

# Qualcomm<sup>®</sup> Hexagon<sup>™</sup> Profiler

**User Guide** 

80-N2040-10 Rev. E

May 23, 2022

All Qualcomm products mentioned herein are products of Qualcomm Technologies, Inc. and/or its subsidiaries.

Qualcomm and Hexagon are trademarks or registered trademarks of Qualcomm Incorporated. Other product and brand names may be trademarks or registered trademarks of their respective owners.

This technical data may be subject to U.S. and international export, re-export, or transfer ("export") laws. Diversion contrary to U.S. and international law is strictly prohibited.

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

© 2016-2017, 2019, 2022 Qualcomm Technologies, Inc. and/or its subsidiaries. All rights reserved.

# Contents

1 Introduction	4
1.1 Conventions	4
1.2 Technical assistance	4
2 Use the profiler	
	-
2.1 Create the profile data file	5
2.2 Run the Hexagon profiler	b
2.2.1 Command	б
2.2.2 Options	б ~
2.2.3 Example	
2.3 View the profile report file	8
3 Hexagon profiler user interface	
3.1 User controls	10
<ul><li>3.1 User controls</li><li>3.1.1 Profile information tabs</li></ul>	
<ul> <li>3.1 User controls</li></ul>	10 10 
<ul> <li>3.1 User controls</li></ul>	

# Figures

Figure 2-1 Using the Hexagon profiler	5
Figure 3-1 Hexagon profiler user interface	9
Figure 3-2 Tabs	10
Figure 3-3 Menu icon	11
Figure 3-4 Disassembly Panel configuration options	11
Figure 3-5 CORE Stalls tab	13
Figure 3-6 CORE Stalls tab: total commits and stalls	14
Figure 3-7 CORE Stalls tab: commits and stalls by stall type	15
Figure 3-8 CORE Stalls tab: display stall details	16
Figure 3-9 CORE Stalls tab: display minor stalls	17
Figure 3-10 Events tab	18
Figure 3-11 PMU events tab	19
Figure 3-12 Derived Stats tab	20
Figure 3-13 Instructions tab	21
Figure 3-14 Help tab	22
Figure 3-15 Search grid box	22
Figure 3-16 HVX Stalls tab (optional)	23

# **1** Introduction

This document describes the Qualcomm<sup>®</sup> Hexagon<sup>™</sup> 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.

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*.

CORE Stalls	Events	PMU Events	Derived Stats	Instr	uctions	Help	Searc	h grid			
Summary		Cour	nt Pct		🖲 Тор	Packets	🔘 Тор	Function	15		
Total		35884	101 100.0%			ommite	and Sta	llc			
Stalls		28606	974 79.7%	- 88	#	Count	Pct	CumPct	Address	Packet	
Commits		7277	127 20.3%		1 1	4093247	39.3%	39.3%	648c	thread stop +3c	
Stall Types		Cour	nt Pct		2	2161322	6.0%	45.3%	62dc	compute fractal +88	
TOFF_CYCLES		14372	280 40.1%		3	2153053	6.0%	51.3%	62d8	compute fractal +84	
CU_INTERLOCK_CYCI	LES	9615	151 26.8%		4	2050543	5.7%	57.0%	62d0	compute fractal +7c	
PAUSE_CYCLES		718	676 2.0%		5	2007308	5.6%	62.6%	62fc	compute fractal +a8	
CU_NO_DISPATCH_C	YCLES	703	686 2.0%		6	2006984	5.6%	68.2%	6304	compute fractal +b0	
CU_PREG_INTERLOC	K_CYCLES	659	501 1.8%		7	1676411	4.7%	72.9%	62f4	compute fractal +a0	
CU_FP_RX_NO_NTW	K_CYCLES	577	867 1.6%		8	1490085	4.2%	77.0%	62e4	compute fractal +90	
FE_MISPREDICT_TIM	E_CYCLES	499	997 1.4%		9	1006683	2.8%	79.8%	64c4	thread join +24	
CU_BE_NOB2B_CYCL	ES	284	668 0.8%		10	413534	1.2%	81.0%	64bc	thread join +1c	
DCACHE_DEMAND_LC	OAD_MISS_	CYCLES 257	831 0.7%	•	11	412535	1.1%	82.1%	64c8	thread join +28	
Function	Addre	ess Stal	ls Percent	Co	ount	Disasse	mbly /	Stall Na	ne		
_start	0x0			279	9063	jump 0x9	8 qdsp6	_start_in	it		
_start+4	0x4				0	jump <u>0x8</u>	30 zebu_	mutex+0	x40		
_start+8	0x8				0	jump <u>0x8</u>	Bc zebu_	mutex+0	x4c		
hexagon_start_init	0x98				0	r0 = #0×	0; r1 =	#0×0			
hexagon_start_init+4	0x9c				0	r3:2 = co	ombine(r	1,r0); r5	4 = comi	bine(r1,r0); r7:6 = combi	
hexagon_start_init+1	4 Oxac				0	r11:10 =	combin	e(r1,r0);	r13:12 =	combine(r1,r0); r15:14	
hexagon_start_init+2	24 Oxbc				0	r19:18 =	combin	e(r1,r0);	r21:20 =	combine(r1,r0); r23:22 .	
hexagon_start_init+3	4 Oxcc				0	r27:26 =	combin	e(r1,r0);	r29:28 =	combine(r1,r0); r31:30	

