Qualcomm Technologies, Inc.

Qualcomm® Hexagon™ cDSP Overview for SM8250

80-PK882-38 Rev. B

Confidential and Proprietary – Qualcomm Technologies, Inc.

Revision History

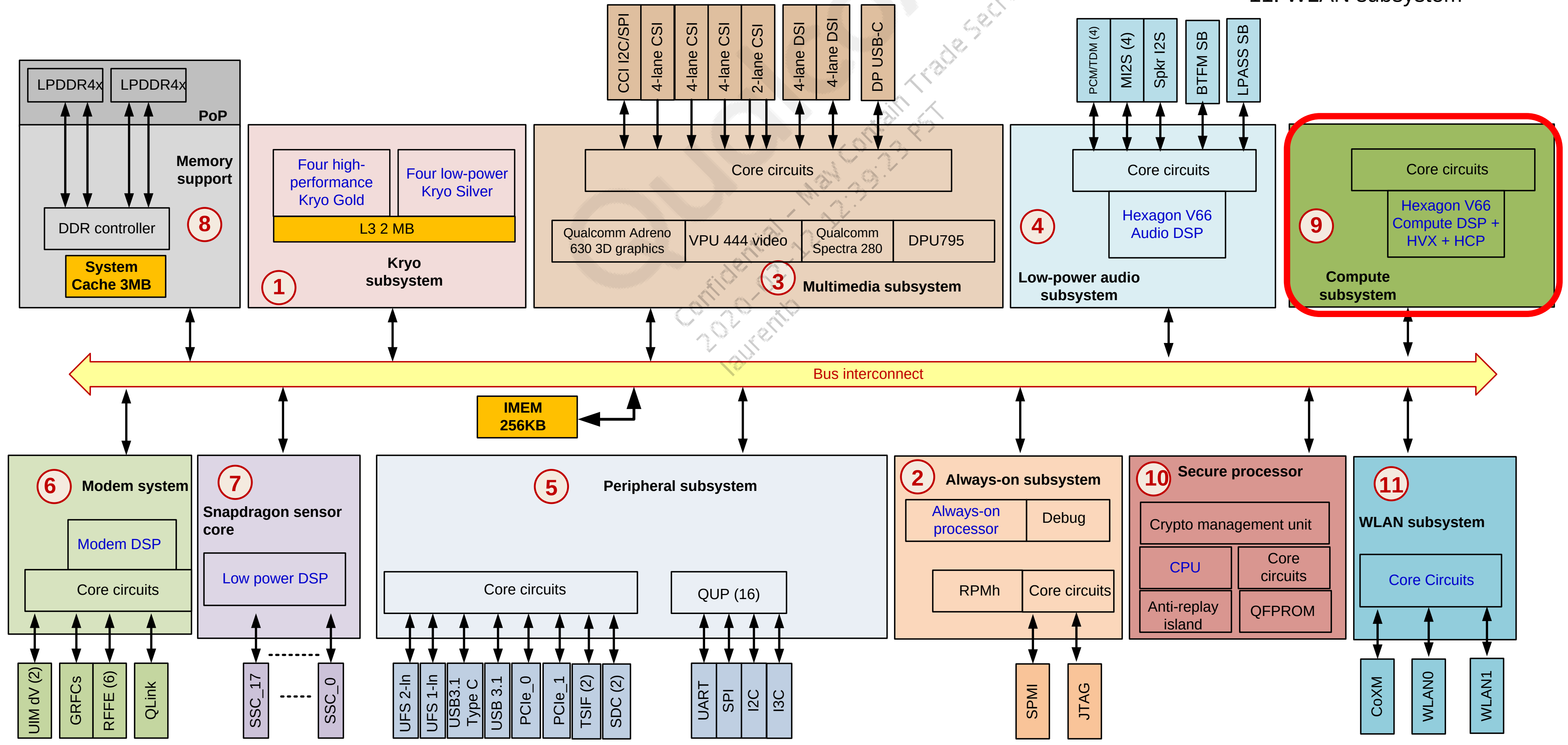
Revision	Date	Description
A	July 2019	Initial release
B	July 2019	• Standardized path to “${HEXAGON_SDK_ROOT}” • Minor editorial changes for consistency

Contents

Qualcomm Hexagon Compute DSP (cDSP)

Architecture Overview

Major architectural blocks:
1. Kryo subsystem
2. Always-on subsystem (AOSS)
3. Multimedia subsystem (MMSS)
4. Low power audio subsystem (LPASS)
5. Peripheral subsystem (PSS)
6. Modem system
7. Snapdragon sensor core
8. Memory support and the bus system that supports all major blocks
9. Compute subsystem
10. Secure processor
11. WLAN subsystem



cDSP – HVX Use Cases

- cDSP is dedicated for compute applications
- Ensures improved image processing, computer vision, camera streaming, and machine learning

Image enhancement – Camera, Still, Video

Computer Vision and Extended Reality

Video

Camera streaming

Machine learning

Qualcomm® Snapdragon™ DSP Evolution

SDM820

- Compute DSP
  - Hexagon V60
  - 825 MHz, 512 KB L2
  - Dual-HVX[1]
  - Imaging and AI
- Audio DSP
  - Shared with Compute DSP

Shared

- Sensors DSP
  - Hexagon V60
  - 700 MHz
  - 512 KB L2$
  - Low-Power Island
  - “Always Aware Hub”

SDM835

- Compute DSP
  - Hexagon V62
  - 900 MHz, 512 KB L2
  - Dual-HVX
  - Imaging and AI
- Audio DSP
  - Shared with Compute DSP

Shared

- Sensors DSP
  - Hexagon V62
  - 600 MHz
  - 1 MB L2$ or L2 TCM
  - Low-Power Island
  - “Always Aware Hub”

SDM845

- Compute DSP
  - Hexagon V65
  - 1.2 GHz, 512 KB L2
  - Dual-HVX
  - Scatter-Gather
  - 256 KB VTCM
  - Enhanced CNN support
  - Imaging and AI
- Audio DSP
  - Hexagon V65
  - 1.2 GHz, 512 KB L2$
- Sensors DSP
  - Hexagon V65
  - 1.2 GHz
  - 512 KB L2$, 1 MB L2 TCM
  - Low-Power Island
  - “Always Aware Hub”

SM8150

- Compute DSP
  - Hexagon V66
  - 1.5 GHz, 1 MB L2$
  - Quad-HVX
  - Scatter-Gather
  - 256 KB VTCM
  - Enhanced CNN support
  - Imaging and AI
- Audio DSP
  - Hexagon V66
  - 1.5 GHz, 512 KB L2$
- Sensors DSP
  - Hexagon V66
  - 1.0 GHz
  - 512 KB L2$, 1 MB L2 TCM
  - Low-Power Island
  - “Always Aware Hub”

SM8250

- Compute DSP
  - Hexagon V66
  - 1.5 GHz, 1 MB L2$
  - Quad-HVX
  - Scatter/Gather
  - 256 KB VTCM
  - Enhanced CNN support
  - Imaging and AI
- Audio DSP
  - Hexagon V66
  - 1.5 GHz
  - 1 MB L2$, 1 MB L2 TCM
  - Low-Power Island
- Sensors DSP
  - Hexagon V66
  - 1.0 GHz
  - 512 KB L2$, 1 MB L2 TCM
  - Low-Power Island
  - “Always Aware Hub”

[1] Qualcomm® Hexagon™ Vector eXtensions (HVX)

