Qualcomm

Qualcomm Technologies, Inc.

Qualcomm® qprintf Library

80-VB419-109 A

March 5, 2018

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

Revision history

Revision	Date	Description
A	February 2018	Initial release
???	March 2018	Expended C support

Contents

1 Introduction

1.1 Purpose

The Qualcomm® qprintf library provides printf-like support on the Qualcomm Hexagon™ processor:

- It can be called from C as well as from assembly, where it can be interleaved with any assembly package
- When invoked from C or assembly, it displays custom messages, along with any available register types (scalar, vector, predicate), in several user-friendly formats
- When invoked from assembly, it displays the file name and line number in which the qprintf command was issued

1.2 Conventions

Function declarations, function names, type declarations, attributes, and code samples appear in a different font, for example, `cp armcc armcpp`.

Code variables appear in angle brackets, for example, `<number>`.

1.3 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 the CDMATech Support website, register for access or send email to support.cdmatech@qti.qualcomm.com.

2 Background

The qprintf library is written exclusively in C and assembly, and it runs on the simulator and all targets. It is a generalization of the qdebug library, which will be discontinued, and it is available in the Hexagon SDK.

The examples provided in this document are available under the `examples/compute/qprintf_example` project, which illustrates how qprintf can be made accessible and used from any project.

3 Assembly support

The following examples provide code blocks followed by the resulting command line output when running on the Hexagon simulator. The examples are taken directly from the qprintf_examples project mentioned in Chapter 2.

3.1 Scalar register display

Code block:

```
qprintf("qprintf extends printf support to assembly.");
qprintf("You can use it for regular text including tabs\t and line breaks\n or even
nothing!");
qprintf("");
qprintf("Unlike printf, qprintf begins each statement with the file name and line
number and ends them with a return character.");
qprintf("You can also print scalar registers in any format you want. For example, R20
is set to -1 and you can express it as %%d: %d",r20);
qprintf("Or any printf format eally: %%u: %u, %%x: %x, %%23d: %23d, %%+.6d: %+.6d.
Etc. ",r20,r20,r20,r20);
qprintf("qprintf also extends new formats to display registers as two 16-bit
values.");
qprintf("E.g. %%dd for two signed 16-bit: %dd. Or %%ud for unsigned-signed: %ud, and
so on... %%du: %du, %%uu: %uu",r20, r20, r20, r20);
qprintf("And same with four 8-bit values. For example, %%dddd: %dddd, %%duud:
%duud, %%uddu: %uddu, etc.",r20, r20, r20);
R22 = #4
R22 = convert_w2sf(R22)
qprintf("We support float format too. r22 (now 4.0 in float) as %%e:%e
or %%5.2f: %5.2f (different from %%d): %d",r22,r22,r22);
```

Output:

```
qdebug_sample_asm.S[231]: qprintf extends printf support to assembly.
qdebug_sample_asm.S[232]: You can use it for regular text including tabs and line
breaks or even nothing!
qdebug_sample_asm.S[233]:
qdebug_sample_asm.S[234]: Unlike printf, qprintf begins each statement with the file
name and line number and ends them with a return character.
qdebug_sample_asm.S[235]: You can also print scalar registers in any format you want.
For example, R20 is set to -1 and you can express it as %d: -1
qdebug_sample_asm.S[236]: Or any printf format really: %u: 4294967295, %x:
ffffffff, %23d: -1, %+.6d: -000001. Etc.
```

```
qdebug_sample_asm.S[237]: qprintf also extends new formats to display registers as two
16-bit values.
qdebug_sample_asm.S[238]: E.g. %dd for two signed 16-bit: -1| -1. Or %ud for unsigned-
signed: 65535| -1, and so on... %du: -1| 65535, %uu: 65535| 65535
qdebug_sample_asm.S[239]: And same with four 8-bit values. For example, %dddd: -1| -1|
-1| -1, %duud: -1| 255| 255| -1, %uddu: 255| -1| -1| 255, etc.
qdebug_sample_asm.S[242]: We support float format too. r22 (now 4.0 in float)
as %e:4.000000e+00 or %5.2f: 4.00 (different from %d): 1082130432
```