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:

CORE Stalls	Events	PMU Events	Derived Stats	Instructions	Help	Search grid
-------------	--------	------------	---------------	--------------	------	-------------

### Figure 3-2 Tabs

Tab	Profile Information	See
CORE Stalls	Total number of cycles and stalls executed	Section 3.2
	Highest cycle or stall counts (by function or instruction packet)	
	Commit and stall statistics (by function or instruction packet)	
	A Search grid tab appears at the far right of the tabs	Section 3.8
Events	Event count statistics (by function or instruction package)	Section 3.3
	Disassembled listing of instruction packets	
	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.

8.2.xx	> PMU Events	5 De	rived Stats	Instr	uctions	Help	Searc	h grid			
Dicaccombly Banal	C4 358	ount 84101	Pct 100.0%	^	🖲 Тор	Packets	ОТор	Function	5		Â
Visible Lines Commits Stalls Stall Details Hinor Stalls Events Filenames	286 72 143 96 7 7 6 5	06974 77127 ount 72280 15151 18676 03686 59501 77867	79.7% 20.3% Pct 40.1% 26.8% 2.0% 2.0% 1.8% 1.6%		# 1 1 2 3 4 5 6 7 8	Count 4093247 2161322 2153053 2050543 2007308 2006984 1676411 1490085	Pct 39.3% 6.0% 6.0% 5.7% 5.6% 5.6% 4.7% 4.2%	CumPct 39.3% 45.3% 51.3% 57.0% 62.6% 68.2% 72.9% 77.0%	Address 648c 62dc 62d8 62d0 62fc 6304 62f4 62e4	Packet thread_stop +3c compute_fractal +88 compute_fractal +84 compute_fractal +7c compute_fractal +a8 compute_fractal +b0 compute_fractal +a0 compute_fractal +90	
Visible Attributes	4 2 (CLES 2 .5 St	84668 57831 talls	0.8% 0.7% Percent	- Ce	9 10 11 ount	1006683 413534 412535 Disasse	2.8% 1.2% 1.1% mbly /	79.8% 81.0% 82.1% Stall Nar	64c4 64bc 64c8 ne	thread_join +24 thread_join +1c thread join +28	1
Packet Braces Leading Zeros Hex Prefix 0x			0% 100% 100%	27	0 0 9063 279033	Comm Stalls	eqdsp6	_start_in ES	it		
Underline links					0 -0 0	jump <u>0x8</u> jump <u>0x8</u> r0 = #0x r3:2 = cc	30 zebu_ 3c zebu_ 30; r1 =	mutex+0 mutex+0 #0x0 1.r0): r5:	x40 x4c	$pine(r1,r0): r7:6 = combi_{r0}$	
Global Options	1				0 0 0	r11:10 = r19:18 = r27:26 =	combine combine combine	e(r1,r0); e(r1,r0); e(r1,r0);	r13:12 = r21:20 = r29:28 =	combine(r1,r0); r15:14 combine(r1,r0); r23:22 combine(r1,r0); r31:30	
magnitier					0	r0 = rev	(r0 #0vf	£)			

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 <i>minor stalls</i> ).
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 <b>0x</b> 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

