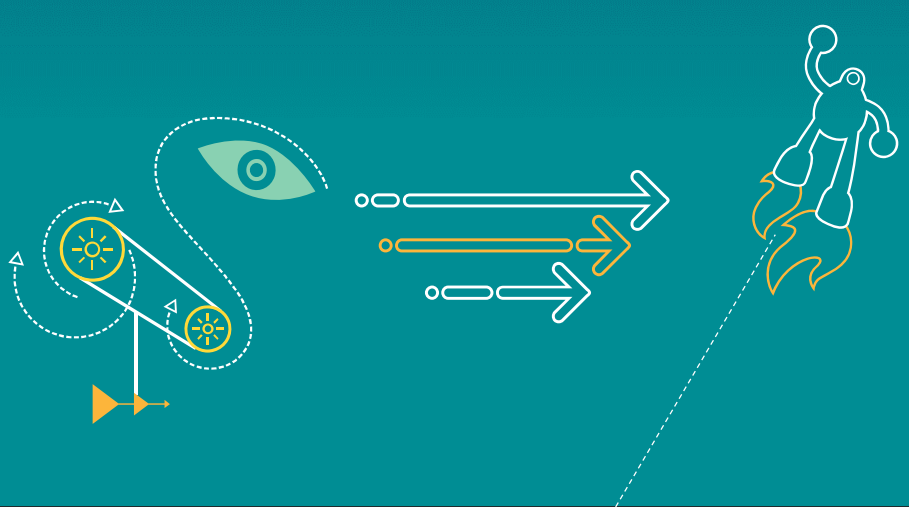
Qualcomm® Hexagon™ Programming: Software Development Tools

Qualcomm Technologies, Inc.

80-VB419-142 Rev. A



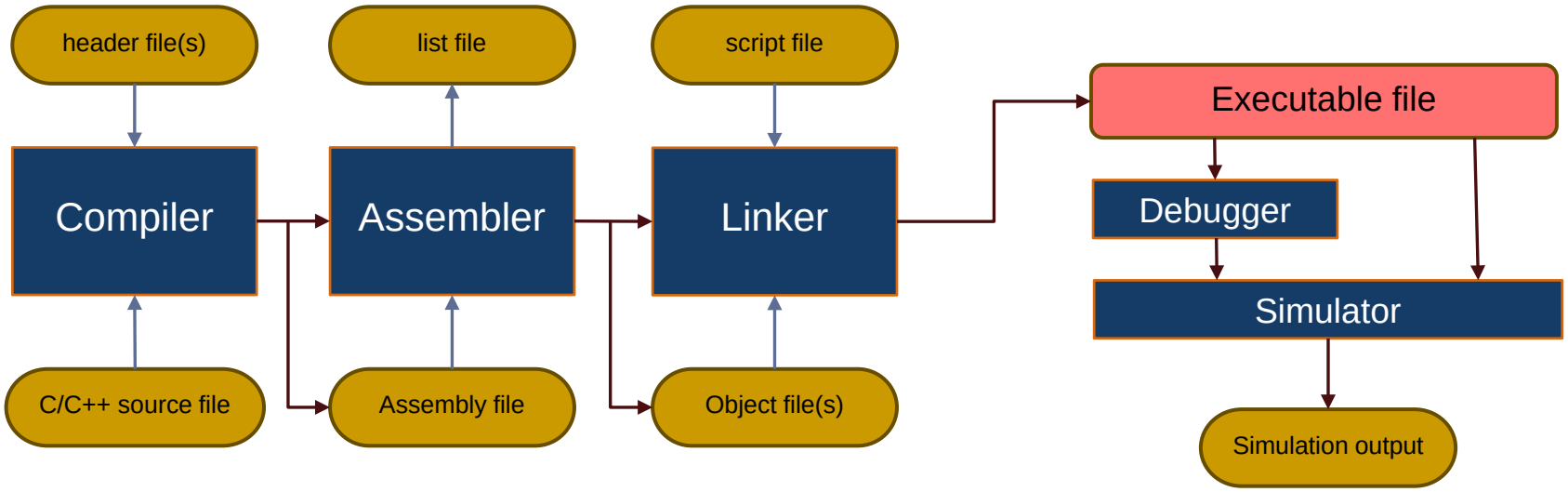
Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