3.2 Vector register display

Code block:

```
qprintf("You can print hvx registers as well. v0 as %%11d: %11d et voila",v0);
qprintf("Note: The end of a formatter for a v register needs to be identified with
<end of string>, <space>, ',', ';', '|', '\\', or ':'");
qprintf("I.e. you can't do %%dthis but you can do %%d that or %%08x|%%+11u,or end the
string with %%u");
qprintf("The 16-bit and 8-bit display format is also supported for hvx vectors. E.g.
v0 as %%du: %du",v0);
qprintf("v0 as %%duud: %duud",v0);
qprintf("You can also specify the number of columns per row to display with (n). For
example, v0 as %%(3)+.6d: %(3)+.6d",v0);
qprintf("And you use a mask (set previously from C) if you only care about some v0
values.");
qprintf("\nHere the mask has been set in C as\n
qd_set_mask(QD_MASK_ODD_32,QD_MASK_EVEN_32);\nthus selecting the 8 even words from the
lowest 64 bytes and the 8 odd words from the upper 64 bytes with v0 as %%md: %md",v0);
qprintf("v0 as %%mdd: %mdd",v0);
qprintf("v0 as %%m(5)uu: %m(5)uu",v0);
qprintf("v0 as %%mdddd: %mdddd",v0);
qprintf("v0 as %%m(12)uuuu: %m(12)dddu",v0);
V22 = VSPLAT(R22)
qprintf("v22 (4.0 in float splat across the entire register) as %%05.4f: %05.4f",v22);
```

