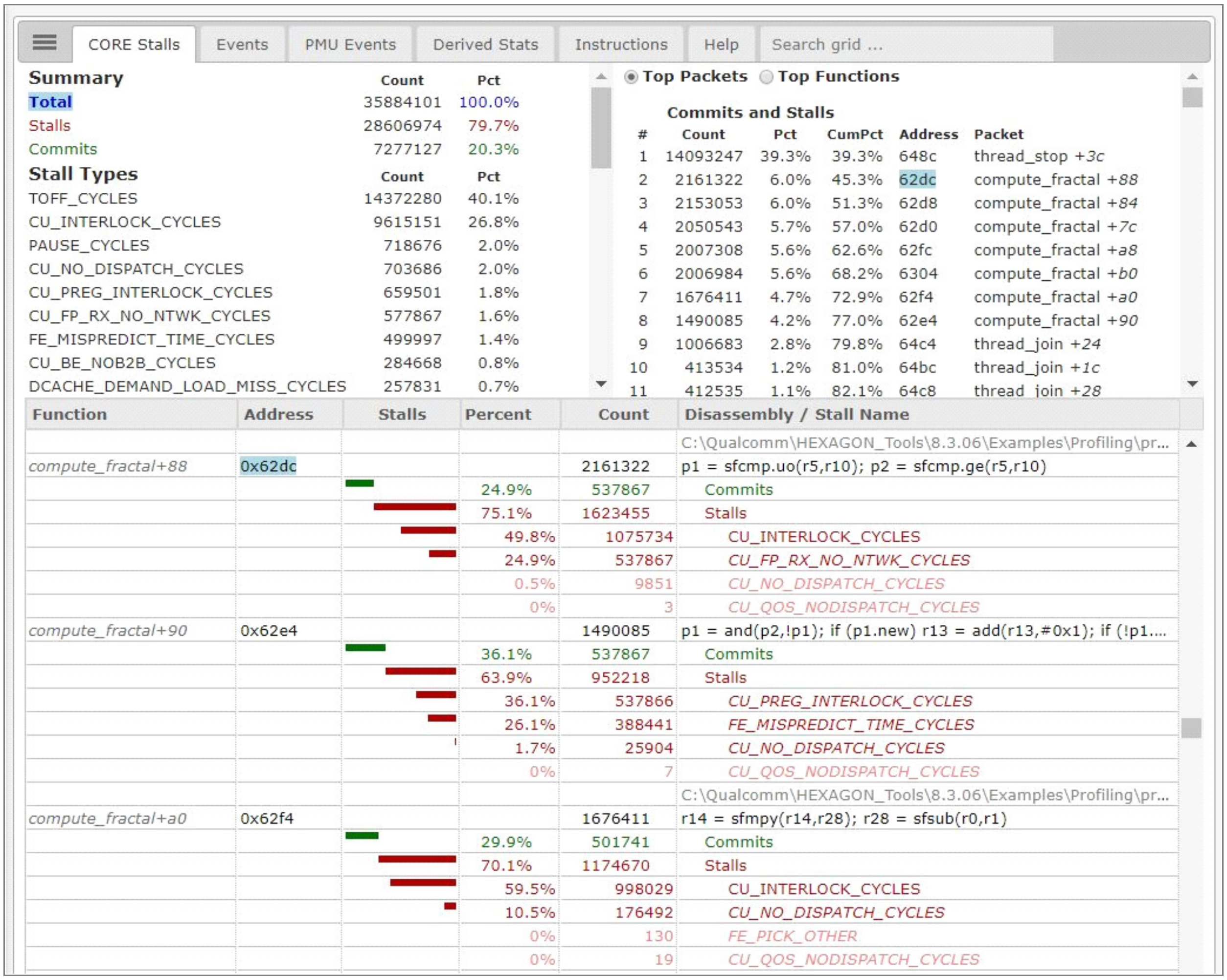


Figure 3-9 CORE Stalls tab: display minor stalls

This display works like the Stall Details display (Section 3.2.3). For example, in the bottom pane, the instruction packet at address **0x62dc** includes the minor stall type, CU_NO_DISPATCH_CYCLES, which has a stall percentage of less than one percent.

Minor stalls are displayed in pink (rather than the usual red).