Hexagon DSP Evolution – Compute DSP

	Compute DSP		Compute DSP	
	Hexagon V65 (SDM845)	**HVX V65 (SDM845)**	**Hexagon V66 (SM8150/SM8250)**	**HVX V66 (SM8150)**
Threads/Core	4 scalar threads	+ 2 128-byte vector threads	4 scalar threads	+ 4 128-byte vector threads
Processor Clock	950 MHz Nominal (1200 MHz Turbo)	+ 950 MHz Nominal (+ 1.2 GHz Turbo)	1180 MHz Nominal (1500 MHz Turbo)	+ 1180 MHz Nominal (+ 1500 MHz Turbo)
Arithmetic	Fixed and Floating Point	Fixed Point	Fixed and Floating Point	Fixed Point
Caches	L1 Instruction: 16 KB L1 Data: 16 KB L2: 512 KB	Same L2	L1 Instruction: 16 KB L1 Data: 16 KB L2: 1024 KB	Same L2
Performance • **Peak OPS** • **Dhrystone** • **Coremark**	Performance • 8 MMAC/MHz (16-bit fix point) 16 MMAC/MHz (8-bit fix point) • 4 MFLOP (SP)/MHz • 1T: 3.11 DMIPS/MHz 2T: 4.46 DMIPS/MHz 4T: 7.58 DMIPS/MHz	Performance Dual-HVX • + 64 MMAC/MHz (16-bit fix point) • + 256 MMAC/MHz (8-bit fix point) [+ 512 MMAC/MHz w/NN_Lib] • + 256 Mops/MHz (16-bit fix point)	Performance • 16 MMAC/MHz (16-bit fix point) 32 MMAC/MHz (8-bit fix point) • 8 MFLOP (SP)/MHz • 1T: 3.65 DMIPS/MHz 4T: 8.43 DMIPS/MHz	Performance Quad-HVX • + 128 MMAC/MHz (16-bit fix point) [+ 256 MMAC/MHz with NN_Lib] • + 512 MMAC/MHz (8-bit fix point) [+ 1024 MMAC/MHz with NN_Lib] • + 512 Mops/MHz (16-bit fix point)
Architecture	Simultaneous Multi-Threading (SMT)		Simultaneous Multi-Threading (SMT)	
DDR Bus Interface	18 GB per second Direct interface to DDR		18 GB per second Direct interface to DDR	
Instruction/System Enhancements	Multimedia enhanced Co-processor capability Automatic Power Collapse (APC)	Hexagon Vector eXtensions (HVX) Dual-clusters Scatter-Gather, (256 KB VTCM), DMA, UBWC	Multimedia enhanced Dual-Clusters Co-processor capability Automatic Power Collapse (APC)	Hexagon Vector eXtensions (HVX) Dual-Clusters Scatter-Gather, (256 KB VTCM), DMA, UBWC
	Scalar DSP		***Scalar DSP***	

cDSP – Frequency

Voltage setting vs. maximum frequency (MHz)	SM8250[1]
TURBO_L1	1497.6
TURBO	1401.6
NOM_L1	1324.8
NOM	1171.2
SVS_L1	960
SVS	768
LowSVS	576
MinSVS	364.8

[1] New instructions, vector TCM, and scatter-gather allow for comparable tasks at lower frequency requirements. Values are subject to change.

cDSP – Feature Summary

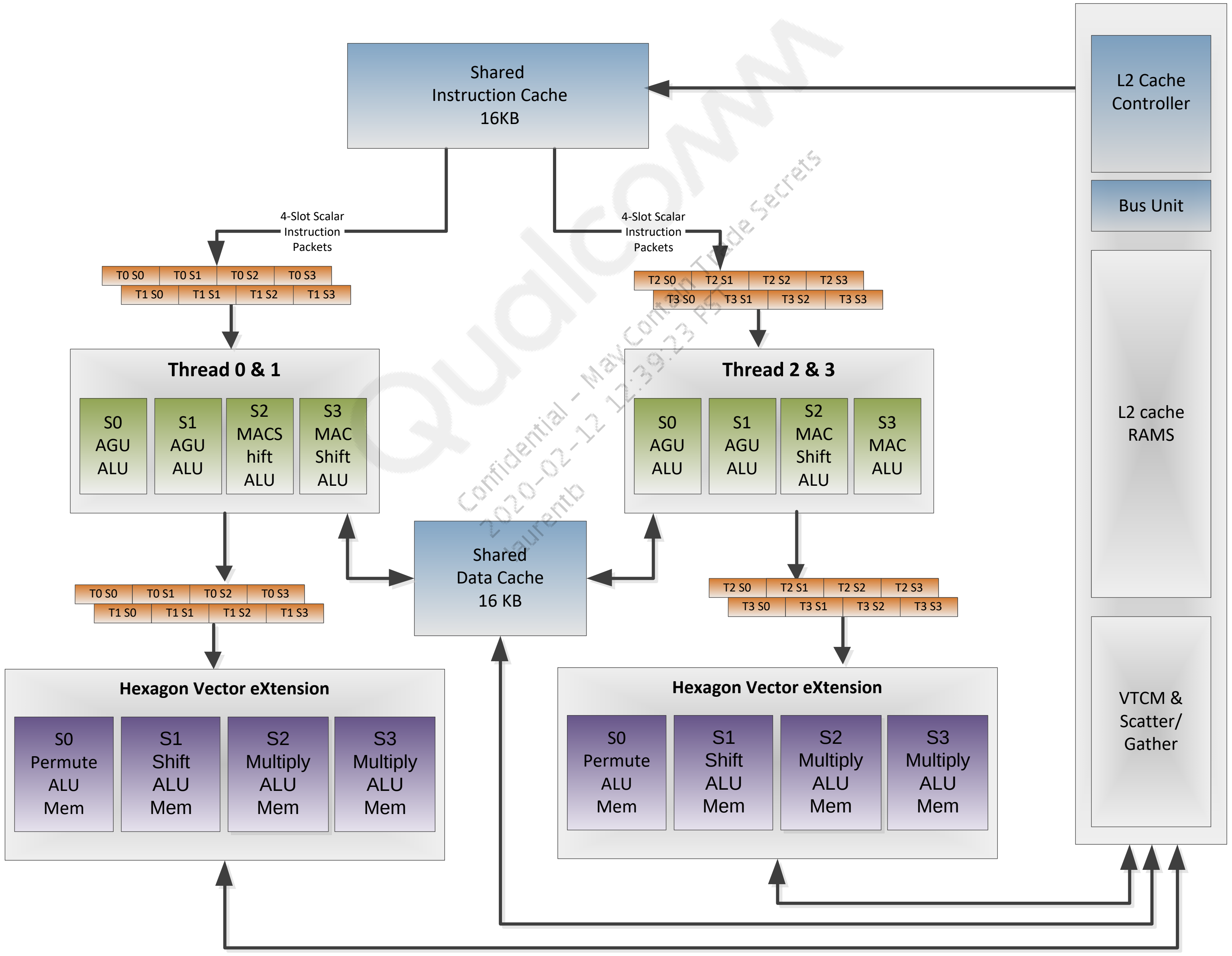
Feature	Description
Compute Resource Manager	Framework that provides APIs to request and release compute resources (such as, HVX, VTCM, hardware threads, memory buses) and enables serialization
Dynamic Secure PD	Dynamic loading support in securePD
Debugging Tools	Remote debugger, user PD crash log, simulator debug enhancements
Profiling Tools – Hexagon Trace Analyzer	Software trace analysis tool provides increased granularity at function and packet level
Unsigned protection domain (PD)	Support for signature-free dynamic shared objects
Clock voting	Updated voltage corners and clock voting considerations
Sub-CA	Production signing mechanism that replaces CASS or USB token signing
HexagonNN	Enhanced Open HexagonNN API support for offloading graphs to cDSP

cDSP Hardware Architecture

Hexagon cDSP with HVX

- Interleaved multi-thread VLIW DSP
- Combines best of DSP with general purpose processor ease-of-programming
- Provides excellent control code and signal processing
- Focuses on throughput and efficiency instead of peak single-thread performance
- HVX is included in cDSP
- Instruction extensions to Hexagon V66 processor architecture support vector operations on 1024-bit wide data
- Suitable for high performance imaging, compute, and machine learning applications