Output:

```
qdebug_sample_asm.S[245]: You can print hvx registers as well. v0 as %11d:
 -1, -1, -1, -1, -1, -1, -1, -1
 -1, -1, -1, -1, -1, -1, -1, -1
 -1, -1, -1, -1, -1, -1, -1, -1
 -1, -1, -1, -1, -1, -1, -1, -1 et voila
qdebug_sample_asm.S[246]: Note: The end of a formatter for a v register needs to be
identified with <end of string>, <space>, ',', ';', '|', '\', or ':'
qdebug_sample_asm.S[247]: I.e. you can't do %dthis but you can do %d that
or %08x|%+11u,or end the string with %u
qdebug_sample_asm.S[248]: The 16-bit and 8-bit display format is also supported for
hvx vectors. E.g. v0 as %du:
 -1| 65535, -1| 65535, -1| 65535, -1| 65535, -1| 65535, -1| 65535, -1| 65535, -1|
65535
```

```
 -1| 65535, -1| 65535, -1| 65535, -1| 65535, -1| 65535, -1| 65535, -1| 65535, -1|
65535
 -1| 65535, -1| 65535, -1| 65535, -1| 65535, -1| 65535, -1| 65535, -1| 65535, -1|
65535
 -1| 65535, -1| 65535, -1| 65535, -1| 65535, -1| 65535, -1| 65535, -1| 65535, -1|
65535
qdebug_sample_asm.S[249]: v0 as %duud:
 -1| 255| 255| -1, -1| 255| 255| -1, -1| 255| 255| -1, -1| 255| 255| -1
 -1| 255| 255| -1, -1| 255| 255| -1, -1| 255| 255| -1, -1| 255| 255| -1
 -1| 255| 255| -1, -1| 255| 255| -1, -1| 255| 255| -1, -1| 255| 255| -1
 -1| 255| 255| -1, -1| 255| 255| -1, -1| 255| 255| -1, -1| 255| 255| -1
 -1| 255| 255| -1, -1| 255| 255| -1, -1| 255| 255| -1, -1| 255| 255| -1
 -1| 255| 255| -1, -1| 255| 255| -1, -1| 255| 255| -1, -1| 255| 255| -1
 -1| 255| 255| -1, -1| 255| 255| -1, -1| 255| 255| -1, -1| 255| 255| -1
 -1| 255| 255| -1, -1| 255| 255| -1, -1| 255| 255| -1, -1| 255| 255| -1
qdebug_sample_asm.S[250]: You can also specify the number of columns per row to
display with (n). For example, v0 as %(3)+.6d:
 -000001,-000001,-000001
 -000001,-000001,-000001
 -000001,-000001,-000001
 -000001,-000001,-000001
 -000001,-000001,-000001
 -000001,-000001,-000001
 -000001,-000001,-000001
 -000001,-000001,-000001
 -000001,-000001,-000001
 -000001,-000001,-000001
 -000001,-000001,
qdebug_sample_asm.S[251]: And you use a mask (set previously from C) if you only care
about some v0 values.
qdebug_sample_asm.S[252]:
Here the mask has been set in C as
 qd_set_mask(QD_MASK_ODD_32,QD_MASK_EVEN_32);
thus selecting the 8 even words from the lowest 64 bytes and the 8 odd words from the
upper 64 bytes with v0 as %md:
 [7c]=-1,[74]=-1,[6c]=-1,[64]=-1,[5c]=-1,[54]=-1,[4c]=-1,[44]=-1
 [38]=-1,[30]=-1,[28]=-1,[20]=-1,[18]=-1,[10]=-1,[08]=-1,[00]=-1
qdebug_sample_asm.S[253]: v0 as %mdd:
 [7e]= -1,[7c]= -1,[76]= -1,[74]= -1,[6e]= -1,[6c]= -1,[66]= -1,[64]= -1
 [5e]= -1,[5c]= -1,[56]= -1,[54]= -1,[4e]= -1,[4c]= -1,[46]= -1,[44]= -1
 [3a]= -1,[38]= -1,[32]= -1,[30]= -1,[2a]= -1,[28]= -1,[22]= -1,[20]= -1
 [1a]= -1,[18]= -1,[12]= -1,[10]= -1,[0a]= -1,[08]= -1,[02]= -1,[00]= -1
qdebug_sample_asm.S[254]: v0 as %m(5)uu:
 [7e]= 65535,[7c]= 65535,[76]= 65535,[74]= 65535,[6e]= 65535
 [6c]= 65535,[66]= 65535,[64]= 65535,[5e]= 65535,[5c]= 65535
 [56]= 65535,[54]= 65535,[4e]= 65535,[4c]= 65535,[46]= 65535
 [44]= 65535,[3a]= 65535,[38]= 65535,[32]= 65535,[30]= 65535
 [2a]= 65535,[28]= 65535,[22]= 65535,[20]= 65535,[1a]= 65535
 [18]= 65535,[12]= 65535,[10]= 65535,[0a]= 65535,[08]= 65535
 [02]= 65535,[00]= 65535,
qdebug_sample_asm.S[255]: v0 as %mdddd:
 [7f]= -1,[7e]= -1,[7d]= -1,[7c]= -1,[77]= -1,[76]= -1,[75]= -1,[74]= -1
 [6f]= -1,[6e]= -1,[6d]= -1,[6c]= -1,[67]= -1,[66]= -1,[65]= -1,[64]= -1
 [5f]= -1,[5e]= -1,[5d]= -1,[5c]= -1,[57]= -1,[56]= -1,[55]= -1,[54]= -1
 [4f]= -1,[4e]= -1,[4d]= -1,[4c]= -1,[47]= -1,[46]= -1,[45]= -1,[44]= -1
 [3b]= -1,[3a]= -1,[39]= -1,[38]= -1,[33]= -1,[32]= -1,[31]= -1,[30]= -1
 [2b]= -1,[2a]= -1,[29]= -1,[28]= -1,[23]= -1,[22]= -1,[21]= -1,[20]= -1
 [1b]= -1,[1a]= -1,[19]= -1,[18]= -1,[13]= -1,[12]= -1,[11]= -1,[10]= -1
 [0b]= -1,[0a]= -1,[09]= -1,[08]= -1,[03]= -1,[02]= -1,[01]= -1,[00]= -1
qdebug_sample_asm.S[256]: v0 as %m(12)uuuu:
 [7f]= 255,[7e]= 255,[7d]= 255,[7c]= 255,[77]= 255,[76]= 255,[75]= 255,[74]= 255,[6f]=
255,[6e]= 255,[6d]= 255,[6c]= 255
 [67]= 255,[66]= 255,[65]= 255,[64]= 255,[5f]= 255,[5e]= 255,[5d]= 255,[5c]= 255,[57]=
255,[56]= 255,[55]= 255,[54]= 255
 [4f]= 255,[4e]= 255,[4d]= 255,[4c]= 255,[47]= 255,[46]= 255,[45]= 255,[44]= 255,[3b]=
255,[3a]= 255,[39]= 255,[38]= 255
 [33]= 255,[32]= 255,[31]= 255,[30]= 255,[2b]= 255,[2a]= 255,[29]= 255,[28]= 255,[23]=
255,[22]= 255,[21]= 255,[20]= 255
 [1b]= 255,[1a]= 255,[19]= 255,[18]= 255,[13]= 255,[12]= 255,[11]= 255,[10]= 255,[0b]=
255,[0a]= 255,[09]= 255,[08]= 255
 [03]= 255,[02]= 255,[01]= 255,[00]= 255,
qdebug_sample_asm.S[258]: v22 (4.0 in float splat across the entire register)
as %05.4f:
 4.0000,4.0000,4.0000,4.0000,4.0000,4.0000,4.0000,4.0000
 4.0000,4.0000,4.0000,4.0000,4.0000,4.0000,4.0000,4.0000
 4.0000,4.0000,4.0000,4.0000,4.0000,4.0000,4.0000,4.0000
 4.0000,4.0000,4.0000,4.0000,4.0000,4.0000,4.0000,4.0000
```