3.3 Events tab

The Events tab displays information generated by the Hexagon performance management unit (PMU). The Hexagon processor architecture defines a PMU to support on-target performance tracking. (For more information, see the applicable *Qualcomm Hexagon Programmer's Reference Manual*).

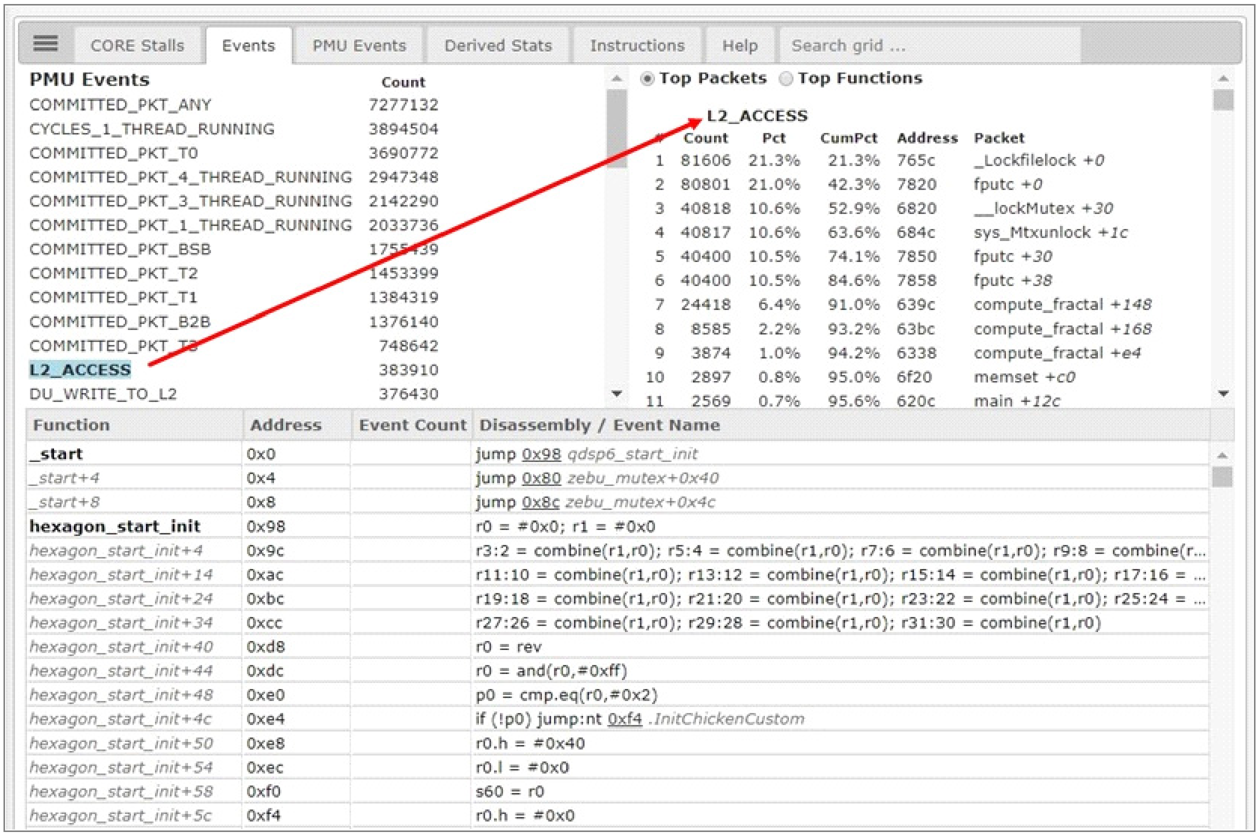


Figure 3-10 Events tab

The Events tab works like the CORE Stalls tab (Section 3.2). It displays information in three panes:

- The top left pane lists the name and event count for the PMU event types with the highest event counts.
- The top right pane displays the PMU event count, percentage, cumulative percentage, PC address, and containing function for each instruction packet listed.
- The bottom pane provides a disassembled listing of instruction packets along with detailed PMU event counts and statistics for each by function or instruction packet.
- A Search grid tab also appears. It allows you to quickly search for a function (symbol) (see Section 3.8).

For example, clicking the **L2_ACCESS** item in the top left pane changes the top right pane to list the instruction packets with the highest PMU event counts for L2_ACCESS. Clicking a PC address in the top right pane changes the bottom pane to display a disassembled listing of the program code, starting at the specified event.