CORE Stalls	Events PM	U Events	Derived Stats	Instruct	ions	Help	Searc	h grid			
Summary Total		Coun 358841	t Pct .01 100.0%	* ®	Тор	Packets	Top and Stal	Function	5		
Stalls Commits Stall Types TOFF_CYCLES CU_INTERLOCK_CYCLES CU_NO_DISPATCH_CYC CU_PREG_INTERLOCK_ CU_PP_RX_NO_NTWK_ CU_PP_RX_NO_NTWK_ CU_BE_NOB2B_CYCLES	S CYCLES CYCLES CYCLES CYCLES	286069 72771 <b>Coun</b> 143722 96151 7186 7036 6595 5778 4999 2846	774         79.7%           27         20.3%           t         Pct           180         40.1%           51         26.8%           676         2.0%           686         2.0%           601         1.8%           667         1.6%           997         1.4%           668         0.8%		# 1 1 2 3 4 5 6 7 8 9 0	Count 4093247 2161322 2153053 2050543 2007308 2006984 1676411 1490085 1006683 413534	Pct 39.3% 6.0% 5.7% 5.6% 5.6% 4.7% 4.2% 2.8% 1.2%	CumPct 39.3% 45.3% 51.3% 57.0% 62.6% 68.2% 72.9% 77.0% 79.8% 81.0%	Address 648c 62dc 62d8 62d0 62fc 6304 62f4 62e4 64c4 64bc	Packet thread_stop +3c compute_fractal +88 compute_fractal +84 compute_fractal +7c compute_fractal +a8 compute_fractal +b0 compute_fractal +00 compute_fractal +90 thread_join +24 thread_ioin +1c	
DCACHE_DEMAND_LOA	D_MISS_CYCL	ES 2578	31 0.7%	• 1	1	412535	1.1%	82.1%	64c8	thread join +28	•
Function	Address	Stall	s Percent	Cour	It	Disasse	mbly /	Stall Nar	ne		
_start	0x0			27900	0	jump <u>0x98</u> qdsp6_start_init					
_Start+9	029				0	jump 0x80 zebu_mutex+0x40					
hexagon start init	0x98				0	$r0 = \pm 0x$	0: r1 =	#0x0	A 70		
hexagon start init+4	0x9c				0	$r_{3:2} = c_{1}$	ombine(r	1.r0): r5	4 = comb	bine(r1.r0): r7:6 = combi	
hexagon start init+14	0xac				0	r11:10 =	combine	e(r1,r0);	r13:12 =	combine(r1,r0); r15:14	
hexagon_start_init+24	0xbc				0	r19:18 =	combine	e(r1,r0);	r21:20 =	combine(r1,r0); r23:22	
hexagon_start_init+34	0xcc				0	r27:26 =	combine	e(r1,r0);	r29:28 =	combine(r1,r0); r31:30	
hexagon_start_init+40	0xd8				0	r0 = rev					
hexagon_start_init+44	0xdc				0	r0 = and	(r0,#0xf	f)			
hexagon_start_init+48	0xe0				0	p0 = cm	p.eq(r0,#	≠0x2)			
hexagon_start_init+4c	0xe4				0	if (!p0) j	ump:nt 0	xf4 .Init(	ChickenCu	istom	

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

CORE Stalls	Events	PMU Events	Derived Stats	Instru	uctions	Help	Searc	h grid			
Summary		Count	Pct		⊛ To	p Packets	🔘 Тор	Function	s		
Total		3588410	1 100.0%			Commite	and Stal	le			
Stalls		2860697	4 79.7%		#	Count	Pct	CumPct	Address	Packet	
Commits		727712	7 20.3%		1	14093247	39.3%	39.3%	648c	thread stop +3c	
Stall Types		Count	Pct		2	2161322	6.0%	45.3%	62dc	compute fractal +88	
TOFF_CYCLES		1437228	40.1%		3	2153053	6.0%	51.3%	62d8	compute_fractal +84	
CU_INTERLOCK_CYCLE	S	961515	1 26.8%		4	2050543	5.7%	57.0%	62d0	compute_fractal +7c	
PAUSE_CYCLES		71867	6 2.0%		5	2007308	5.6%	62.6%	62fc	compute_fractal +a8	
CU_NO_DISPATCH_CY	CLES	70368	6 2.0%		6	2006984	5.6%	68.2%	6304	compute_fractal +b0	
CU_PREG_INTERLOCK_	_CYCLES	65950	1 1.8%		7	1676411	4.7%	72.9%	62f4	compute_fractal +a0	
CU_FP_RX_NO_NTWK_	CYCLES	57786	1.6%		8	1490085	4.2%	77.0%	62e4	compute_fractal +90	
FE_MISPREDICT_TIME	_CYCLES	49999	1.4%		9	1006683	2.8%	79.8%	64c4	thread_join +24	
CU_BE_NOB2B_CYCLE	S	28466	8 0.8%		10	413534	1.2%	81.0%	64bc	thread_join +1c	
DCACHE_DEMAND_LOA	AD_MISS_CY	CLES 25783	1 0.7%	•	11	412535	1.1%	82.1%	64c8	thread join +28	
Function	Addres	s Stalls	Percent	Co	ount	Disasse	mbly /	Stall Nar	ne		
						C:\Qualc	omm\HE	XAGON_	Tools\8.3.	06\Examples\Profiling\pr	
compute_fractal+88	0x62dc			2161	1322	p1 = sfcr	np.uo(r5	,r10); p2	= sfcmp.	.ge(r5,r10)	
compute_fractal+90	0x62e4			1490	0085	p1 = and	l(p2,!p1)	; if (p1.n	ew) r13 =	add(r13,#0x1); if (!p1	
						C:\Qualc	omm\HE	XAGON_	Tools\8.3.	06\Examples\Profiling\pr	
compute_fractal+a0	0x62f4			1676	5411	r14 = sfr	mpy(r14,	r28); r28	= sfsub(	r0,r1)	
compute_fractal+a8	0x62fc			2007	7308	r14 = sfa	add(r14,r	14); r0 =	sfadd(r1	5,r28)	
compute_fractal+b0	0x6304			2006	5984	r14 = sfa	add(r12,r	14); r28	= r0 :end	lloop0	