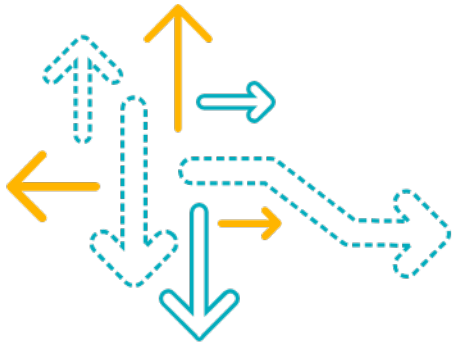
Contents

- Software Development Tools: LLVM Tools
- Software Development Tools: Simulator
- Software Development Tools: Debugger
- Software Development Tools: Profiling
- Software Development Tools: Documentation and Examples
- Questions?

Software Development Tools – Tool Flow

- Tool flow breaks out the compiler, assembler, and linker
- All three tools are used to produce an executable file
- Assembler and linker can be invoked directly by the compiler driver





Software Development Tools: LLVM Tools

LLVM Tools – Overview

- Compilers
- Assembler
- Linker
- Hexagon utilities

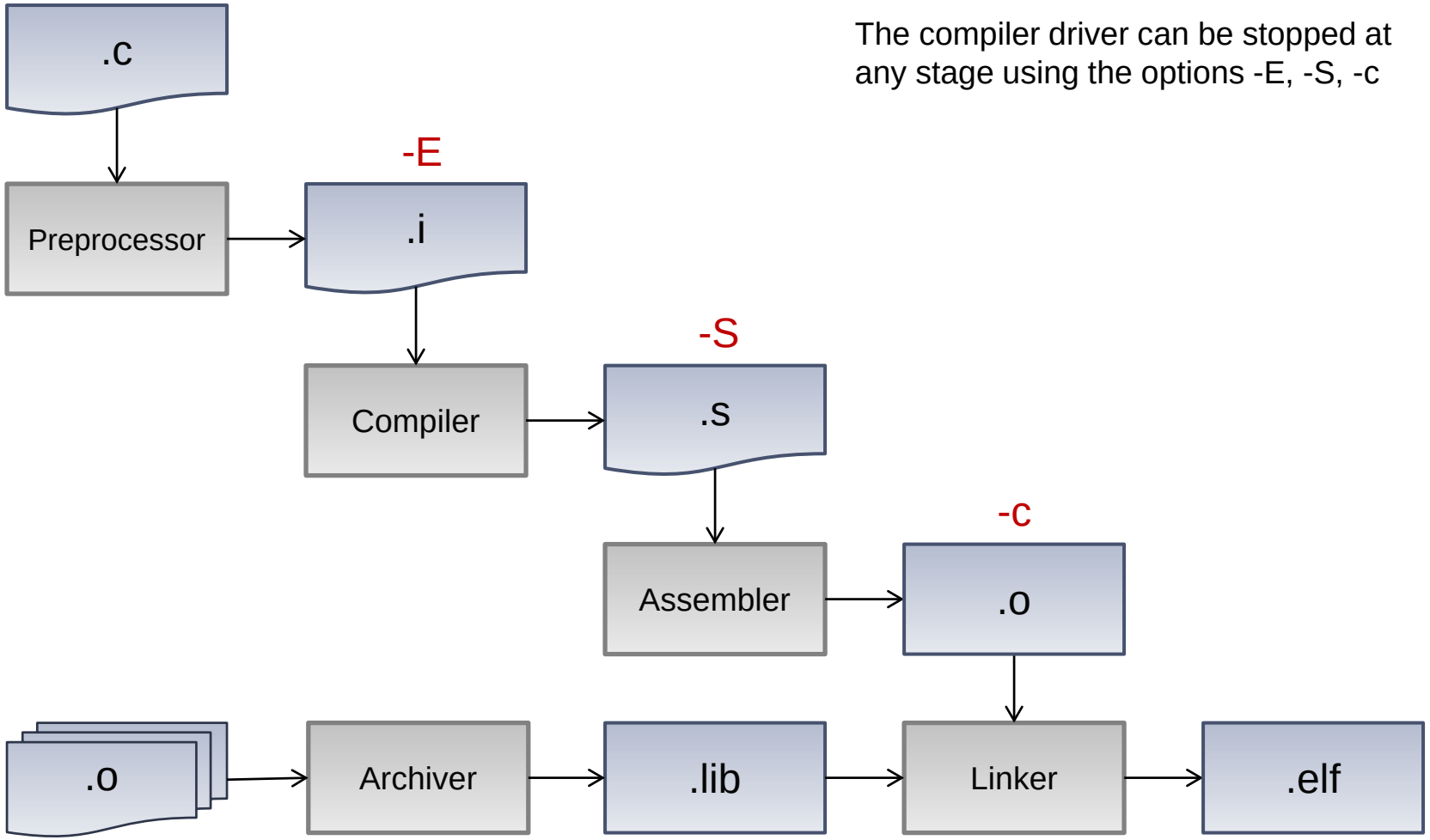
LLVM Tools – Compiler Technology

- Hexagon compilers based on LLVM
  - **`hexagon-clang`** → C compiler
  - **`hexagon-clang++`** → C++ compiler

- Hexagon compilers are cross-compilers
  - They generate assembly code for Hexagon™ processor, but run only on non-Hexagon host platforms (Windows, Linux)

- Hexagon compilers are VLIW compilers
  - They handle scheduling and packetization of code for VLIW pipeline
    - Complexity shifted from hardware to compiler

LLVM Tools – Compiler Driver Overview



The compiler driver can be stopped at any stage using the options -E, -S, -c

LLVM Tools – Compiler Usage

- Command line syntax
  - **`hexagon-clang [option…] input_file`**
    - **`hexagon-clang –O2 –o hello.exe hello.c`** → **`hello.exe (executable file)`**

Flag	Use
--help	Display help information
-O0 < -O1 < -O2 < -O3	Optimizing number of cycles
-Os	Optimize space (code size)
-g	Debug flag (best with –O0)
-c	Compile and assemble to create object file; no link
--save-temps	Save all intermediate files to disk (.i, .s)
-o file	Write output to file rather than a.out
-Wall	Enable all warning messages
-mv***x***	Build for Vx processor version (e.g., -mv5, -mv62)

LLVM Tools – Compiler Attributes

- Attributes
  - Instruct compiler to set special properties for functions or variables