3.4 PMU Events tab

The PMU Events tab shows the performance monitor event counts.

CORE Stalls | Events | PMU Events | Derived Stats | Instructions | Help

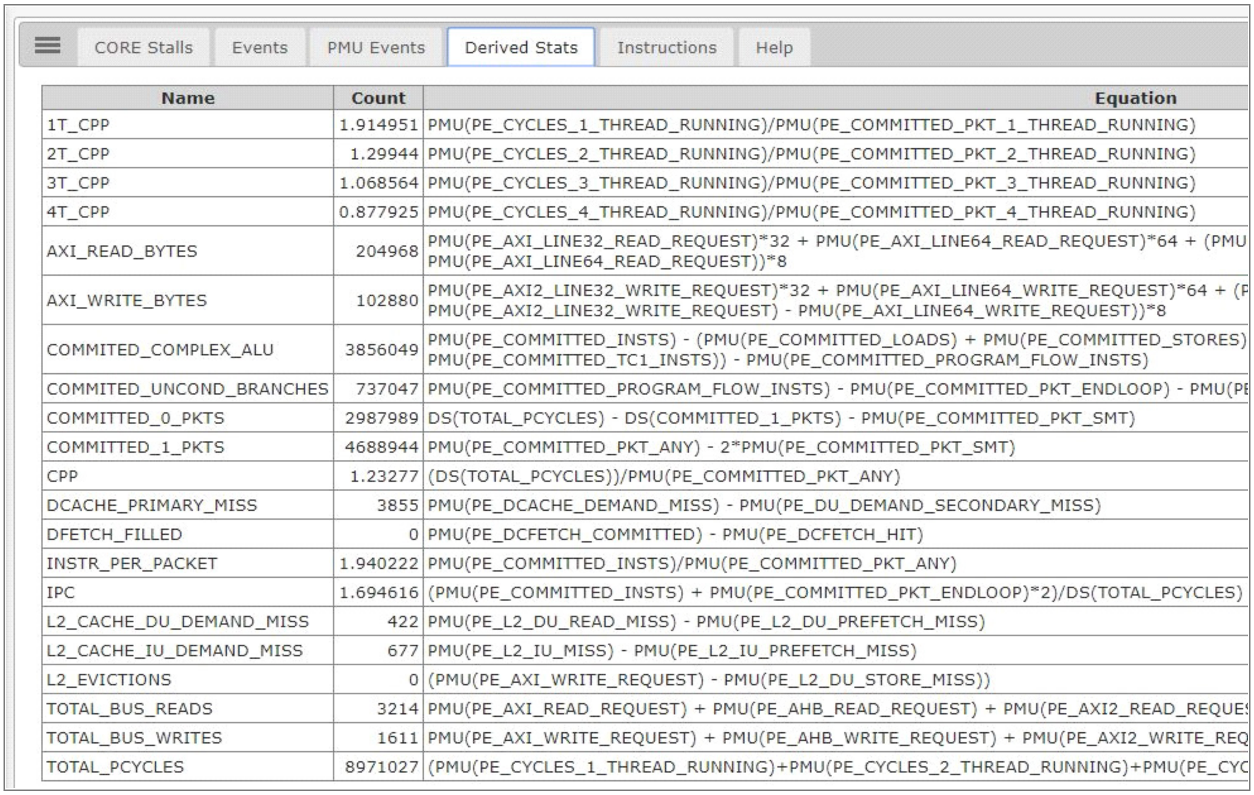
Index	Name	Count	Count Per-packet
0x1	COUNTER0_OVERFLOW	0	0.0
0x2	COUNTER2_OVERFLOW	0	0.0
0x3	COMMITTED_PKT_ANY	7277132	1.0
0x4	COMMITTED_PKT_BSB	1755439	0.241227
0x5	COUNTER4_OVERFLOW	0	0.0
0x6	COUNTER6_OVERFLOW	0	0.0
0x7	COMMITTED_PKT_B2B	1376140	0.189105
0x8	COMMITTED_PKT_SMT	1294094	0.17783
0x9	IU_CREDIT_FAIL	0	0.0
0xa	CYCLES_5_THREAD_RUNNING	0	0.0
0xb	CYCLES_6_THREAD_RUNNING	0	0.0
0xc	COMMITTED_PKT_T0	3690772	0.507174
0xd	COMMITTED_PKT_T1	1384319	0.190229
0xe	COMMITTED_PKT_T2	1453399	0.199721
0xf	COMMITTED_PKT_T3	748642	0.102876
0x12	ICACHE_DEMAND_MISS	132	1.8e-05
0x13	DCACHE_DEMAND_MISS	3876	0.000533
0x14	DCACHE_STORE_MISS	83698	0.011502
0x17	CU_PKT_READY_NOT_DISPATCHED	703714	0.096702
0x1c	IU_L1S_ACCESS	0	0.0
0x1d	IU_L1S_PREFETCH	0	0.0
0x1e	IU_L1S_AXIS_STALL	0	0.0
0x1f	IU_L1S_NO_GRANT	0	0.0
0x20	ANY_IU_REPLAY	56588	0.007776
0x21	ANY_DU_REPLAY	415	5.7e-05
0x23	ISSUED_PACKETS	7924991	1.089027
0x24	LOOPCACHE_PACKETS	0	0.0
0x25	COMMITTED_PKT_1_THREAD_RUNNING	2033736	0.279469
0x26	COMMITTED_PKT_2_THREAD_RUNNING	153757	0.021129
0x27	COMMITTED_PKT_3_THREAD_RUNNING	2142290	0.294387
0x2a	COMMITTED_INSTS	14119253	1.940222
0x2b	COMMITTED_TC1_INSTS	5907975	0.811855
0x2c	COMMITTED_PRIVATE_INSTS	7278886	1.000241
0x2f	COMMITTED_PKT_4_THREAD_RUNNING	2947348	0.405015

