

Hands-on Hexagon Toolchain Training

80-VB419-42 N



Confidential and Proprietary – Qualcomm Technologies, Inc.

Restricted Distribution: Not to be distributed to anyone who is not an employee of either Qualcomm or its subsidiaries without the express approval of Qualcomm's Configuration Management.

Confidential and Proprietary – Qualcomm Technologies, Inc.

Confidential and Proprietary – Qualcomm Technologies, Inc.

NO PUBLIC DISCLOSURE PERMITTED: Please report postings of this document on public servers or websites to: DocCtrlAgent@qualcomm.com.

Restricted Distribution: Not to be distributed to anyone who is not an employee of either Qualcomm or its subsidiaries without the express approval of Qualcomm's Configuration Management.

Not to be used, copied, reproduced, or modified in whole or in part, nor its contents revealed in any manner to others without the express written permission of Qualcomm Technologies, Inc.

Qualcomm reserves the right to make changes to the product(s) or information contained herein without notice. No liability is assumed for any damages arising directly or indirectly by their use or application. The information provided in this document is provided on an "as is" basis.

This document contains confidential and proprietary information and must be shredded when discarded.

Qualcomm and Hexagon are trademarks of QUALCOMM Incorporated, registered in the United States and other countries. All QUALCOMM Incorporated trademarks are used with permission. 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.

© 2007-2013 Qualcomm Technologies, Inc.
All rights reserved.

Structures, padding and alignment – Example1

```
// this structure will have 4 byte alignment
// because it's largest value is int
struct unpacked_struct
{
    char a;
    int i1;
    int i2;
};

// declare and initialize the unpacked struct
struct unpacked_struct my_unpacked_struct = { 'a', 0x11, 0x22 };

// this structure will have 1 byte alignment
// because it is packed
struct packed_struct
{
    char a;
    int i1;
    int i2;

} __attribute__((__packed__));

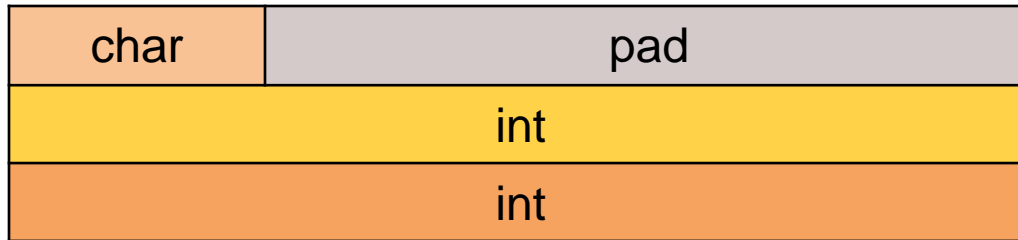
// declare and initialize the packed struct
struct packed_struct my_packed_struct = { 'b', 0x33, 0x44 };
```

Structures, padding and alignment

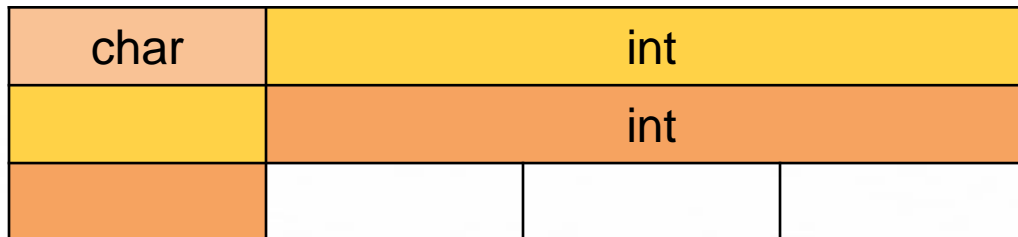
- hexagon-readelf -s main | grep packed

```
462: 0000d430      9 OBJECT GLOBAL DEFAULT    7 my_packed_struct
501: 0000d424     12 OBJECT GLOBAL DEFAULT    7 my_unpacked_struct
```