  - Standard set of attributes supported

__attribute__((aligned(1024)))
Force alignment of a variable
__attribute__((always_inline))
Force the compiler to inline this function
__attribute__((noinline))
Force the compiler to not inline this function
__attribute__((packed))
Remove alignment and padding from structure
__attribute__((section("mysection"))
Force this symbol to a specific section
__attribute__((used))
Marks a function or variable as used so the compiler does not delete it

LLVM Tools – Compiler Intrinsics

- Hexagon Intrinsics
  - Functions that map directly to Hexagon assembly instructions
    - "To support efficient coding of the time-critical sections of a program (without resorting to assembly language), the C compilers support intrinsics which are used to directly express Hexagon processor instructions from within C code." - *Hexagon PRM*
  - To access intrinsics, include header file **`hexagon_protos.h`**

Hexagon assembly instruction:

```
Rx+=mpy(Rs,Rt):<<1:sat
```

Hexagon C intrinsic:

```
#include "hexagon_protos.h“
#include "hexagon_types.h"
HEXAGON_Vect32  varOutRx,  varInRx, varInRs,  varInRt;
varOutRx = Q6_R_mpyacc_RR_s1_sat (varInRx, varInRs, varInRt);
```

LLVM Tools – Compiler Intrinsics (cont.)

- Hexagon Intrinsics (cont.)

Class: XTYPE (slots 2,3)

Notes

- If saturation occurs during execution of this instruction (a result is clamped to either maximum or minimum values), then the OVF bit in the Status Register is set. OVF will remain set until explicitly cleared by a transfer to SR.

Intrinsics

`Rd=mpy(Rs,Rt.H):<<1:rnd:sat`	`Word32 Q6_R_mpy_RRh_s1_rnd_sat(Word32 Rs, Word32 Rt)`
`Rd=mpy(Rs,Rt.H):<<1:sat`	`Word32 Q6_R_mpy_RRh_s1_sat(Word32 Rs, Word32 Rt)`
`Rd=mpy(Rs,Rt.L):<<1:rnd:sat`	`Word32 Q6_R_mpy_RRl_s1_rnd_sat(Word32 Rs, Word32 Rt)`
`Rx+=mpy(Rs,Rt):<<1:sat`	`Word32 Q6_R_mpyacc_RR_s1_sat(Word32 Rx, Word32 Rs, Word32 Rt)`
`Rx-=mpy(Rs,Rt):<<1:sat`	`Word32 Q6_R_mpynac_RR_s1_sat(Word32 Rx, Word32 Rs, Word32 Rt)`