3.3 Predicate register display

Code block:

```
qprintf("You can also print p registers in any format. p0: %x, p1: %08x",p0,p1);
Q1=vcmp.gt(V18.w,V19.w)
qprintf("You can also print vector predicate registers. E.g. word-wise comparison
between %(32)2d and %(32)2d is\n  %d",v18,v19,q1);
Q3=vcmp.gt(V18.b,V19.b)
qprintf("while byte-wise comparison between %(8)dddd and %(8)dddd is\n  %x",
v18,v19,q3);
// Please note formatter is ignored when displaying vector predicate registers. We
always display in hex.
```

Output:

```
qdebug_sample_asm.S[261]: You can also print p registers in any format. p0: ff, p1:
00000000
qdebug_sample_asm.S[263]: You can also print vector predicate registers. E.g. word-
wise comparison between
 32,31,30,29,28,27,26,25,24,23,22,21,20,19,18,17,16,15,14,13,12,11,10, 9, 8, 7, 6, 5,
4, 3, 2, 1 and
 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
8, 8, 8, 8 is
 Q1: 0xffffffff|ffffffff|ffffffff|00000000
qdebug_sample_asm.S[265]: while byte-wise comparison between
 0| 0| 0| 32, 0| 0| 0| 31, 0| 0| 0| 30, 0| 0| 0| 29, 0| 0| 0| 28, 0| 0| 0| 27, 0| 0|
0| 26, 0| 0| 0| 25
 0| 0| 0| 24, 0| 0| 0| 23, 0| 0| 0| 22, 0| 0| 0| 21, 0| 0| 0| 20, 0| 0| 0| 19, 0| 0|
0| 18, 0| 0| 0| 17
 0| 0| 0| 16, 0| 0| 0| 15, 0| 0| 0| 14, 0| 0| 0| 13, 0| 0| 0| 12, 0| 0| 0| 11, 0| 0|
0| 10, 0| 0| 0| 9
 0| 0| 0| 8, 0| 0| 0| 7, 0| 0| 0| 6, 0| 0| 0| 5, 0| 0| 0| 4, 0| 0| 0| 3, 0| 0| 0| 2,
0| 0| 0| 1 and
 0| 0| 0| 8, 0| 0| 0| 8, 0| 0| 0| 8, 0| 0| 0| 8, 0| 0| 0| 8, 0| 0| 0| 8, 0| 0| 0| 8,
0| 0| 0| 8
 0| 0| 0| 8, 0| 0| 0| 8, 0| 0| 0| 8, 0| 0| 0| 8, 0| 0| 0| 8, 0| 0| 0| 8, 0| 0| 0| 8,
0| 0| 0| 8
 0| 0| 0| 8, 0| 0| 0| 8, 0| 0| 0| 8, 0| 0| 0| 8, 0| 0| 0| 8, 0| 0| 0| 8, 0| 0| 0| 8,
0| 0| 0| 8
 0| 0| 0| 8, 0| 0| 0| 8, 0| 0| 0| 8, 0| 0| 0| 8, 0| 0| 0| 8, 0| 0| 0| 8, 0| 0| 0| 8,
0| 0| 0| 8 is
 Q3: 0x11111111|11111111|11111111|00000000
```