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 <code>compute\_fractal()</code> 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

CORE Stalls	Events PM	IU Events	Derived Stats	Instructions	Help	Search grid
Summary		Count	Pct	▲ () To	Packets	Top Functions
Total		3588410	1 100.0%		DALISE C	VOLES
Stalls		28606974	4 79.7%	# 6	PAUSE_C	t CumPct Address Packet
Commits		7277123	7 20.3%	1 71	B676 100.	0% 100.0% 64c4 thread join +24
Stall Types		Count	Pct			
TOFF_CYCLES		14372280	40.1%			
CU_INTERLOCK_CYCL	ES	961515:	1 26.8%			
PAUSE_CYCLES		718676	5 2.0%			
CU_NO_DISPATCH_CY	CLES	703686	5 2.0%			
CU_PREG_INTERLOCK	_CYCLES	65950:	1 1.8%			
CU_FP_RX_NO_NTWK	CYCLES	57786	7 1.6%			
FE_MISPREDICT_TIME	CYCLES	499993	7 1.4%			
CU_BE_NOB2B_CYCLE	S	284668	B 0.8%			
DCACHE_DEMAND_LO	AD_MISS_CYCL	ES 25783	1 0.7%	-		
Function	Address	Stalls	Percent	Count	Disasser	mbly / Stall Name
thread_join+24	0x64c4			1006683	pause(#0	x1)
			17.8%	179669	Comm	its
			82.2%	827014	Stalls	
thread_join+28	0x64c8			412535	jump	4b8 thread_join+0x18
			43.6%	179669	Comm	its
			56.4%	232866	Stalls	
sys_get_cmdline	0x6500			180	immext(#	#0x9400); r2 = memw(##0x9420); if (cmp.eq(r2.n
		1	0.6%	1	Comm	its
			99.4%	179	Stalls	
sys_get_cmdline+c	0x650c			22	memd(r2	9+#-0x10) = r17:16; allocframe(#0x10)
		-	9.1%	2	Comm	its
			90.9%	20	Stalls	
sys_get_cmdline+1	0 0x6510			2	call 0x710	00 hexagon_cache_cleaninv; memw(r29+#0x0) = r
			50%	1	Comm	its
			50%	1	Stalls	
sys aet cmdline+1	8 0x6518			1	call 0x710	00 hexagon cache cleaninv; r1 = #0x8; r0 = add(r
	and the second se					
			100%	1	Comm	its

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).

- CONCE Stans			child office	anotre		incip	Gearc	in grid in		
Summary		Count	Pct		• То	p Packets	🔘 Тор	Function	15	
Total		35884101	100.0%			Commits	and Sta	lls		
Stalls		28606974	79.7%		#	Count	Pct	CumPct	Address	Packet
Commits		7277127	20.3%		1	14093247	39.3%	39.3%	648c	thread_stop +3c
Stall Types		Count	Pct		2	2161322	6.0%	45.3%	62dc	compute_fractal +88
TOFF_CYCLES	14372280	40.1%		3	2153053	6.0%	51.3%	62d8	compute_fractal +84	
CU_INTERLOCK_CYCLE	U_INTERLOCK_CYCLES 9615151				4	2050543	5.7%	57.0%	62d0	compute_fractal +7c
PAUSE_CYCLES		718676	2.0%		5	2007308	5.6%	62.6%	62fc	compute_fractal +a8
CU_NO_DISPATCH_CYC	CLES	703686	2.0%		6	2006984	5.6%	68.2%	6304	compute_fractal +b0
CU_PREG_INTERLOCK_	CYCLES	659501	1.8%		7	1676411	4.7%	72.9%	62f4	compute_fractal +a0
CU_FP_RX_NO_NTWK_	CYCLES	577867	1.6%		8	1490085	4.2%	77.0%	62e4	compute_fractal +90
FE_MISPREDICT_TIME_	CYCLES	499997	1.4%		9	1006683	2.8%	79.8%	64c4	thread_join +24
CU_BE_NOB2B_CYCLES	5	284668	0.8%		10	<mark>4</mark> 13534	1.2%	81.0%	64bc	thread_join +1c
DCACHE_DEMAND_LOAD_MISS_C		S 257831	0.7%	-	11	412535	1.1%	82.1%	64c8	thread join +28
Function	Address	Stalls	Percent	Co	unt	Disasse	mbly /	Stall Na	ne	
						C:\Qualc	omm\HE	XAGON_	Tools\8.3.	06\Examples\Profiling\pr
ompute_fractal+88	0x62dc			2161	322	p1 = sfcr	mp.uo(r5	5,r10); p2	= sfcmp.	.ge(r5,r10)
			24.9%	537867		Comm	nits			
			75.1%	1623455		Stalls				
			49.8%	% 1075734		4 CU_INTERLOCK_CYCLES				
			24.9%	537867		7 CU_FP_RX_NO_NTWK_CYCLES				
compute_fractal+90	0x62e4			1490	085	p1 = and(p2,!p1); if (p1.new) r13 = add(r13,#0x1); if (!p1				
			36.1%	537	867	Commits				
			63.9%	952	218	Stalls				
			36.1%		53786	6 <i>CU</i>	_PREG_1	INTERLOC	CK_CYCLE	S
		-	26.1%		38844	1 FE_	MISPRE	DICT_TIM	1E_CYCLE	S
			1.7%		2590	4 <i>CU</i>	_NO_DI	SPATCH_	CYCLES	
						C:\Qualc	omm\HE	XAGON_	Tools\8.3.	06\Examples\Profiling\pr.
compute_fractal+a0	0x62f4			1676	6411	r14 = sfr	mpy(r14,	r28); r28	= sfsub(	r0,r1)
			29.9%	501	.741	Comm	nits			
			70.1%	1174	670	Stalls				
			59.5%		99802	9 CU	_INTERL	OCK_CYC	CLES	
			10.5%		17649	2 <i>CU</i>	_NO_DI	SPATCH_	CYCLES	
compute_fractal+a8	0x62fc			2007	308	r14 = sfa	add(r14,i	r14); r0 =	sfadd(r1	5,r28)
			25%	501	741	Comm	nits			