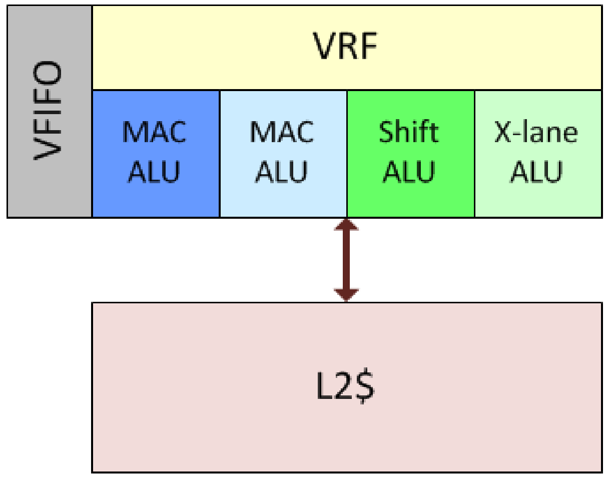
cDSP – HVX



Shared Instruction Cache 16KB
4-Slot Scalar Instruction Packets
T0 S0 T0 S1 T0 S2 T0 S3
T1 S0 T1 S1 T1 S2 T1 S3
4-Slot Scalar Instruction Packets
T2 S0 T2 S1 T2 S2 T2 S3
T3 S0 T3 S1 T3 S2 T3 S3
Thread 0 & 1
S0 AGU ALU
S1 AGU ALU
S2 MACS hift ALU
S3 MAC Shift ALU
Thread 2 & 3
S0 AGU ALU
S1 AGU ALU
S2 MAC Shift ALU
S3 MAC ALU
Shared Data Cache 16 KB
T0 S0 T0 S1 T0 S2 T0 S3
T1 S0 T1 S1 T1 S2 T1 S3
T2 S0 T2 S1 T2 S2 T2 S3
T3 S0 T3 S1 T3 S2 T3 S3
Hexagon Vector eXtension
S0 Permute ALU Mem
S1 Shift ALU Mem
S2 Multiply ALU Mem
S3 Multiply ALU Mem
Hexagon Vector eXtension
S0 Permute ALU Mem
S1 Shift ALU Mem
S2 Multiply ALU Mem
S3 Multiply ALU Mem
L2 Cache Controller
Bus Unit
L2 cache RAMS
VTCM & Scatter/ Gather

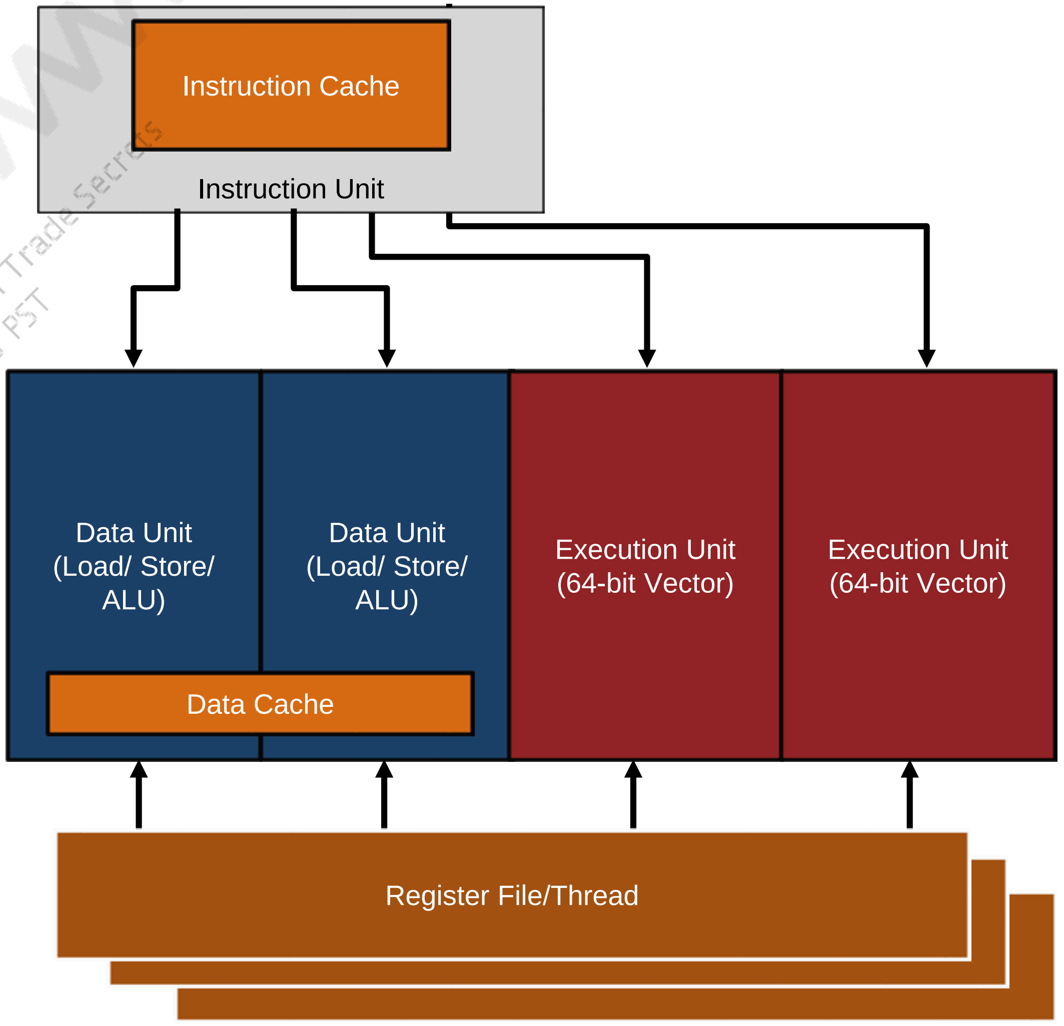
cDSP – HVX (cont.)

- Four vector execution resources
  - Two multiply oriented
  - One shift-oriented
  - One permute-oriented
- Simple instructions supported on all resources
  - Add, sub, transfer, logicals
- One load resource + one store resource
  - Access L2 memory directly
  - Keep consistent with L1D$
- VLIW model
  - Compiler/assembly coder chooses which instructions execute in parallel



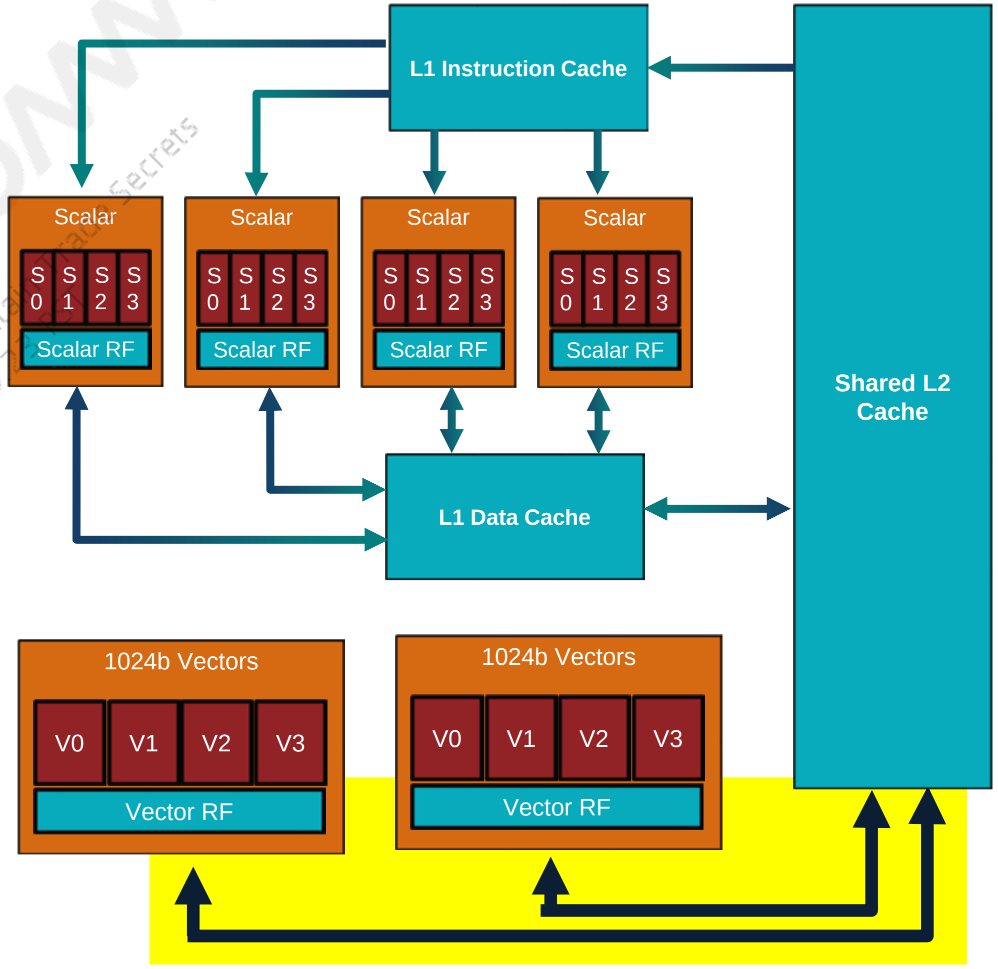
cDSP Hexagon Scalar Core

- Orthogonal 4-issue VLIW
  - 1 to 4+ instructions per cycle
- Dual 64-bit load/store units
  - 8/16/32/64-bit load/store, 32-bit scalar arithmetic
- Dual 64-bit vector units
  - 16/32/64-bit vectorized MPY/ALU/SHIFT/FP, permute, bit manipulation, up to 8 MAC/cycle total/2 SP FMA
- Dual branch units
  - Two prioritized compound compare-jumps per cycle
- Unified vector/scalar registers
  - Expands spectrum of code that can be vectorized
- Rich instruction set
- Multiple threads