Figure 3-11 PMU events tab

3.5 Derived Stats tab

The Derived Stats tab shows a set of calculated values based on collected PMU events.

NOTE: Derived Stats information is presented only for Hexagon V66 and later versions.



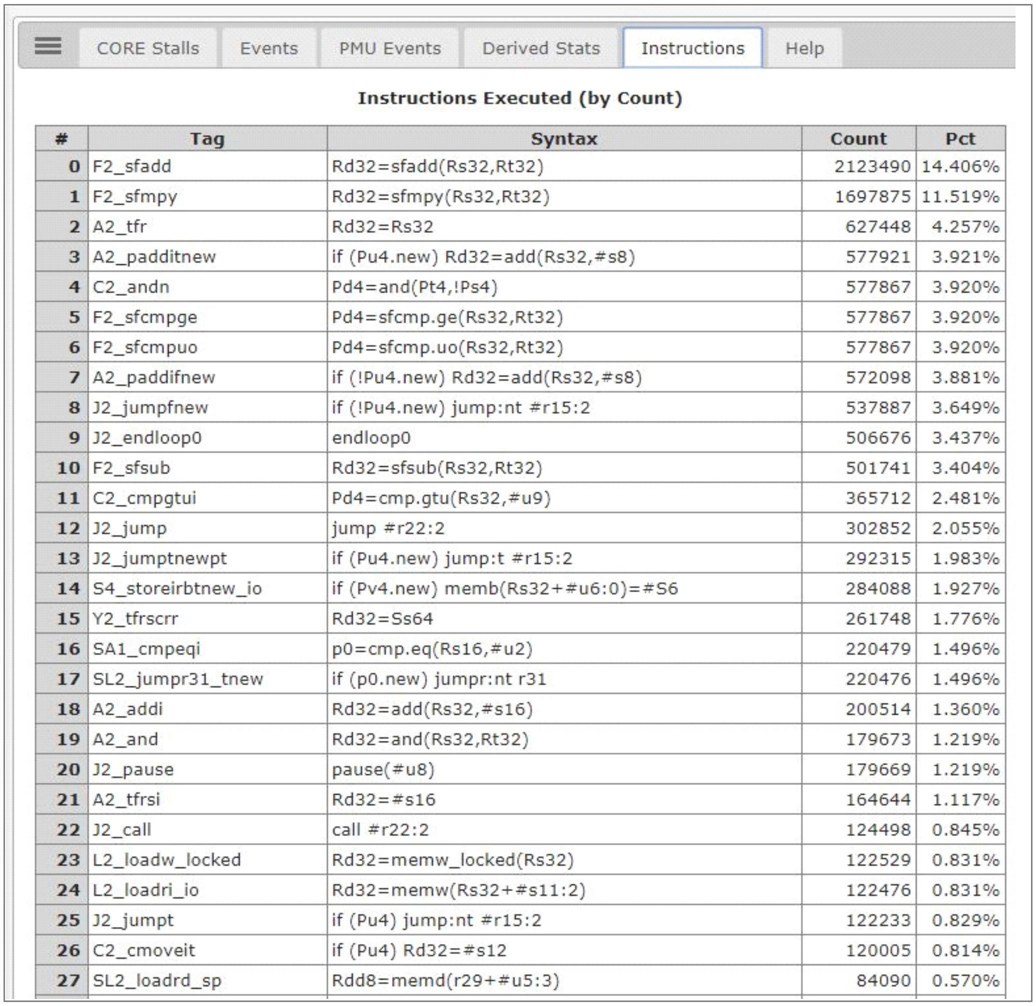
Name	Count	Equation
1T_CPP	1.914951	PMU(PE_CYCLES_1_THREAD_RUNNING)/PMU(PE_COMMITTED_PKT_1_THREAD_RUNNING)
2T_CPP	1.29944	PMU(PE_CYCLES_2_THREAD_RUNNING)/PMU(PE_COMMITTED_PKT_2_THREAD_RUNNING)
3T_CPP	1.068564	PMU(PE_CYCLES_3_THREAD_RUNNING)/PMU(PE_COMMITTED_PKT_3_THREAD_RUNNING)
4T_CPP	0.877925	PMU(PE_CYCLES_4_THREAD_RUNNING)/PMU(PE_COMMITTED_PKT_4_THREAD_RUNNING)
AXI_READ_BYTES	204968	PMU(PE_AXI_LINE32_READ_REQUEST)*32 + PMU(PE_AXI_LINE64_READ_REQUEST)*64 + (PMU PMU(PE_AXI_LINE64_READ_REQUEST))*8
AXI_WRITE_BYTES	102880	PMU(PE_AXI2_LINE32_WRITE_REQUEST)*32 + PMU(PE_AXI_LINE64_WRITE_REQUEST)*64 + (P PMU(PE_AXI2_LINE32_WRITE_REQUEST) - PMU(PE_AXI_LINE64_WRITE_REQUEST))*8
COMMITED_COMPLEX_ALU	3856049	PMU(PE_COMMITTED_INSTS) - (PMU(PE_COMMITTED_LOADS) + PMU(PE_COMMITTED_STORES) PMU(PE_COMMITTED_TC1_INSTS)) - PMU(PE_COMMITTED_PROGRAM_FLOW_INSTS)