LLVM Tools – Assembler

- Translates Hexagon assembly language into object code
  - Generates ELF format object file and DWARF4 debugging information
- Supports Hexagon parallel instruction “packets”
  - Packet grouping rules
  - Packet dependency constraints
  - Packet ordering constraints (shuffle instructions to fit into correct slot)
- Supports common assembly directives:
  - .word 0xDEADBEEF → Data allocation
  - .p2align 12 → 4K alignment (2^{12})
  - .falign → Ensures following packet does not cross 4-word boundary
- The assembler should be invoked with the compiler driver
  - `hexagon-clang -mv60 -c file.S -o file.o`

LLVM Tools – Linker

- Auto-invoked by the compiler driver when generating an executable image
- Extensive linker script language
  - Supports custom section placement
  - Specify **`-T`** or **`--script`** to use a custom linker script
- LTO – Link Time Optimization
  - Available in Hexagon 8.0 tools and later
- Linker outputs are ELF files
- **`hexagon-link --section_start .init=0x1000 –o hello crt0.o init.o hello.o libc.a libqcc.a fini.o`**
- **`hexagon-link --help`** → **Provides help information**

LLVM Tools – Linker (cont.)

- Linking libraries
  - Setup library search path with **`-L`**
  - Specify libraries with **`-l`**
    - **`hexagon-link crt0.o init.o -L${MY_INSTALL_DIR}/mylibs hello.o –lmylib fini.o`**

- Objects and libraries are processed in order of appearance on the command line (in a single pass)
  - Use **`--start-group`** and **`--end-group`** to specify libraries to be checked repeatedly until no more symbols can be resolved
  - **`hexagon-link –o hello crt0.o init.o hello.o --start-group mylib1.a mylib2.a --end-group libc.a libqcc.a fini.o`**

LLVM Tools – Linker (cont.)

- Objects vs. Archives
  - Objects are created by the assembler, typically .o
  - The object file is the smallest linkable unit; the linker will always bring in an object file in its entirety
  - Archives are created by the archiver, typically .lib or .a
  - An archive is a collection of object files; the linker only brings in the objects that are needed for symbol resolution

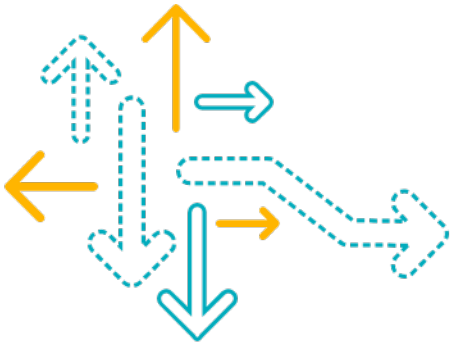
LLVM Tools – Hexagon Utilities

- **`--help`** can be used for all tools to show available options
- Readelf
  - Displays header, section, segment, symbol table information
    - **`hexagon-readelf -h hello.elf`**
    - **`hexagon-readelf -S hello.elf`**
    - **`hexagon-readelf -l hello.elf`**
    - **`hexagon-readelf -s hello.o`**
- Object file dump
  - Disassembly
    - **`hexagon-llvm-objdump -d hello`**
    - **`hexagon-llvm-objdump -S hello  (mix assembly and C)`**
- Object file symbols
  - View symbol addresses
    - **`hexagon-nm hello`**

```
00003060 T main
000044e0 T malloc
000057e0 T memcpy
```

LLVM Tools – Hexagon Utilities (cont.)

- Archiver
  - Create a library from multiple object files and generate an index into the archive
  - List the objects in an archive
  - No symbol resolution or relocation
    - **`hexagon-ar -ru mylib.a a.o b.o`**
- Object file stripper
  - Removes information from specified files
    - Remove debugging information
      - **`hexagon-strip hello.elf -g -o hello.stripped`**
    - Remove compiler generated local symbols
      - **`hexagon-strip hello.o -X -o hello.stripped.o`**
    - Remove all non-global symbols, but keep two selected symbols
      - **`hexagon-strip hello.elf -x -K xx_msg_  -K xx_err_msg_  -o  hello.stripped.elf`**
- List the symbols of an ELF file
  - List the symbols
    - **`hexagon-nm main.o`**
  - List the symbols, with demangling
    - **`hexagon-nm -C main.o`**