Instruction Cache
Instruction Unit
Data Unit (Load/ Store/ ALU)
Data Unit (Load/ Store/ ALU)
Execution Unit (64-bit Vector)
Execution Unit (64-bit Vector)
Data Cache
Register File/Thread

HVX Architecture – Memory

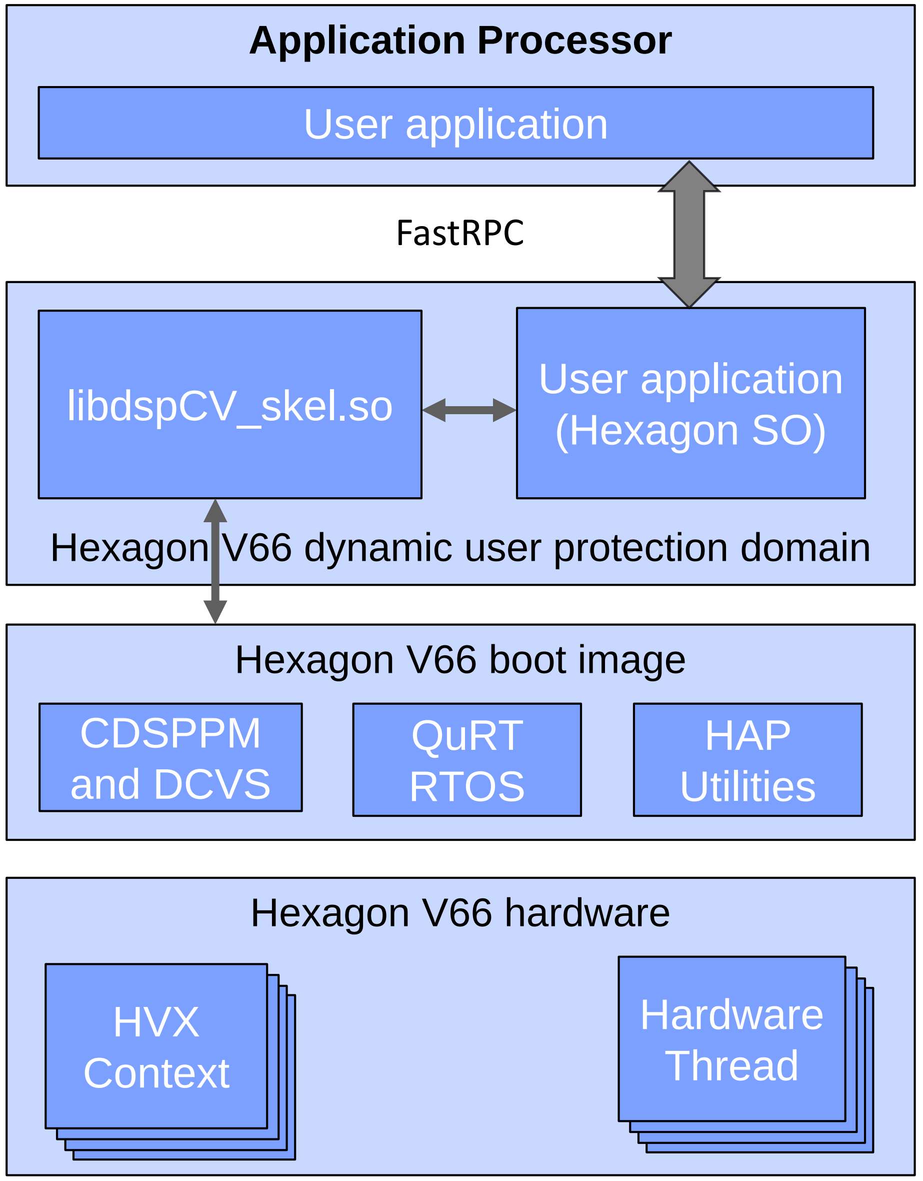
- L2 is first-level memory for vector units
  - Large primary memory to hold image data reduces tiling overheads seen on small L1
  - Single cycle load to use
  - Supports full bandwidth
  - Simplifies programming
- L1 and L2 are hardware coherent
- Streaming prefetch from DDR to L2
- Vector units support a variety of load/store instructions
  - Unaligned
  - Per-byte conditional



cDSP Software Architecture

cDSP Software Architecture

- Basic compute software architecture maintained from previous non-HVX to HVX targets
- Minor changes to programming model
  - Assembly code can contain HVX instructions
  - C code can contain HVX intrinsics
  - HVX context automatically locked to software thread when thread issues HVX instructions
    - QuRT™ Software saves and restores HVX register set during context switch
  - libdspCV_skel.so provides only multi-threading support that can be directly invoked from DSP side
  - DCVS v2 APIs provided for clock and power management



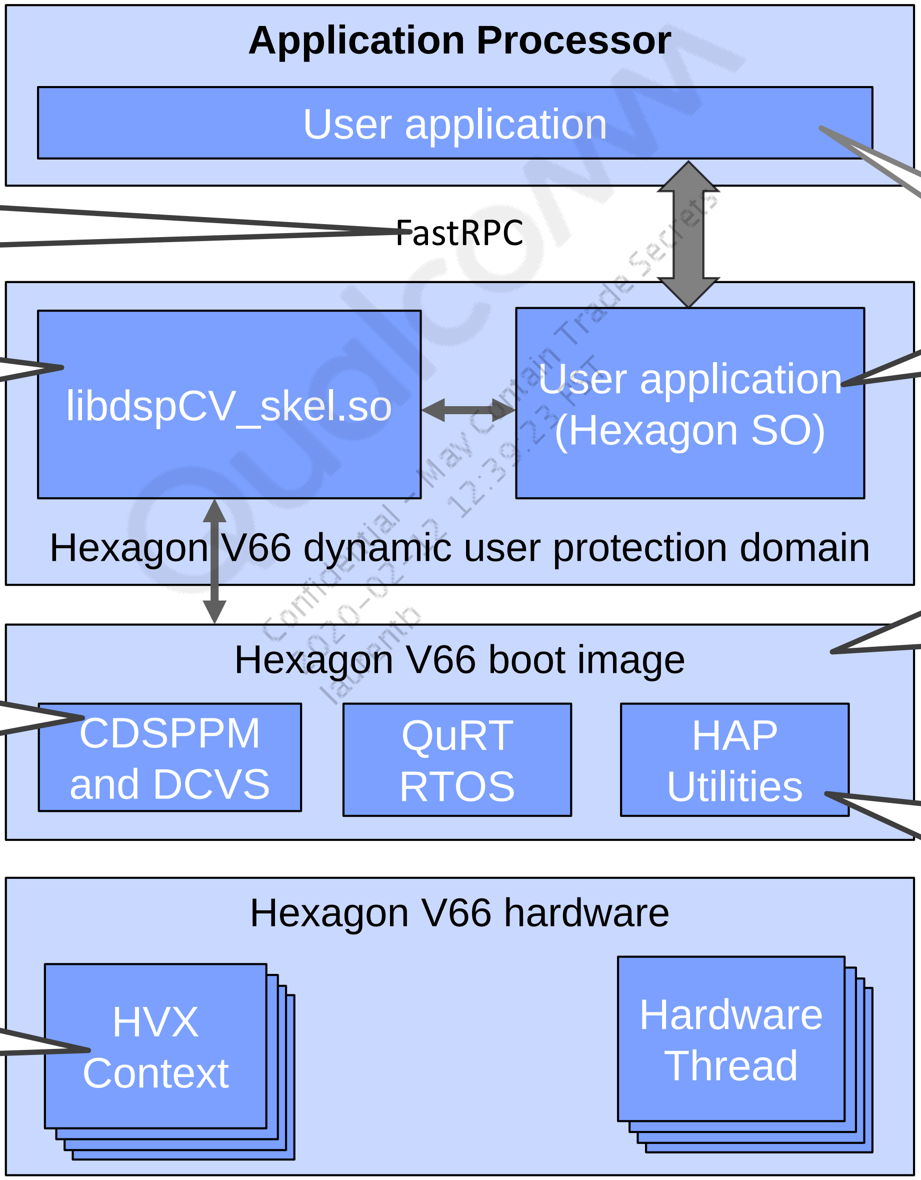
cDSP Software Architecture (cont.)