- my_unpacked_struct



- my_packed_struct



Structures, padding and alignment

```
>hexagon-llvm-objdump -d main.o
```

```
main.o: file format ELF32-hexagon
```

```
Disassembly of section .text:
```

```
.text:
```

```
main:
```

```
0:      00 40 00 00 00004000 { immext(#0)
4:      10 3c 00 68 68003c10   r0 = ##0; allocframe(#8) }
8:      00 40 00 5a 5a004000 { call 0x20
c:      00 40 00 00 00004000   immext(#0)
10:     42 40 80 49 49804042   r2 = memw(##2)
14:     00 d2 bd a1 a1bdd200   memw(r29 + #0) = r2.new }
18:     00 40 00 00 00004000 { immext(#0)
1c:     a2 40 00 78 780040a2   r2 = ##5
20:     00 40 00 00 00004000   immext(#0)
24:     00 c0 00 78 7800c000   r0 = ##0 }
28:     24 11 23 10 10231124 { r3 = memub(r2 + #0); r4 = memub(r2 + #1) }
2c:     c3 48 44 8e 8e4448c3 { r3 |= asl(r4, #8)
30:     22 12 25 13 13251222   r5 = memub(r2 + #3); r2 = memub(r2 + #2) }
34:     c2 c8 45 8e 8e45c8c2 { r2 |= asl(r5, #8) }
38:     c3 50 42 8e 8e4250c3 { r3 |= asl(r2, #16)
3c:     00 40 00 5a 5a004000   call 0x38
40:     00 d4 bd a1 a1bdd400   memw(r29 + #0) = r3.new }
44:     40 3f 00 48 48003f40 { r0 = #0; dealloc_return }
```

Structures, padding and alignment – Example2

```
struct unpacked_struct
{
    char a;
    int i1;
    int i2;
};

struct packed_struct
{
    int i;
    char c;
    struct unpacked_struct s;
} __attribute__((__packed__));
struct packed_struct my_packed_struct = {0x33, 'a', 'b', 0x55, 0x66 };

// create a pointer to the packed structure
struct packed_struct *ptr_to_packed = &my_packed_struct;
// create a pointer to the unpacked structure
struct unpacked_struct *ptr_to_unpacked = &my_packed_struct.s;

int main(int argc, char **argv)
{
    printf("\nVariable i2 = %x\n", ptr_to_packed->s.i2);
    printf("\nVariable i2 = %x\n", ptr_to_unpacked->i2);
    return 0;
}
```

Structures, padding and alignment – Example2

```
hexagon-sim -mv5 main
```

```
Hexagon-sim INFO: the rev_id used in the simulation is 0x00002105 (v5A)
```

```
Variable i2 = 0x66
```

```
CRASH from thread 0!
```

```
I think the exception was: 0x20, Misaligned Load @ 0xd431
```

```
Register Dump (r0 clobbered, pc subject to prior action by the exception handler):
```

```
r00=000000cd r01=00000001 r02=0000d429 r03=00000000
```

```
r04=00000000 r05=00000000 r06=00000000 r07=00000000
```

```
r08=0000ffff r09=00000000 r10=ffffffffe r11=fffffffff
```

```
r12=00000004 r13=00000000 r14=0000e339 r15=ffff1cc8
```

```
r16=0000b200 r17=00000001 r18=babebeef r19=babebeef
```

```
r20=babebeef r21=babebeef r22=babebeef r23=babebeef
```

```
r24=babebeef r25=babebeef r26=babebeef r27=babebeef
```

```
r28=000053a0 r29=0410e2c0 r30=0410e2c8 r31=000050f0
```

```
sa0=00008a34 lc0=7fffffff sa1=00000000 lc1=00000000
```

```
p30=ff00ffff m0=00000000 m1=00000000 usr=00010000
```

```
pc=00000954 ugp=00000000 gp=00000000 elr=00005100
```

```
badva0=0000d90c badva1=0000d431
```

```
ssr=80740020 ccr=00130000 tid=00000000 imask=00000000
```

```
evb=00005000 modectl=00000001 syscfg=0083a07f ipend=00000000
```