COMMITED_UNCOND_BRANCHES	737047	PMU(PE_COMMITTED_PROGRAM_FLOW_INSTS) - PMU(PE_COMMITTED_PKT_ENDLOOP) - PMU(PE
COMMITTED_0_PKTS	2987989	DS(TOTAL_PCYCLES) - DS(COMMITTED_1_PKTS) - PMU(PE_COMMITTED_PKT_SMT)
COMMITTED_1_PKTS	4688944	PMU(PE_COMMITTED_PKT_ANY) - 2*PMU(PE_COMMITTED_PKT_SMT)
CPP	1.23277	(DS(TOTAL_PCYCLES))/PMU(PE_COMMITTED_PKT_ANY)
DCACHE_PRIMARY_MISS	3855	PMU(PE_DCACHE_DEMAND_MISS) - PMU(PE_DU_DEMAND_SECONDARY_MISS)
DFETCH_FILLED	0	PMU(PE_DCFETCH_COMMITTED) - PMU(PE_DCFETCH_HIT)
INSTR_PER_PACKET	1.940222	PMU(PE_COMMITTED_INSTS)/PMU(PE_COMMITTED_PKT_ANY)
IPC	1.694616	(PMU(PE_COMMITTED_INSTS) + PMU(PE_COMMITTED_PKT_ENDLOOP)*2)/DS(TOTAL_PCYCLES)
L2_CACHE_DU_DEMAND_MISS	422	PMU(PE_L2_DU_READ_MISS) - PMU(PE_L2_DU_PREFETCH_MISS)
L2_CACHE_IU_DEMAND_MISS	677	PMU(PE_L2_IU_MISS) - PMU(PE_L2_IU_PREFETCH_MISS)
L2_EVICTIONS	0	(PMU(PE_AXI_WRITE_REQUEST) - PMU(PE_L2_DU_STORE_MISS))
TOTAL_BUS_READS	3214	PMU(PE_AXI_READ_REQUEST) + PMU(PE_AHB_READ_REQUEST) + PMU(PE_AXI2_READ_REQUE
TOTAL_BUS_WRITES	1611	PMU(PE_AXI_WRITE_REQUEST) + PMU(PE_AHB_WRITE_REQUEST) + PMU(PE_AXI2_WRITE_REQ
TOTAL_PCYCLES	8971027	(PMU(PE_CYCLES_1_THREAD_RUNNING)+PMU(PE_CYCLES_2_THREAD_RUNNING)+PMU(PE_CYC