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).

CORE Stalls	Events PMU	Events [	Derived Stats	Instr	uctions	Help	Searc	h grid															
Summary		Count	Pct	*	🖲 Тор	Packets	🔘 Тор	Function	5		3												
Total		35884101	100.0%			Commite	and Sta	lle			- 8												
Stalls		28606974	79.7%		#	Count	Pct	CumPct	Address	Packet													
Commits		7277127	20.3%		1 1	4093247	39.3%	39.3%	648c	thread stop +3c													
Stall Types		Count	Pct		2	2161322	6.0%	45.3%	62dc	compute fractal +88													
TOFF_CYCLES		14372280	40.1%		3	2153053	6.0%	51.3%	62d8	compute fractal +84													
CU_INTERLOCK_CYCLE	S	9615151	26.8%		4	2050543	5.7%	57.0%	62d0	compute fractal $+7c$													
PAUSE_CYCLES		718676	2.0%		5	2007308	5.6%	62.6%	62fc	compute fractal +a8													
CU_NO_DISPATCH_CYC	LES	703686	2.0%		6	2006984	5.6%	68.2%	6304	compute fractal +b0													
CU_PREG_INTERLOCK_	CYCLES	659501	1.8%		7	1676411	4.7%	72.9%	62f4	compute fractal +a0													
CU_FP_RX_NO_NTWK_	CYCLES	577867	1.6%		8	1490085	4.2%	77.0%	62e4	compute fractal +90													
FE_MISPREDICT_TIME_	CYCLES	499997	1.4%		9	1006683	2.8%	79.8%	64c4	thread join +24													
CU_BE_NOB2B_CYCLES	5	284668	0.8%		10	413534	1.2%	81.0%	64bc	thread join +1c													
DCACHE_DEMAND_LOA	D_MISS_CYCLES	257831	0.7%	-	11	412535	1.1%	82.1%	64c8	thread join +28	100												
Function	Address	Stalls	Percent	C	ount	Disasse	mbly /	Stall Nar	ne														
						C:\Qualc	omm\HE	XAGON_	Tools\8.3.	06\Examples\Profiling\pr.													
compute_fractal+88	0x62dc			216	1322	p1 = sfc	mp.uo(r5	5,r10); p2	= sfcmp.	.ge(r5,r10)													
			24.9%	537867		Comm	nits																
			75.1%	1623455		Stalls																	
		_	49.8%	1	075734	CU	_INTERL	OCK_CYC	LES														
												-					537867	CU	_FP_RX_	NO_NTW	K_CYCLE	S	
			0.5%		9851	. CU	_NO_DI	SPATCH_(	CYCLES														
			0%		3	CU	_QOS_N	ODISPAT	CH_CYCLI	ES													
compute_fractal+90	0x62e4			149	0085	p1 = and	(p2,!p1)	; if (p1.n	ew) r13 =	= add(r13,#0x1); if (!p1	•												
			36.1%	53	7867	Comm	nits																
			63.9%	95	2218	Stalls																	
			36.1%		537866	CU	_PREG_I	INTERLOC	K_CYCLE	S													
		-	26.1%		388441	FE_	MISPRE	DICT_TIM	ME_CYCLE	S													
			1.7%		25904	CU	_NO_DI	SPATCH_	CYCLES														
			0%		7	CU	_QOS_N	ODISPAT	CH_CYCLI	ES													
						C:\Qualc	omm\HE	XAGON_	Tools\8.3.	06\Examples\Profiling\pr.													
compute_fractal+a0	0x62f4			167	6411	r14 = sfr	npy(r14,	r28); r28	= sfsub(	r0,r1)													
			29.9%	50	1741	Comm	nits																
			70.1%	117	4670	Stalls																	
			59.5%		998029	CU	_INTERL	OCK_CYC	LES														
			10.5%		176492	cu	NO_DI	SPATCH_	CYCLES														
			0%		130	FE_	PICK_O	THER															
			0%		19	CU	QOS N	ODISPAT	CH_CYCLI	ES													