FastRPC interface between user application and Hexagon libraries with HVX implementations

Wraps multi-threading callback interface

Aggregates clock votes and HVX power votes. Drives hardware accordingly

To be used, must be locked to a hardware thread by RTOS



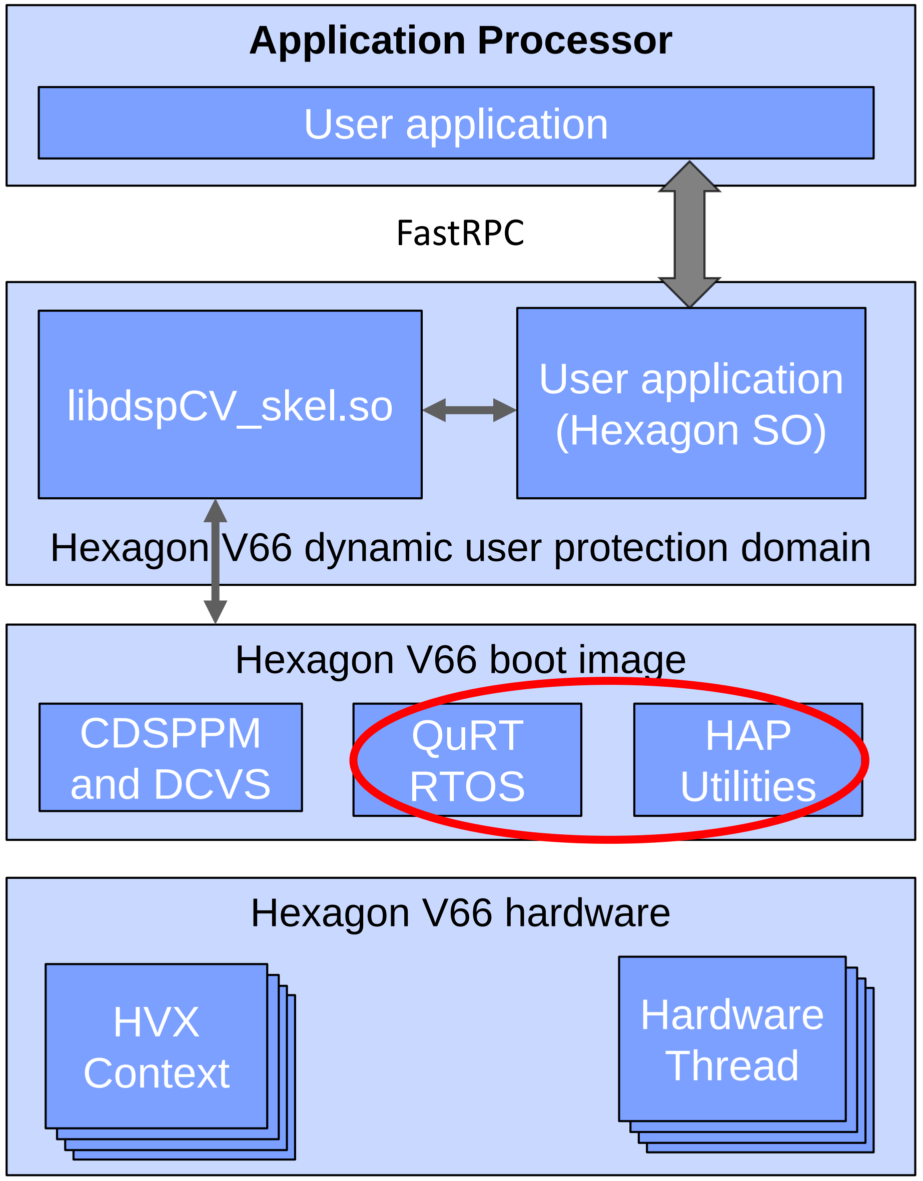
User application partitioned between application processor and Hexagon

Assigns HVX contexts to software threads (upon request) and software threads to hardware threads

Provides access for setting clock and bus votes together with other DCVS parameters

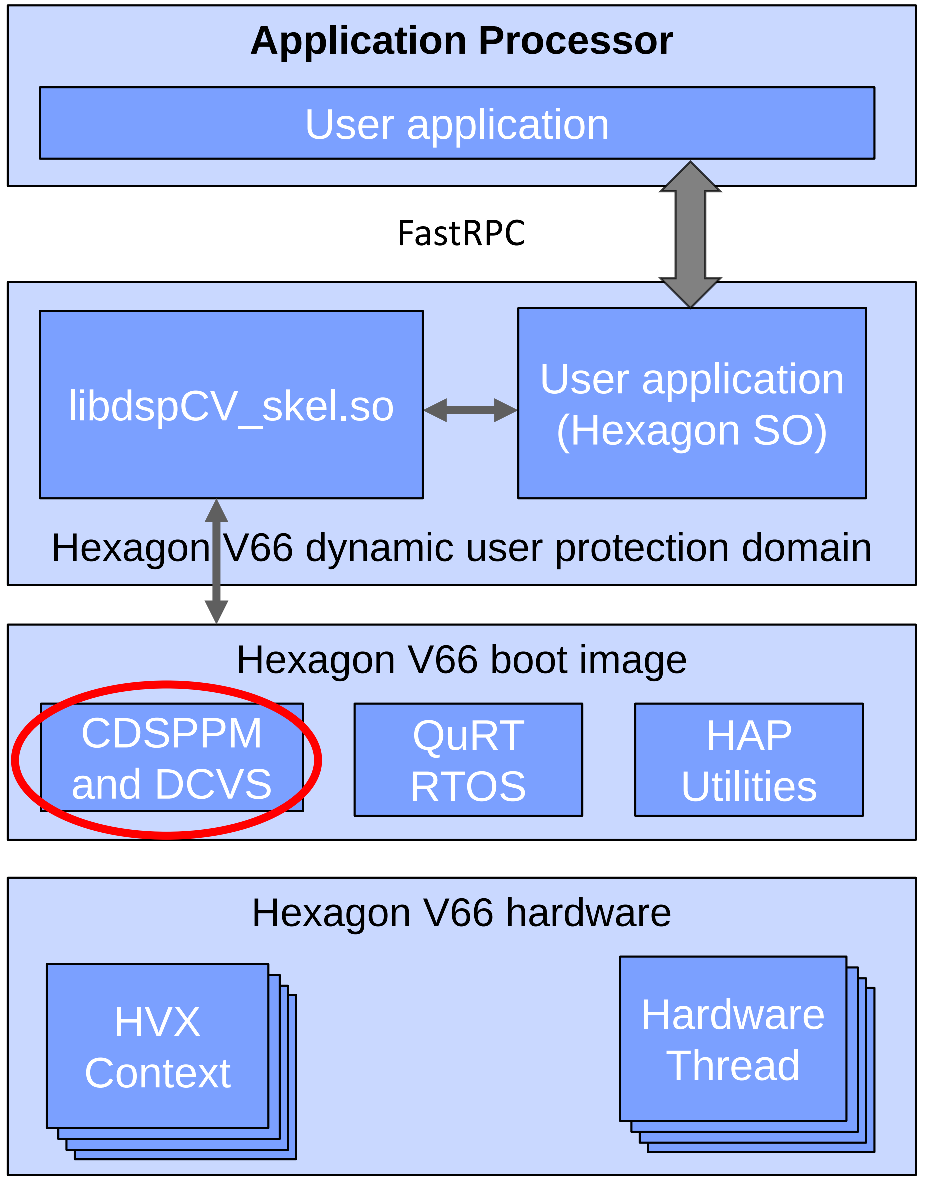
cDSP Software Architecture – QuRT RTOS

- Schedules software threads onto hardware threads, based on thread priority
- Provides APIs for HVX control
  - Reserves number of HVX contexts for user protection domain (held until reservation is cancelled)
  - Locks/unlocks an HVX context to requesting software thread, with specified1024-bit mode
  - Queries for HVX capabilities of target