Software Development Tools: Simulator

Simulator – Overview

- Features
- Usage

Simulator – Features

- Instruction set simulator
- Cycle-approximate simulation

- Models I-cache, D-cache, and L2 cache behavior
- Models memory delays with simple bus model
- Models processor-level timers as external co-simulation models

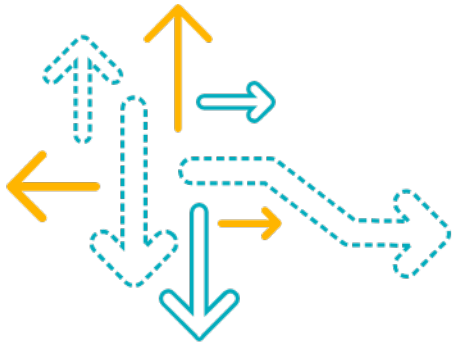
- Supports co-simulations and system simulations via a robust API
- Trace statistics for instruction execution, processor stalls, and bus, cache, and memory accesses

- Remote file I/O through ARM® Angel interface
- LLDB and TRACE32 (via MCD interface) debugging using the simulator as the execution engine
- Rich profiler support (gprof, proftool)

Simulator – Usage

- **`hexagon-sim [option] executable_file`**
- **`hexagon-sim -- factorial.elf 10`** → **pass arguments to EXE**

Flag	Use
--timing	Simulate cache misses, stalls, etc.
-mv*xy*	*x* → processor type; *y* → cache model e.g., -mv65a_1024 → mv65, 4-thread, 16K L1$, 1M L2$ --timing required if caches are to be profiled
--buspenalty 20	Modify default bus delay (in pcycles)
-G <port number>	Port number specifies debug connection port
--cosim_file cosims	Specify co-simulation modules
--pmu_statsfile <name>	Generate a PMU statistics file with specified name



Software Development Tools: Debugger

Debugger – Overview

- **`hexagon-lldb`**
- TRACE32 (not discussed in this presentation)

Debugger – hexago-lldb

`hexagon-lldb` offers the following features:

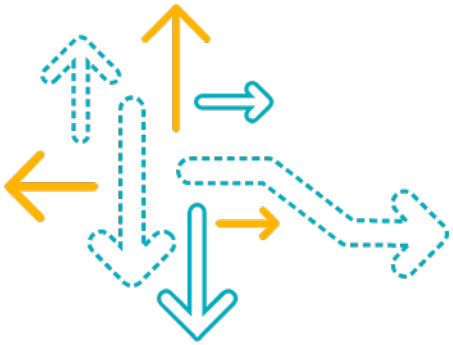
- Debug Hexagon programs running on a simulator
  - Start simulation of an executable
  - Attach to a running process
  - Break program execution on specified conditions
  - Analyze state of simulation when program is stopped
  - Change state of simulation by modifying registers or variables
- C/C++ debugging
- Multi-threaded debugging
- Remote protocol/debug server
- Extensible Python scripting

Debugger – hexago-lldb (cont.)

- **`hexagon-lldb [`*`options`*`] `*`executable_file`***
- **`hexagon-lldb --args factorial.elf 10` → pass arguments to EXE**
- **`hexagon-lldb --help` → Provide hexagon-lldb help information**

- Common commands
  - “run” → start the simulator with your executable and connect
  - “start” → start a simulator, connect, and run to main
  - “b <func>” → set a breakpoint at function “func”
  - “s” → step to the next instruction
  - “n” → step over (next)
  - “c” → continue
  - “re r” → view registers
  - “re r <regname>” → view a specific register (e.g., ssr, pmucnt0)
  - “re r –s 3” → view vector registers
  - “re w <regname> <value>” → write a value to a register
  - “tlb” → view tlb information
  - “pagetable” → view pagetable information

- LLDB to GDB comparison
  - http://lldb.llvm.org/lldb-gdb.html



Software Development Tools: Profiling

Profiling – Overview

- **`hexagon-gprof`** → flat profile, call graph
- **`hexagon-profiler`** → gather/analyze stall trace data