### 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*).

CORE Stalls	Events P	MU Events De	erived Stats	Instru	uction	s H	elp S	earch grid			
PMU Events	Received	Count		*	• To	p Pack	ets 🛛 1	op Funct	ions		
COMMITTED_PKT_ANY		7277132				1.2	ACCECC				
CYCLES_1_THREAD_RUI	NNING	3894504			-	Count	ACCESS	CumPet	Address	Packet	
COMMITTED_PKT_T0		3690772			-	81606	21 3%	21.3%	765c	Lockfilelock +0	
COMMITTED_PKT_4_TH	READ_RUNN	ING 2947348			2	80801	21.0%	42 304	7820	foutc +0	
COMMITTED_PKT_3_TH	READ_RUNN	ING 2142290			3	40818	10.6%	52 9%	6820	lockMutex + 30	
COMMITTED_PKT_1_THE	READ_RUNN	ING 2033736			4	40817	10.6%	62.6%	6840	eve Mtyuplock +1c	
COMMITTED_PKT_BSB		1755+39			5	40400	10.5%	74 196	7850	foutc + 20	
COMMITTED_PKT_T2		1453399			6	40400	10.5%	84 6%	7858	foutc +38	
COMMITTED_PKT_T1	/	1384319			7	24418	6 4%	91.0%	6390	compute fractal +148	
COMMITTED_PKT_B2B		1376140			8	8585	2 20%	93 2%	63bc	compute_fractal +168	
COMMITTED_PKT_T		748642			9	3874	1 0%	94 2%	6338	compute_fractal +e4	
L2_ACCESS		383910			10	2807	0.8%	95.0%	6f20	memset +c0	
DU_WRITE_TO_L2		376430		-	11	2569	0.7%	95.6%	620c	main +12c	
Function	Address	Event Coun	t Disassemt	oly / Ev	ent N	ame					
_start	0x0		jump 0x98	qdsp6_s	tart_i	nit					
_start+4	0x4		jump 0x80	zebu_m	utex+	0x40					
_start+8	0x8		jump 0x8c	zebu_mu	itex+	0x4c					
hexagon_start_init	0x98		r0 = #0x0;	r1 = #0	0x0						
hexagon_start_init+4	0x9c		r3:2 = com	bine(r1,	r0); r!	5:4 = c	ombine(i	1,r0); r7:	6 = comb	ine(r1,r0); r9:8 = combine	(r
hexagon_start_init+14	0xac		r11:10 = co	mbine(r	1,r0)	r13:13	2 = comb	oine(r1,r0)	; r15:14	= combine(r1,r0); r17:16 =	
hexagon_start_init+24	0xbc		r19:18 = co	mbine(r	1,r0)	r21:20	) = com	oine(r1,r0)	; r23:22	= combine(r1,r0); r25:24 =	
hexagon_start_init+34	0xcc		r27:26 = co	mbine(r	1,r0)	r29:28	B = comb	pine(r1,r0)	; r31:30	= combine(r1,r0)	
hexagon_start_init+40	0xd8		r0 = rev								
hexagon_start_init+44	0xdc		r0 = and(r0)	,#0xff)							
hexagon_start_init+48	0xe0		p0 = cmp.e	q(r0,#0	x2)						
hexagon_start_init+4c	0xe4		if (!p0) jum	p:nt Oxf	4 .Ini	Chicke	nCustom				
hexagon_start_init+50	0xe8		r0.h = #0x4	40							
hexagon_start_init+54	0xec		r0.1 = #0x0								
hexagon_start_init+58	0xf0		s60 = r0								
hexagon_start_init+5c	0xf4		r0.h = #0x0	0							

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.

E c	ORE Stalls Events PMU Events	Derived Stats	Instru	ctions	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	C.	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_RUNNIN	G	2033736		0.279469
0x26	COMMITTED_PKT_2_THREAD_RUNNIN	G	153757		0.021129
0x27	COMMITTED_PKT_3_THREAD_RUNNIN	G	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_RUNNIN	G	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.