Figure 3-12 Derived Stats tab

3.6 Instructions tab

The Instructions tab shows the Hexagon assembly instructions that were executed, the syntax of each assembly instruction, the count of each instruction, and the percentage of each instruction.

The Hexagon assembly instructions are described in the *Qualcomm Hexagon Programmer's Reference Manual*.



#	Tag	Syntax	Count	Pct
0	F2_sfadd	Rd32=sfadd(Rs32,Rt32)	2123490	14.406%
1	F2_sfmpy	Rd32=sfmpy(Rs32,Rt32)	1697875	11.519%
2	A2_tfr	Rd32=Rs32	627448	4.257%
3	A2_padditnew	if (Pu4.new) Rd32=add(Rs32,#s8)	577921	3.921%
4	C2_andn	Pd4=and(Pt4,!Ps4)	577867	3.920%
5	F2_sfcmpge	Pd4=sfcmp.ge(Rs32,Rt32)	577867	3.920%
6	F2_sfcmpuo	Pd4=sfcmp.uo(Rs32,Rt32)	577867	3.920%
7	A2_paddifnew	if (!Pu4.new) Rd32=add(Rs32,#s8)	572098	3.881%
8	J2_jumpfnew	if (!Pu4.new) jump:nt #r15:2	537887	3.649%
9	J2_endloop0	endloop0	506676	3.437%
10	F2_sfsub	Rd32=sfsub(Rs32,Rt32)	501741	3.404%
11	C2_cmpgtui	Pd4=cmp.gtu(Rs32,#u9)	365712	2.481%
12	J2_jump	jump #r22:2	302852	2.055%
13	J2_jumptnewpt	if (Pu4.new) jump:t #r15:2	292315	1.983%
14	S4_storeirbtnew_io	if (Pv4.new) memb(Rs32+#u6:0)=#S6	284088	1.927%
15	Y2_tfrscrr	Rd32=Ss64	261748	1.776%
16	SA1_cmpeqi	p0=cmp.eq(Rs16,#u2)	220479	1.496%
17	SL2_jumpr31_tnew	if (p0.new) jumpr:nt r31	220476	1.496%
18	A2_addi	Rd32=add(Rs32,#s16)	200514	1.360%
19	A2_and	Rd32=and(Rs32,Rt32)	179673	1.219%
20	J2_pause	pause(#u8)	179669	1.219%
21	A2_tfrsi	Rd32=#s16	164644	1.117%
22	J2_call	call #r22:2	124498	0.845%
23	L2_loadw_locked	Rd32=memw_locked(Rs32)	122529	0.831%
24	L2_loadri_io	Rd32=memw(Rs32+#s11:2)	122476	0.831%
25	J2_jumpt	if (Pu4) jump:nt #r15:2	122233	0.829%
26	C2_cmoveit	if (Pu4) Rd32=#s12	120005	0.814%
27	SL2_loadrd_sp	Rdd8=memd(r29+#u5:3)	84090	0.570%

Figure 3-13 Instructions tab

3.7 Help tab

The Help tab shows information about the test parameters.