Profiling – hexagon-gprof

- **`hexagon-sim --profile` *`other_simulator_args`***
  - Similar to Linux gprof, simulator produces gmon files at runtime
- **`hexagon-gprof` *`image_file gmon_file`***
  - Stand-alone: single gmon output file per hardware thread
    - **`hexagon-gprof app1 gmon.t_0`** (single thread)
    - **`hexagon-gprof app1 gmon.t_*`** (all threads)
  - RTOS: separate gmon files per software thread
    - **`gmon.t_`*`threadName_threadID`***
  - Special gmon files for kernel and cache-miss profiling
    - **`gmon.kernel`**, **`gmon.icachemiss.t_`*`threadName_threadID`*** etc.
- Flat profile
  - Total amount of execution time consumed by each function
  - Useful for identifying candidate functions for optimization
- Call graph
  - List all callers and subroutines for each function
  - Useful for identifying individual components of an algorithm which could benefit from optimization

Profiling – hexagon-gprof (cont.)

```
$ make profile
hexagon-gprof mandelbrot gmon.t*
Flat profile:

  %     cumulative    self              self       total
 time    cycle(s)   cycle(s)  calls    Mc/call    Mc/call  name
 48.94   11539631   11539631      0                        compute_fractal
 24.86   17401194    5861563      2       2.93       2.93  thread_join
  5.69   18742520    1341326      0                        fputc
  3.81   19641360     898840 40811       0.00       0.00  _Lockfilelock.extracted_region
  3.64   20498570     857210 40803       0.00       0.00  _Lockfilelock
```

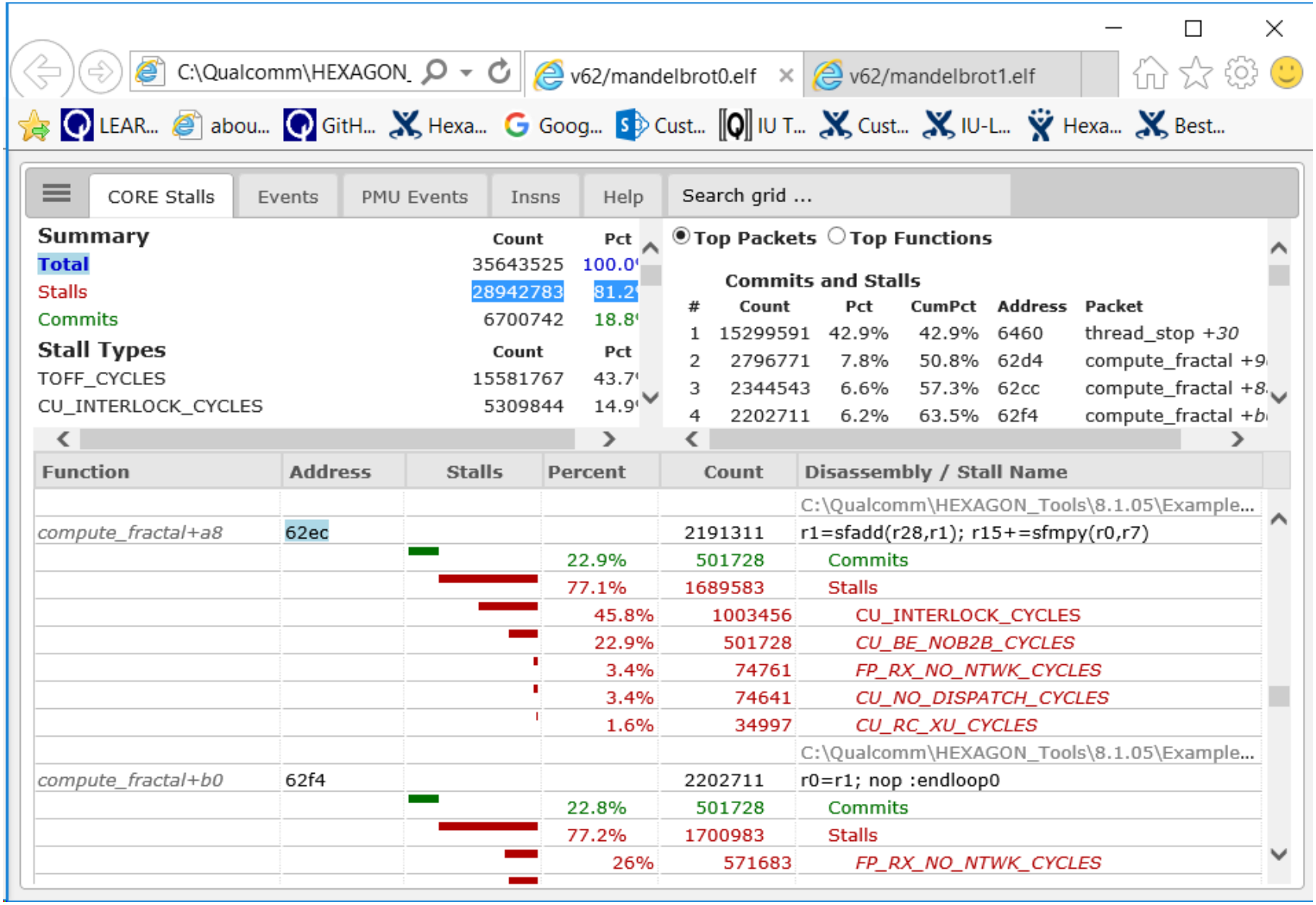
```
Call graph:

index   % time        self   children    called     name
-----------------------------------------------------------------
[1]        48.9   11539631          0       0       compute_fractal [1]
-----------------------------------------------------------------
[2]        28.3        392    6677175       0       hexagon_start_main [2]
                      1319    6675856     1/1       __libc_start_main [3]
-----------------------------------------------------------------
                      1319    6675856     1/1       hexagon_start_main [2]
[3]        28.3       1319    6675856       1       __libc_start_main [3]
                    691343    5982961     1/1       main [4]
                       747        175     1/1       _init [32]
                       546         84     1/1       atexit [35]
-----------------------------------------------------------------
```