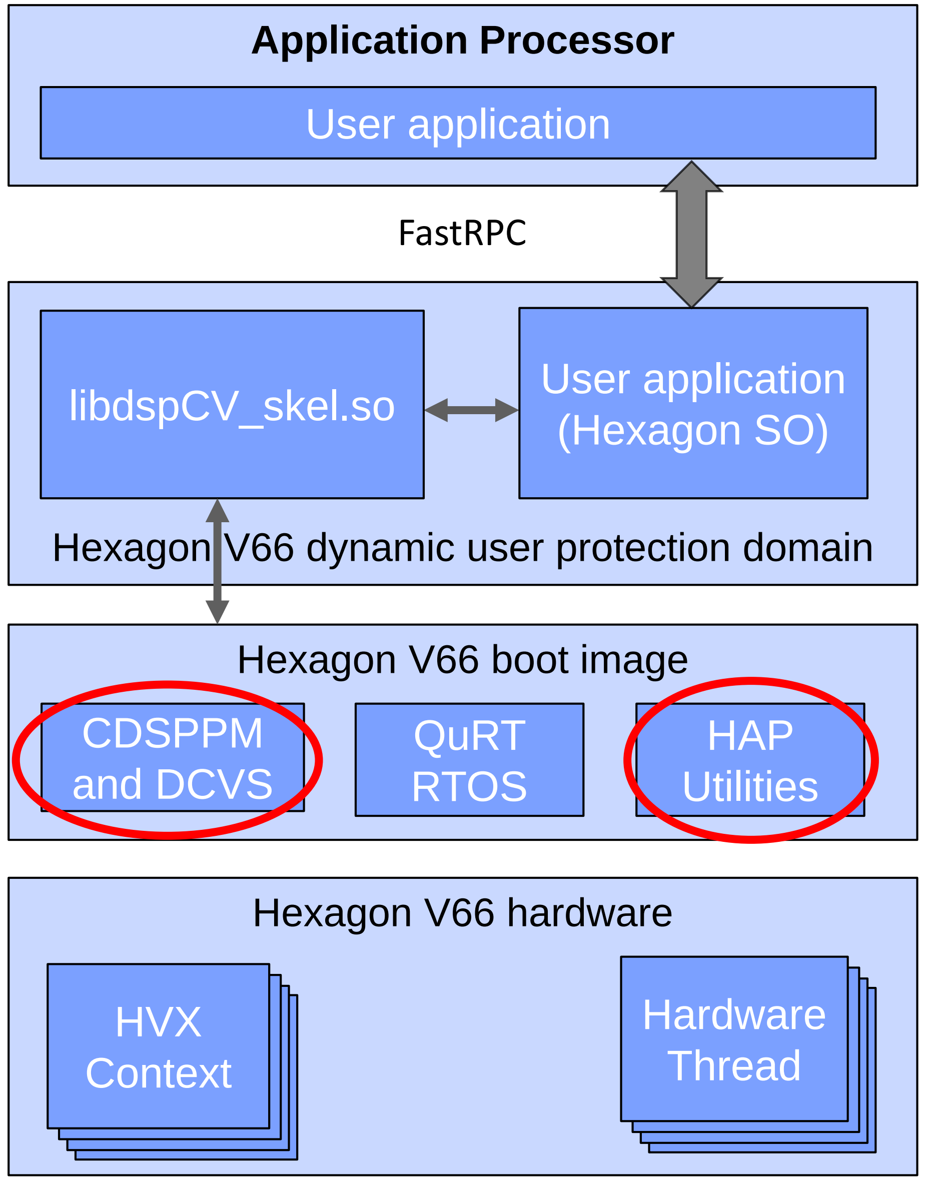
cDSP Software Architecture – cDSP Power Management (CDSPPM) and DCVS

- cDSP and bus clocks are managed within cDSP by combination of voting (CDSPPM) and background DCVS task
  - Voting establishes initial clocks, and floor clocks are guaranteed throughout voting use case life cycle
  - CDSPPM aggregates client votes for Hexagon core clock and bus clock to DDR. Sets default clock to NOM (Nominal) frequency
  - DCVS monitors processor loading and might raise or lower clocks to meet real-time needs with lowest possible power



cDSP Software Architecture – HAP Utilities

- HAP Utility APIs provide access for setting clock and bus votes together with other DCVS parameters



cDSP Power Management – DCVS v2 API (HAP Utility API)

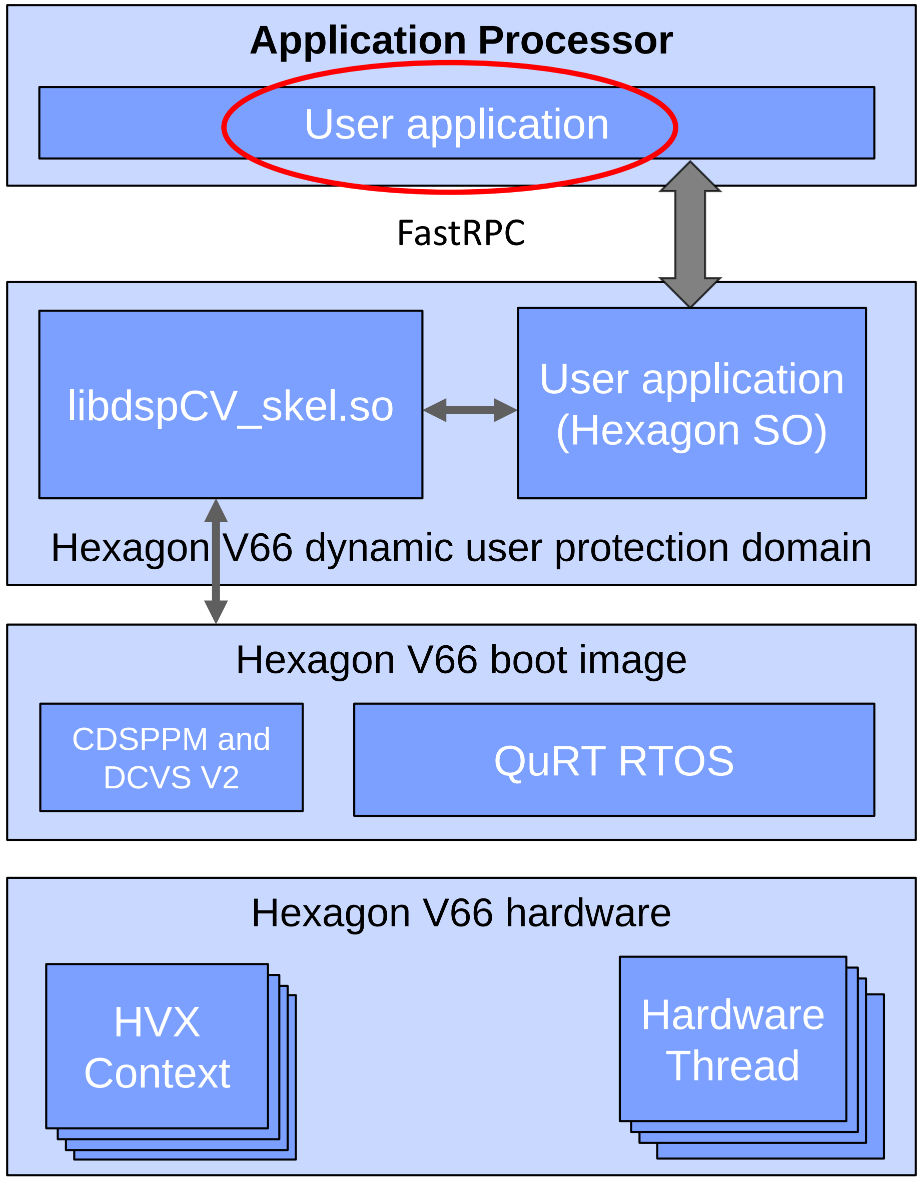
- **`HAP_power_set (void* context, HAP_power_request_t* request)`** is used to vote and remove DCVS_v2 vote after the use-case is complete
- **`HAP_power_set_DCVS_v2` (request type)** provides the ability to:
  - Enable/disable DCVS
  - Set DCVS mode or options, such as:
    - HAP_DCVS_V2_POWER_SAVER_MODE
    - HAP_DCVS_V2_POWER_SAVER_AGGRESSIVE_MODE
    - HAP_DCVS_V2_PERFORMANCE_MODE
  - Set sleep latency vote in microseconds. Sleep latency is the amount of time within which DSP must wake up. Based on value passed, DSP decides power collapse state it can enter. Typically set to 1000 microseconds
  - Set DCVS algorithm parameters, such as:
    - Target, Min, and Max voltage corners
    - HAP_DCVS_VCORNER_TURBO_PLUS and HAP_DCVS_VCORNER_TURBO_MAX supported on 8150 and later
      - HAP_DCVS_VCORNER_TURBO_PLUS
      - HAP_DCVS_VCORNER_TURBO_MAX is maximum possible corner defined for maximum performance
    - Set CPU L3 clock frequency to requested corner
- **`context`** provides multiple clients voting from same process, to use unique value per client and set respective DCVS values

  Note: Votes across all contexts, including NULL, are aggregated to determine final clock value. By default, FastRPC votes Nominal with NULL context. Prior to setting its unique value, client for a given process should remove vote for NULL context and then set its vote. Failure to do so can result in higher clock than expected because NULL vote is added to client vote, resulting in higher power consumption.