CORE Stalls Events F	MU Events	Derived Stats Instructions Help							
Name	Count	Equation							
1T_CPP	1.914951	PMU(PE_CYCLES_1_THREAD_RUNNING)/PMU(PE_COMMITTED_PKT_1_THREAD_RUNNING)							
2T_CPP	1.29944	MU(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	MU(PE_AXI_LINE32_READ_REQUEST)*32 + PMU(PE_AXI_LINE64_READ_REQUEST)*64 + (PMU MU(PE_AXI_LINE64_READ_REQUEST))*8							
AXI_WRITE_BYTES	102880	PMU(PE_AXI2_LINE32_WRITE_REQUEST)*32 + PMU(PE_AXI_LINE64_WRITE_REQUEST)*64 + (F 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(PI							
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_REQU							
TOTAL_BUS_WRITES	1611	PMU(PE_AXI_WRITE_REQUEST) + PMU(PE_AHB_WRITE_REQUEST) + PMU(PE_AXI2_WRITE_RE							
TOTAL_PCYCLES	8971027	PMU(PE_CYCLES_1_THREAD_RUNNING)+PMU(PE_CYCLES_2_THREAD_RUNNING)+PMU(PE_CY							

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*.

•	CORE Stalls	Events	PMU Events	Derived Stats	Instructions	Help				
Instructions Executed (by Count)										
#	Tag	)		Syntax		Count	Pct			
0	F2_sfadd		Rd32=sfadd(R	s32,Rt32)		2123490	14.406%			
1	F2_sfmpy		Rd32=sfmpy(F	Rs32,Rt32)		1697875	11.519%			
2	A2_tfr		Rd32=Rs32			627448	4.257%			
3	A2_padditnew		if (Pu4.new) R	d32=add(Rs32,#s	B)	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	Pd4=sfcmp.uo(Rs32,Rt32)						
7	A2_paddifnew		if (!Pu4.new) F	572098	3.881%					
8	J2_jumpfnew		if (!Pu4.new) j	537887	3.649%					
9	J2_endloop0		endloop0	endloop0						
10	F2_sfsub		Rd32=sfsub(R	s32,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_jumptnewp	t	if (Pu4.new) ju	292315	1.983%					
14	S4_storeirbtne	w_io	if (Pv4.new) m	284088	1.927%					
15	Y2_tfrscrr		Rd32=Ss64	261748	1.776%					
16	SA1_cmpeqi		p0=cmp.eq(Rs	220479	1.496%					
17	SL2_jumpr31_	tnew	if (p0.new) jun	220476	1.496%					
18	A2_addi		Rd32=add(Rs3	200514	1.360%					
19	A2_and		Rd32=and(Rs3	179673	1.219%					
20	J2_pause		pause(#u8)			179669	1.219%			
21	A2_tfrsi		Rd32=#s16			164644	1.1179			
22	J2_call		call #r22:2			124498	0.845%			
23	L2_loadw_lock	ed	Rd32=memw_	Rd32=memw_locked(Rs32)						
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	0.8149			
27	SL2 loadrd sp	6	84090	0.570%						

Figure 3-13 Instructions tab

# 3.7 Help tab

The Help tab shows information about the test parameters.

CORE Stalls	Events	PMU Events	Derived Stats	Instructions	Help			
NOTE: On each cloc with 4 threads that s Further assume the case the cycles*4 wi of all other stall type	k tick, the a simulates fo active threa Il be 4 billio es will be 50	nctivity on every r 1 second but o d spends half its n, WAIT_CYCLEs 0 million.	thread is accumul only 1 thread is act s time committing 5 will be 3 billion,	ated. Assume for tive and the othe packets and half commits will be 5	example rs are in the time 500 millio	e a 1GHz core WAIT mode. stalled. In this on, and the total		
Simulation Setting	IS:							
Profile data version:	2.5							
cache_config:	L1-I = 3	2 Kb, L1-D\$ = 1	16 Kb, L2-\$ = 102	4 Kb				
command_line:	magic_ ahbbus l2tcm_ba	magic_angelquietinfo_quietrevid 0xa666axibusratio 2axibuspenalty 75 ahbbusratio 2ahbbuspenalty 75axi2busratio 2axi2buspenalty 75 l2tcm base 0xd8000000timingclade2 stable assert hack 0x0						
core:	V66H_15	36						
a6version:	v66							

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.