Hexagon-lldb – Example2

```
>hexagon-lldb.cmd main
```

```
Current executable set to 'main' (hexagon).
```

```
Hexagon utilities (run, start, pagetable, tlb, pv) loaded
```

```
(lldb) start
```

```
Hexagon-sim INFO: the rev_id used in the simulation is 0x00002105 (v5A)
```

```
hexagon-sim WARNING: StartGDBserver:Setting up GDB server on port 33099
```

```
Process 1 stopped
```

```
(lldb) Breakpoint 1: where = main`main + 4 at main.c:41, address = 0x000050c4
```

```
Process 1 stopped
```

```
* thread #1: tid = 0x0001, 0x000050c4 main`main(argc=1, argv=0x0000f740) + 4 at  
main.c:41, stop reason = breakpoint 1.1
```

```
    frame #0: 0x000050c4 main`main(argc=1, argv=0x0000f740) + 4 at main.c:41
```

```
    38
```

```
    39  int main(int argc, char **argv)
```

```
    40  {
```

```
-> 41          printf("\nVariable i2 = 0x%x\n", ptr_to_packed->s.i2);
```

```
    42
```

```
    43          printf("\nVariable i2 = 0x%x\n", ptr_to_unpacked->i2);
```

```
    44
```

```
(lldb)
```


Hexagon-lldb – Example2

```
(lldb) b *0x5104
```

```
Breakpoint 2: where = main`main + 68 at main.c:43, address = 0x00005104
```

```
(lldb) c
```

```
Process 1 resuming
```

```
(lldb)
```

```
Variable i2 = 0x66
```

```
Process 1 stopped
```

```
* thread #1: tid = 0x0001, 0x00005104 main`main(argc=<unavailable>, argv=<unavailable>) + 68 at main.c:43, stop reason = breakpoint 2.1
```

```
    frame #0: 0x00005104 main`main(argc=<unavailable>, argv=<unavailable>) + 68 at main.c:43
```

```
    40  {
    41      printf("\nVariable i2 = 0x%x\n", ptr_to_packed->s.i2);
    42
-> 43      printf("\nVariable i2 = 0x%x\n", ptr_to_unpacked->i2);
    44
    45      return 0;
    46  }
```

Hexagon-lldb – Example2

```
(lldb) re r
```

Thread Registers:

```
r00 = 0x0000c000  main`  
r01 = 0x00000001  main`_start + 1  
r02 = 0x0000d429  my_packed_struct + 5  
r03 = 0x00000000  main`_start
```

```
(lldb) disassemble -s 0x5104
```

main`main + 68 at main.c:43:

```
-> 0x5104: {          call 21040                      ; printf  
0x5108: r2          = memw(r2 + #8)  
0x510c: memw(r29 + #0) = r2.new }  
0x5110: {          r0 = #0; dealloc_return }
```

R2 = 0xd429

R2 + 8 = 0xd431

R2 = memw(0xd431) will cause a misaligned load exception

Hexagon-lldb – Example2

(lldb) **s**

Process 1 stopped

* thread #1: tid = 0x0001, 0x00005008 main`.EventVectors + 8, stop reason = step in
frame #0: 0x00005008 main`.EventVectors + 8

main`.EventVectors + 8:

-> 0x5008:	{	jump 2676 }	; event_handle_error
0x500c:	{	jump 2688 }	; event_handle_rsvd
0x5010:	{	jump 2752 }	; event_handle_tlbmissx
0x5014:	{	jump 2688 }	; event_handle_rsvd

(lldb) **re r**

Thread Registers:

...

elr = 0x00005104 main`main + 68 at main.c:43

badva0 = 0x0410e2c0

badva1 = 0x0000**d431** my_packed_struct + 13

ssr = 0x807600**20**

Hexagon-lldb – Example2

```
(lldb) target variable *ptr_to_packed
(packed_struct) *ptr_to_packed = {
    i = 51
    c = 'a'
    s = (a = 'b', i1 = 85, i2 = 102)
}
```

```
(lldb) target variable *ptr_to_unpacked
(unpacked_struct) *ptr_to_unpacked = (a = 'b', i1 = 85, i2 = 102)
```

```
(lldb) target variable &ptr_to_unpacked.i2
(int *) &ptr_to_unpacked.i2 = 0x0000d431
```

Trace32 – Example3

The screenshot displays the TRACE32 PowerView for Hexagon interface, showing a C program being debugged. The main window displays the source code of `main.c`, which includes a `struct packed_struct` and a `main` function. The program is currently stopped at a breakpoint at address `0x0005104`.