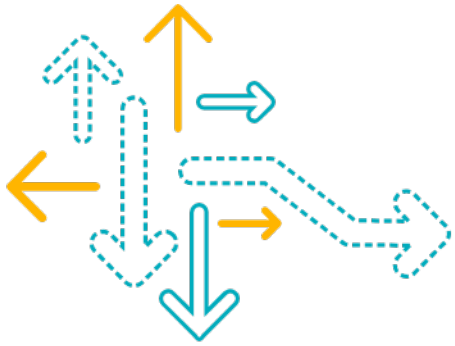
Profiling – Proftool

- **`hexagon-sim -mv60 --timing --packet_analyze stats.json`** ***`other_sim_args`***
  - **`stats.json`** → Intermediate output file
- **`hexagon-profiler --packet_analyze stats.json --elf=binary.elf -o output0.html`**

- Open generated .html file in a web browser
  - Top Left pane
    - Shows cycle information including a breakdown of stalls vs. commit cycles
    - Menu buttons and check boxes that control the other two panes
  - Top Right pane
    - Shows list of packets ordered by cycles consumed
  - Bottom pane
    - Shows the disassembly view

Profiling – Proftool (cont.)





Software Development Tools: Documentation and Examples

Documentation and Examples – Documentation

Hexagon LLVM Tools 8.x Document Bundle

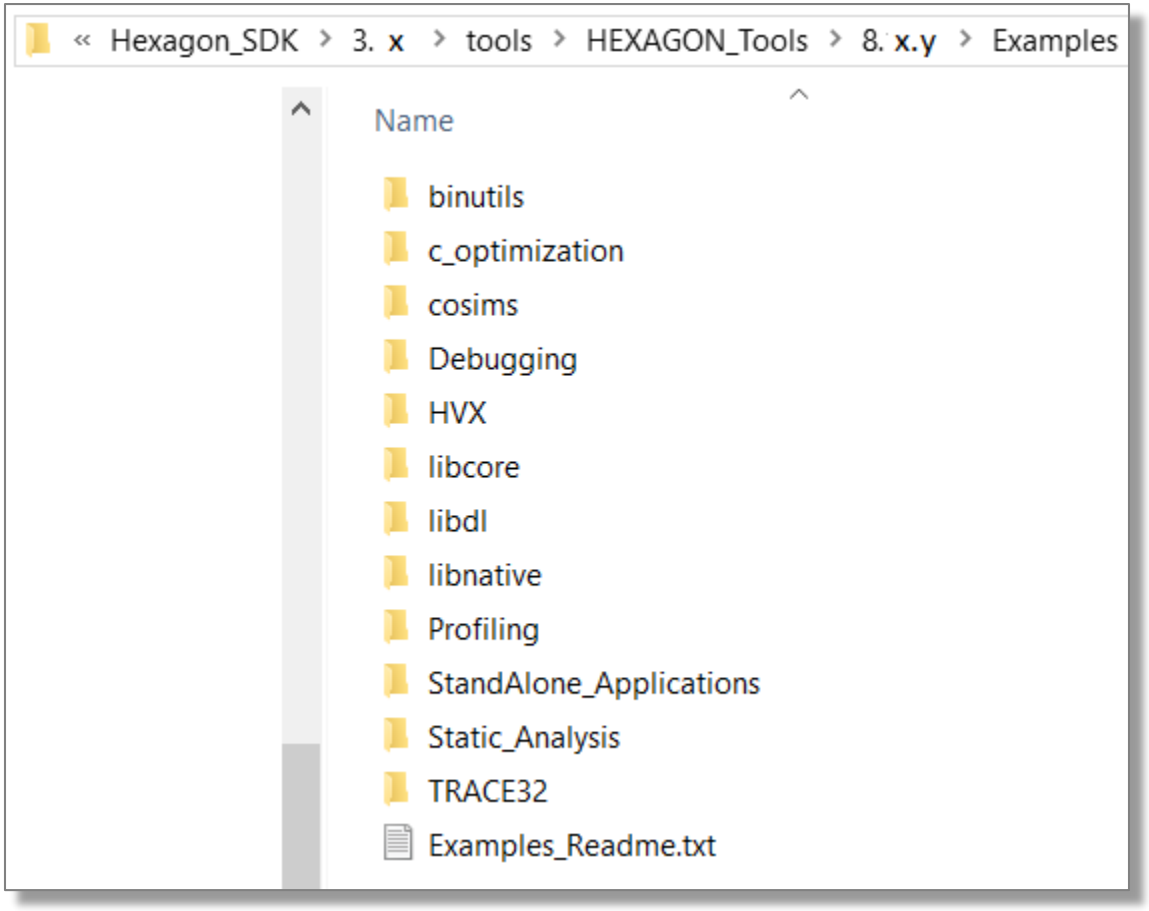
- Hexagon LLVM C/C++ Compiler User Guide (80-VB419-89)
- Hexagon C Library User Guide (80-N2040-13)
- Hexagon C++ Library User Guide (80-N2040-14)
- Hexagon Utilities User Guide (80-N2040-15)
- Hexagon Simulator User Guide (80-N2040-17)
- Hexagon Simulator System API User Guide (80-N2040-18)
- Hexagon LLDB Debugger User Guide (80-N2040-31)
- Hexagon gprof Profiler User Guide (80-N2040-29)