CORE Stalls	Events PM	U Events	Derived Stats	Instru	uctions	Help	thread	Ъ		×	
Summary		Count	Pct	-	. Тор	Packets	Тор	Function	15		
Total		3588410	1 100.0%		c	ommits	and Sta	lls			
Stalls		2860697	4 79.7%		#	Count	Pct	CumPct	Address	Packet	
Commits		727712	7 20.3%		1 1	4093247	39.3%	39.3%	648c	thread_stop +3c	
Stall Types		Count	Pct	/	2	2161322	6.0%	45.3%	62dc	compute_fractal +88	
TOFF_CYCLES		1437228	0 40.1%		3	2153053	6.0%	51.3%	62d8	compute_fractal +84	
CU_INTERLOCK_CYCLES		961515	1 26.8%		4	2050543	5.7%	57.0%	62d0	compute_fractal +7c	
PAUSE_CYCLES		71867	6 2.0%		5	2007308	5.6%	62.6%	62fc	compute_fractal +a8	
CU_NO_DISPATCH_CYCL	ES	70369	2.0%		6	2006984	5.6%	68.2%	6304	compute_fractal +b0	
CU_PREG_INTERLOCK_C	YCLES	85950	1 1.8%		7	1676411	4.7%	72.9%	62f4	compute_fractal +a0	
CU_FP_RX_NO_NTWK_C	YCLES	57786	7 1.6%		8	1490085	4.2%	77.0%	62e4	compute_fractal +90	
FE_MISPREDICT_TIME_C	YCLES	49999	7 1.4%		9	1006683	2.8%	79.8%	64c4	thread join +24	
CU_BE_NOB2B_CYCLES		28466	8 0.8%		10	413534	1.2%	81.0%	64bc	thread join +1c	
DCACHE_DEMAND_LOAD	MISS_CYCL	ES 25783	1 0.7%	-	11	412535	1.1%	82.1%	64c8	thread join +28	-
Function	Address	Stalls	Percent	Co	ount	Disasse	mbly /	Stall Na	ne		
thread start	0xd9c				6	jumpr r2	8				
			83.3%		5	Comn	nits				
			16.7%		1	Stalls					
			- 16.7%		1	CU	NO_DIS	SPATCH_	CYCLES		
event_handle_tlbmis	. 0xf00				173	crswap(r	29,sgp0)	)			
		1	0.6%		1	Comn	nits				
		-	99.4%		172	Stalls					
			97.7%	169		ICACHE DEMAND MISS CYCLES					100

Figure 3-15 Search grid box

# 3.9 Optional HVX Stalls tab

CORE Stalls	HVX Stalls	vents PM	1U Events	Instructions	Help	Sear	ch grid			
Summary	Count	Pct		@ T	op Pac	kets 🔘 T	op Functi	ons		-
Total	284	100.0%				0.001703				
Stalls	184	64.8%		HVX	_LD_L	2_001SI	ANDING	Addungs	Dackat	- 1
Commits	100	35.2%		#	Count	FCE 20/-	56 20%	Address	Packet	
Stall Types	Count	Pct		2	15	10.4%	66 7%	6ed4	main $\pm 2b0$	
HVX_LD_L2_OUT	STANDING 144	50.7%		3	12	8 3%	75.0%	6ec8	main + 324	
HVX_LD_VTCM_OL	TSTANDING 34	12.0%		4	5	3.5%	78.5%	6010	main +f8	
HVX_ST_L2_OUTS	TANDING 6	2.1%		5	5	3.5%	81.9%	6d24	main +200	
				6	5	3.5%	85.4%	6d40	main $\pm 21c$	
				7	- 5	3.5%	88.9%	6da8	main +284	
				8	5	3.5%	92.4%	6e3c	main +318	
				9	5	3.5%	95.8%	6e48	main +324	
				10	5	3.5%	99.3%	6eac	main +388	
				11	1	0.7%	100.0%	6d3c	main +218	
Function	Address	Stalls	Percent	Count	Dis	assembl	y / Stall N	lame		
					C:\(	Qualcomm	HEXAGO	V_Tools\8	.3.06\Examples\Stand	Alone
main+164	6c88			84	v3 =	= vmemu	(r20+#0x0	))		
		•	3.6%	3	(	Commits				
			96.4%	81	9	Stalls				
			96.4%	, ,	81	HVX_LL	L2_OUTS	STANDING	5	
					C:\(	Qualcomm	HEXAGO	V_Tools\8	.3.06\Examples\Stand	Alone
main+168	6c8c			0	m0	= r3				
main+16c	6c90			1	vtm	np.h = vga	ther(r18,n	n0,v3.h).h	n; vmem(r2+#0x0) =	vtmp
			100%	1	(	Commits				
			0%	0	5	Stalls				
					C:\(	Qualcomm	HEXAGO	V_Tools\8	.3.06\Examples\Stand	Alone
main+174	6c98			9	call	<u>0x6f70</u> q	5sim_read_	_pcycles;	v4.cur = vmem(r2+#	0x0);
			22.2%	2	(	Commits				
			77.8%	7	5	Stalls				
			77.8%	2	7	HVX_LL	_VTCM_O	UTSTAND	ING	

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.