The `B::register /spotlight` window shows the current state of the registers. The `R0` register contains `C000`, `R1` contains `D429`, and `R2` contains `0`. The `R15` register contains `FFFF1CC6`. The `PC` register contains `5104`.

The `B::var.frame /locals /caller` window shows the call stack. The current frame is `main`, which is calling `__libc_start_main(asm)` and `.PreStart(asm)`. The `end of frame` is also visible.

The bottom status bar shows the current address `P:00005104` and the instruction `\\main\\main\\main+0x44`. The status is `stopped at breakpoint`.

```
struct packed_struct *ptr_to_packed = &my_packed_struct;
// create a pointer to the unpacked structure
struct unpacked_struct *ptr_to_unpacked = &my_packed_struct.s;

int main(int argc, char **argv)
{
    P:000050C0 0A09DC001 main: allocframe(#0x8);
    P:000050C4 41 printf("\nVariable i2 = 0x%x\n", ptr_to_packed->s.i2);
    P:000050D4 0000430078C040.. R0=#0xC000; R2=memw(GP+#0xD438);
    P:000050D8 1D231E24 R3=memub(R2+#0x0D); R4=memub(R2+#0x0E);
    P:000050E0 8E4448C3912242.. R3|=asl(R4,#0x8); R5=memub(R2+#0x10); R2=memub(R2+#0x0F);
    P:000050E4 8E45C8C2 R2|=asl(R5,#0x8);
    P:000050E8 8E4250C35A0040.. R2|=asl(R2,#0x10); call 0x5230; memw(SP+#0x0)=R3.new; ; R3|=asl(R2,#

    P:000050F4 43 printf("\nVariable i2 = 0x%x\n", ptr_to_unpacked->i2);
    P:000050F8 0000430078C040.. R0=#0xC000; R2=memw(GP+#0xD43C);
    P:00005104 5A004096918240.. call 0x5230; R2=memw(R2+#0x8); memw(SP+#0x0)=R2.new; ; call printf;

    P:00005110 45 return 0;
    P:00005114 48003F40 R0=#0x0; dealloc_return;
    P:00005118 00000000 R0=memw(R0+#0x0); R0=memw(R0+#0x0);
    P:0000511C 00000000 R0=memw(R0+#0x0); R0=memw(R0+#0x0);
    P:00005120 00000000 R0=memw(R0+#0x0); R0=memw(R0+#0x0);
    P:00005124 0000406B780D5C.. thread_c_:R4=#0x1AE4; R5=#0x1AFC;
    P:00005128 8C024247000040.. R7=asl(R2,#0x2); R6=#0x1B14; R8=#0x1;
    P:00005132 C6484288F30744.. R8=asl(R8,R2); R4=add(R7,R4); R5=add(R7,R5); R6=add(R7,R6);
    P:00005136 A0400051 memw(R4+#0x0)=R0; memw(R5+#0x0)=R1;
    P:00005140 A186C300 memw(R6+#0x0)=R3;
    P:00005144 6468C020 start(R8);
    P:00005148 529FC000 jump LR;
    P:00005152 6E8840007800C0.. thread_s_:R0=HTID; R1=#0x1;
    P:00005156 C641C0C1 R1=asl(R1,R0);
    P:00005160 6461C000 stop(R1);
    P:00005164 0000401D78036E.. R28=#0x770; jump R28;
```

`B::`

emulate trigger devices trace Data Var List PERF SYStem Step Go Break sYmbol Frame Register FPU MMU other previous

P:00005104 \\main\\main\\main+0x44 0 stopped at breakpoint MDX UP

Trace32 – Example3

The screenshot shows the TRACE32 PowerView for Hexagon debugger interface. The main window displays assembly code for the 'B::data.list' file. The code includes instructions like 'R0=memw(R0+#0x0);' and 'jump event_handle_reset;'. The right sidebar contains two panels: 'B::register/spotlight' showing register values (e.g., R0=C000, R1=1, R2=D429) and 'B::var.frame/locals/caller' showing the current function frame with arguments and locals. The bottom status bar indicates the current instruction is 'P:00005008' and the processor is 'stopped'.