- Hexagon Profiler User Guide (80-N2040-10)
- Hexagon Code Coverage Profiler User Guide (80-N2040-20)
- Hexagon Resource Analyzer User Guide (80-N2040-21)
- Hexagon TRACE32 User Guide (80-N2040-28)
- Hexagon Stand-alone Application User Guide (80-N2040-22)
- Hexagon Application Binary Interface Specification (80-N2040-23)

Location:

Linux: <install location>/Qualcomm/HEXAGON_Tools/<version>/Documents/

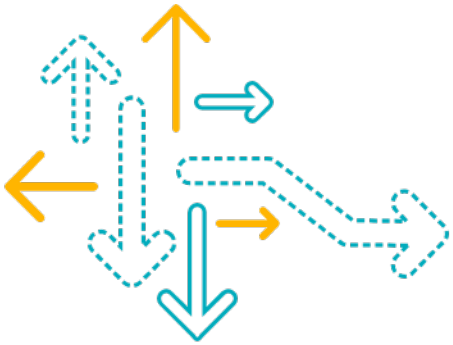
Windows: C:\Qualcomm\ Hexagon_SDK\<version>\tools\HEXAGON_Tools\<version>\Documents

Documentation and Examples – Examples



Documentation and Examples – Examples (cont.)

binutils	Use cases of Hexagon utilities
c_optimization	Demonstrate C optimization techniques
cosims	Demonstrate the use of co-simulation modules
Debugging	Using lldb to debug a crash
HVX	Lots of HVX examples
libcore	AudioProc, sfpFFT,ImagProc, sigProc…
libdl	Create and open a dynamic library
libnative	Build and execute instruction intrinsics
Profiling	Gprof, proftool use cases
StandAlone_Applications	Mandelbrot, tcm, tlb use cases
Static_Analysis	Usage of clang static analyzer
TRACE32	Connect T32 to hexagon simulator



Questions?

https://createpoint.qti.qualcomm.com