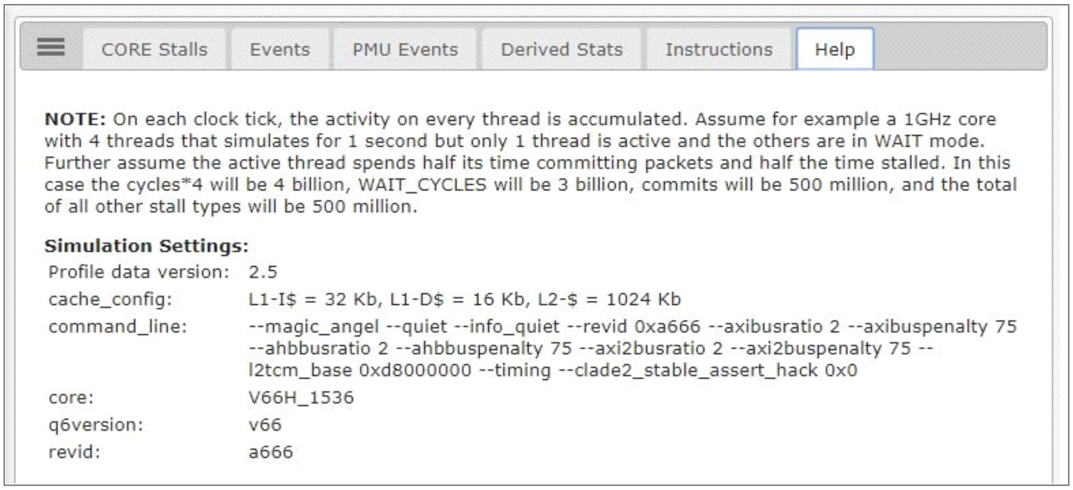


Figure 3-14 Help tab

3.8 Search grid box

The search grid box is present when the CORE Stalls or Events tab is active. As you type in the search grid, the profile display scrolls to the function (symbol) that matches what you type.

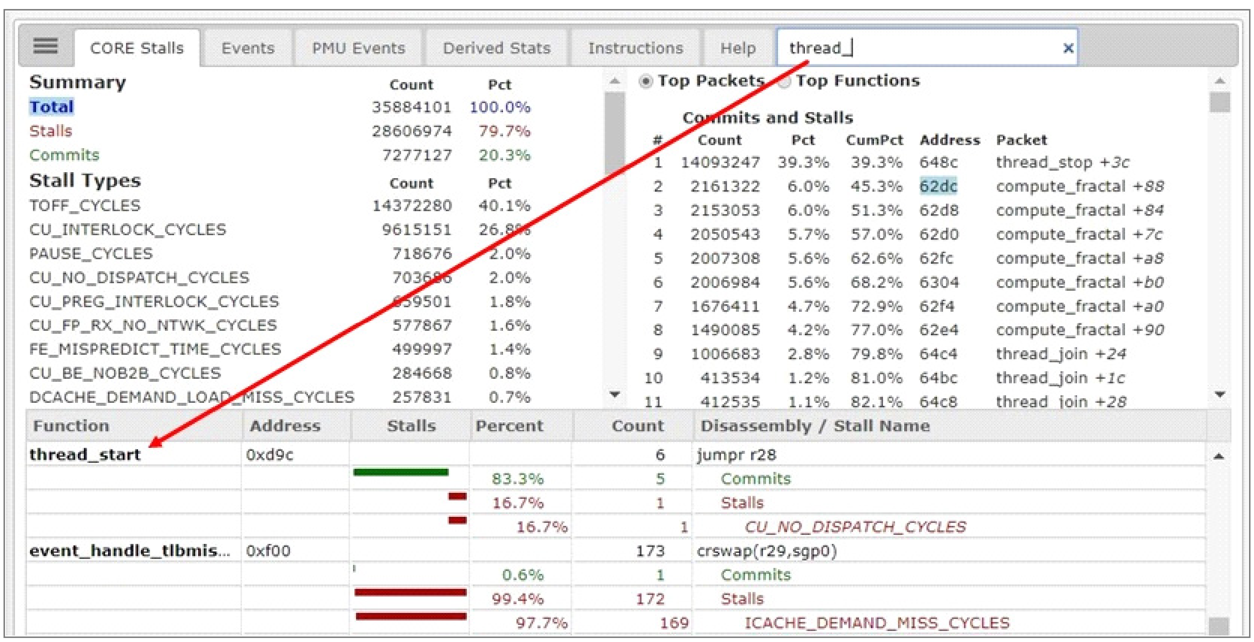


Figure 3-15 Search grid box

3.9 Optional HVX Stalls tab

If the Hexagon ELF file contained HVX instructions, the HVX Stalls tab appears.