  - For more information, see `${HEXAGON_SDK_ROOT}/docs/Hap_power_set_dcvs_2.html` in SDK documentation

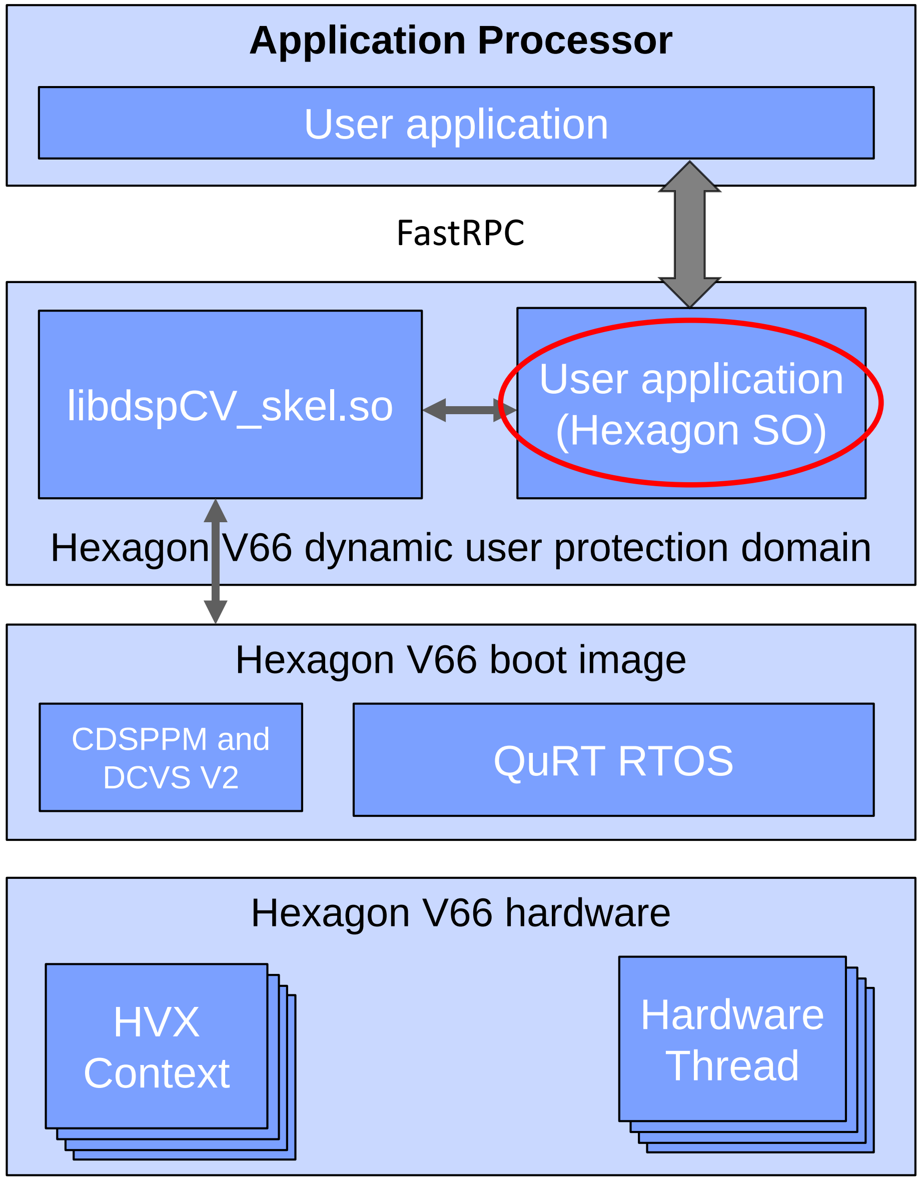
cDSP Software Architecture – User Application

- Partitioned across processors via Hexagon SDK
- User-space application on applications processor
  - Invokes C-callable functions implemented in user application Hexagon SO file, via FastRPC



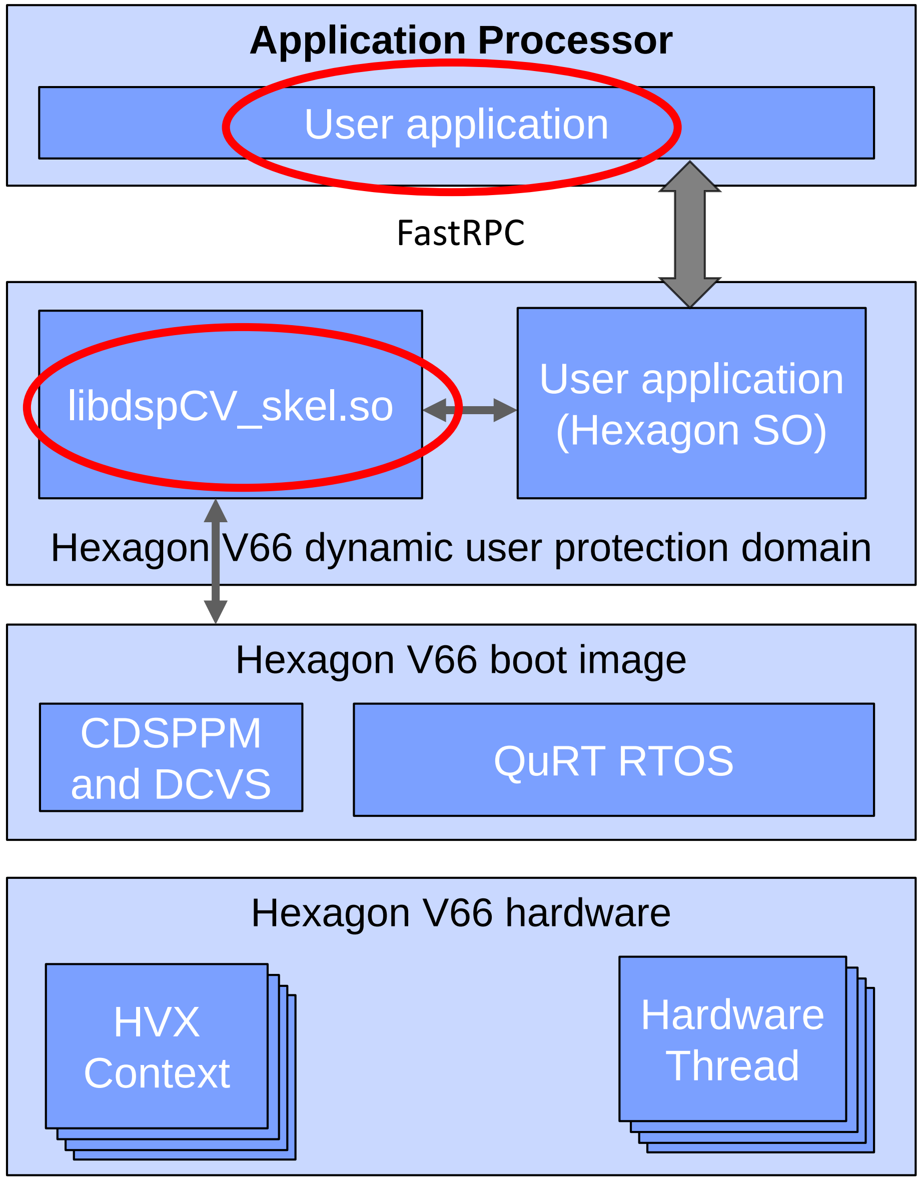
cDSP Software Architecture – User Application (cont.)