3.4 Register dump

Code block:

```
qprintf("And finally you can print all scalar or v registers. For example, r as %%8x
returns: %8x",r);
qprintf("And v as %%(16)x returns: %(16)x",v);
```

Output:

```
qdebug_sample_asm.S[269]: And finally you can print all scalar or v registers. For
example, r as %8x returns:
 R0 : fffffff, R1 : 8, R2 : 4, R3 : 3, R4 : 4, R5 : 5, R6 : 6, R7 : 7
 R8 : 8, R9 : 9, R10: a, R11: b, R12: c, R13: d, R14: e, R15: f
 R16: 10, R17: 11, R18: 12, R19: 13, R20: ffffffff, R21: 15, R22: 40800000, R23: 17
 R24: 18, R25: 19, R26: 1a, R27: 1b, R28: ff, R29: 28007f28, R30: 28007f78, R31:
2301f044
 P0: 000000ff, P1: 00000000, P2: 00000000, P3: 000000ff
qdebug_sample_asm.S[270]: And v as %(16)x returns:
V0 :

ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,fffff
fff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff

ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,fffff
fff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff,ffffffff
V1 :
 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
V2 :
 2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
 2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
V3 :
 3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3
 3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3
V4 :
 4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4
 4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4
V5 :
 5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5
 5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5
V6 :
 6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6
 6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6
V7 :
 7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7
 7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7
```

```
V8 :
 8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8
 8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8
V9 :
 9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
 9,9,9,9,9,9,9,9,9,9,9,9,9,9,9,9
V10:
 a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a
 a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a
V11:
 b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,b
 b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,b
V12:
 c,c,c,c,c,c,c,c,c,c,c,c,c,c,c,c
 c,c,c,c,c,c,c,c,c,c,c,c,c,c,c,c
V13:
 d,d,d,d,d,d,d,d,d,d,d,d,d,d,d,d
 d,d,d,d,d,d,d,d,d,d,d,d,d,d,d,d
V14:
 e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e
 e,e,e,e,e,e,e,e,e,e,e,e,e,e,e,e
V15:
 f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f
 f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f
V16:
 10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10
 10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10
V17:
 11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11
 11,11,11,11,11,11,11,11,11,11,11,11,11,11,11,11
V18:
 20,1f,1e,1d,1c,1b,1a,19,18,17,16,15,14,13,12,11
 10,f,e,d,c,b,a,9,8,7,6,5,4,3,2,1
V19:
 8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8
 8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8
V20:

40800000,40800000,40800000,40800000,40800000,40800000,40800000,40800000,40800000,40800
000,40800000,40800000,40800000,40800000,40800000,40800000

40800000,40800000,40800000,40800000,40800000,40800000,40800000,40800000,40800000,40800
000,40800000,40800000,40800000,40800000,40800000,40800000
V21:
 15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15
 15,15,15,15,15,15,15,15,15,15,15,15,15,15,15,15
```

```
V22:

40800000,40800000,40800000,40800000,40800000,40800000,40800000,40800000,40800000,40800
000,40800000,40800000,40800000,40800000,40800000,40800000

40800000,40800000,40800000,40800000,40800000,40800000,40800000,40800000,40800000,40800
000,40800000,40800000,40800000,40800000,40800000,40800000
V23:
 17,17,17,17,17,17,17,17,17,17,17,17,17,17,17,17
 17,17,17,17,17,17,17,17,17,17,17,17,17,17,17,17
V24:
 18,18,18,18,18,18,18,18,18,18,18,18,18,18,18,18
 18,18,18,18,18,18,18,18,18,18,18,18,18,18,18,18
V25:
 19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19
 19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19
V26:
 1a,1a,1a,1a,1a,1a,1a,1a,1a,1a,1a,1a,1a,1a,1a,1a
 1a,1a,1a,1a,1a,1a,1a,1a,1a,1a,1a,1a,1a,1a,1a,1a
V27:
 1b,1b,1b,1b,1b,1b,1b,1b,1b,1b,1b,1b,1b,1b,1b,1b
 1b,1b,1b,1b,1b,1b,1b,1b,1b,1b,1b,1b,1b,1b,1b,1b
V28:
 1c,1c,1c,1c,1c,1c,1c,1c,1c,1c,1c,1c,1c,1c,1c,1c
 1c,1c,1c,1c,1c,1c,1c,1c,1c,1c,1c,1c,1c,1c,1c,1c
V29:
 1d,1d,1d,1d,1d,1d,1d,1d,1d,1d,1d,1d,1d,1d,1d,1d
 1d,1d,1d,1d,1d,1d,1d,1d,1d,1d,1d,1d,1d,1d,1d,1d
V30:
 1e,1e,1e,1e,1e,1e,1e,1e,1e,1e,1e,1e,1e,1e,1e,1e
 1e,1e,1e,1e,1e,1e,1e,1e,1e,1e,1e,1e,1e,1e,1e,1e
V31:
 1f,1f,1f,1f,1f,1f,1f,1f,1f,1f,1f,1f,1f,1f,1f,1f
 1f,1f,1f,1f,1f,1f,1f,1f,1f,1f,1f,1f,1f,1f,1f,1f
Q0: 0x00000000|00000000|00000000|00000000, Q1: 0xffffffff|ffffffff|ffffffff|00000000
Q2: 0x00000000|00000000|00000000|00000000, Q3: 0x11111111|11111111|11111111|00000000
```