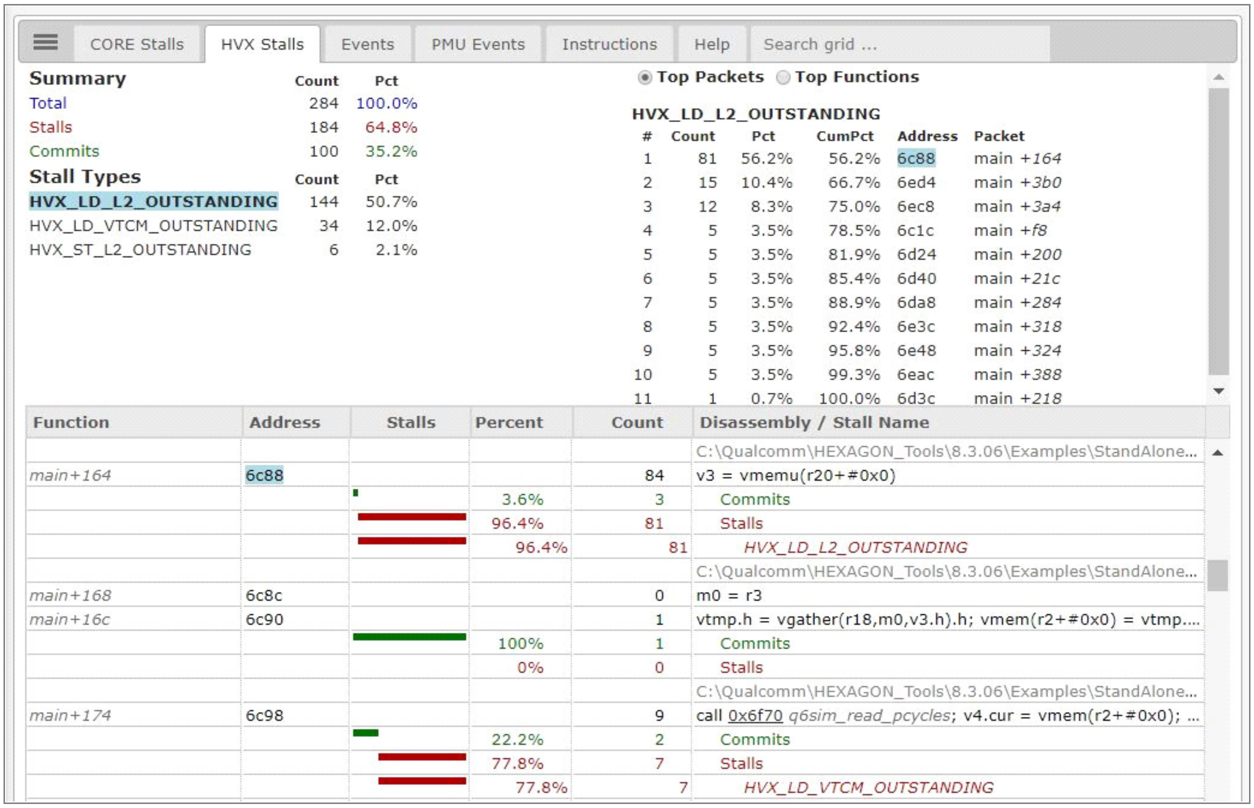


Figure 3-16 HVX Stalls tab (optional)

This tab displays the cycle counts and cycle count statistics for HVX instruction packets.

- The top left pane lists the total HVX cycles and HVX stall cycles for the program, followed by a list of the HVX stall cycles categorized by stall type.
- The top right pane initially contains a brief example that clarifies how HVX cycle commits and HVX stall cycles are defined.

Clicking a hyperlinked item in the top left pane changes the top right pane to lists the instruction packets with the highest cycle counts for the selected item. Clicking a PC address in the top right pane changes the bottom pane to display a disassembled listing of the program code, starting at the specified instruction.

If you select Commits/Stalls on the Disassembly Panel, the commit and stall percentages are represented graphically, with green lines indicating the commit percentages and red lines indicating the stall percentages.