4 C support

The generic C library printf function already supports display of 32-bit variables and should be used to display these variables.

The qprintf library supports functions used to specifically display an HVX regular vector or predicate register. For example, for the following code:

```
HVX_Vector x = Q6_V_vsplat_R(-1);
HVX_VectorPred pred = Q6_Q_vand_VR(x,-1);
qprintf_V("x = %d\n",x);
qprintf_Q("pred = %x\n",pred);
// as with assembly, format is ignored when displaying predicate
registers
```

The result is this example:

```
x =
  -1,-1,-1,-1,-1,-1,-1,-1
  -1,-1,-1,-1,-1,-1,-1,-1
  -1,-1,-1,-1,-1,-1,-1,-1
  -1,-1,-1,-1,-1,-1,-1,-1
pred = 0xffffffff|ffffffff|ffffffff|ffffffff
```

qprintf also provides APIs to display the contents of all vector or scalar registers as hexadecimal numbers in one go. These APIs are `qprintf_V_all` and `qprintf_R_all`.

For example,

```
qprintf_R_all();
```

results in the following type of output.

```
R0 : 00000000, R1 : 00000000, R2 : 2306c2bc, R3 : 28030448, R4 : 28030448, R5 : 230
R8 : 23016b44, R9 : a1870812, R10: 00000000, R11: 30303030, R12: 3030300a, R13: 303
R16: 2306c138, R17: 23058780, R18: 280302a0, R19: 280302c0, R20: 00000001, R21: 230
R24: 00000000, R25: 00000000, R26: 28030448, R27: 2305b610, R28: 00000000, R29: 280
P0:  00000000, P1:  000000ff, P2:  00000004, P3:  00000000
```