- Hexagon SO
  - Hexagon and HVX functions are built with toolchain in Hexagon SDK and placed on device file system
  - Loaded dynamically when invoked by user application on application processor
  - Invokes HAP APIs with DCVS v2 parameters for required clock corner, DDR bandwidth, enable/disable DCVS, power on/off HVX units
  - Supports C and C++ applications



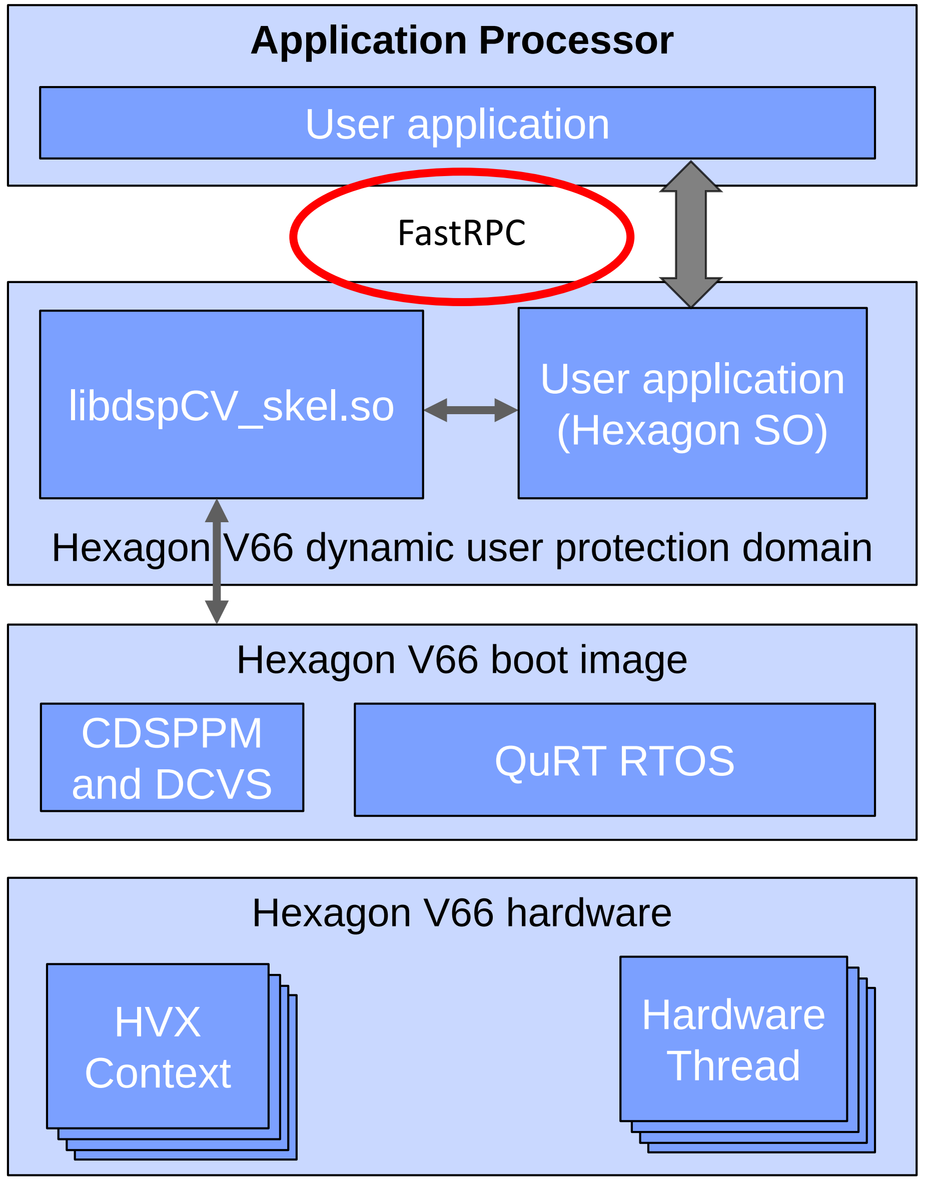
cDSP Software Architecture – libdspCV_skel.so

- `libdspCV_skel.so` library is Hexagon SDK deliverable
- Used by FastCV applications to implement multi-threading
- Wraps required APIs from Hexagon boot image
  - Supplies worker threads and callback interface for user application multi-threading
- One way to add multi-threading capability to algorithm code, customers can implement their own multi-threading mechanism using QuRT API or standard C++ thread APIs



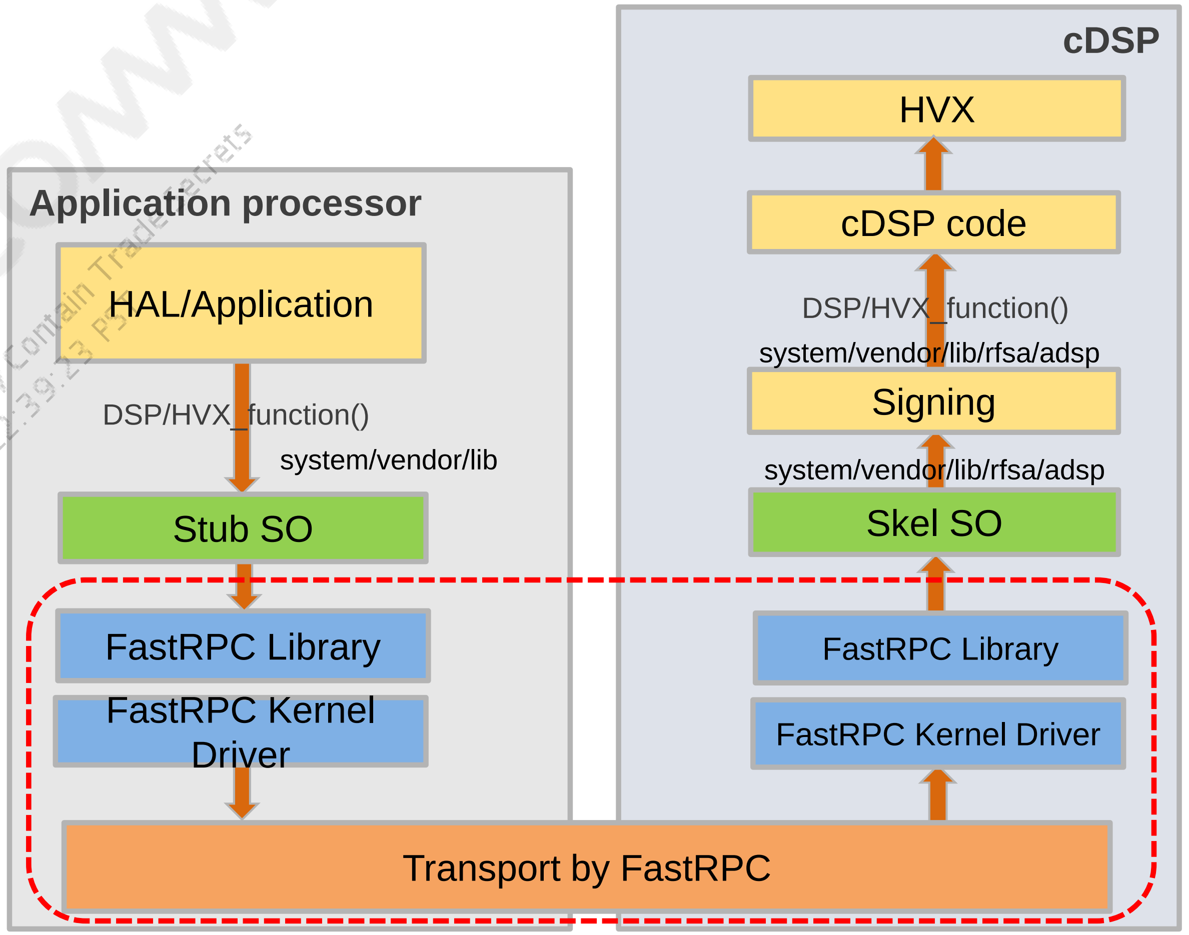
cDSP Software Architecture – FastRPC

- Provides User mode and Kernel mode drivers to enable user application to call functions in Hexagon SO file
- Upon first RPC invocation from user application, FastRPC automatically does the following:
  - Creates user process on cDSP that services this and subsequent calls from caller’s process
  - Marshals function arguments back and forth for each RPC call
  - Tears down cDSP process when application process dies
- Cache IO coherency to reduce RPC latency



Dynamic Loading Data Flow

- Dynamic loading module feature is available
- Used for camera streaming and memory-to-memory processing
- Dynamic loading use is not on cDSP heap
  - Is on ION heap on HLOS side by System Memory Management Unit (SMMU)



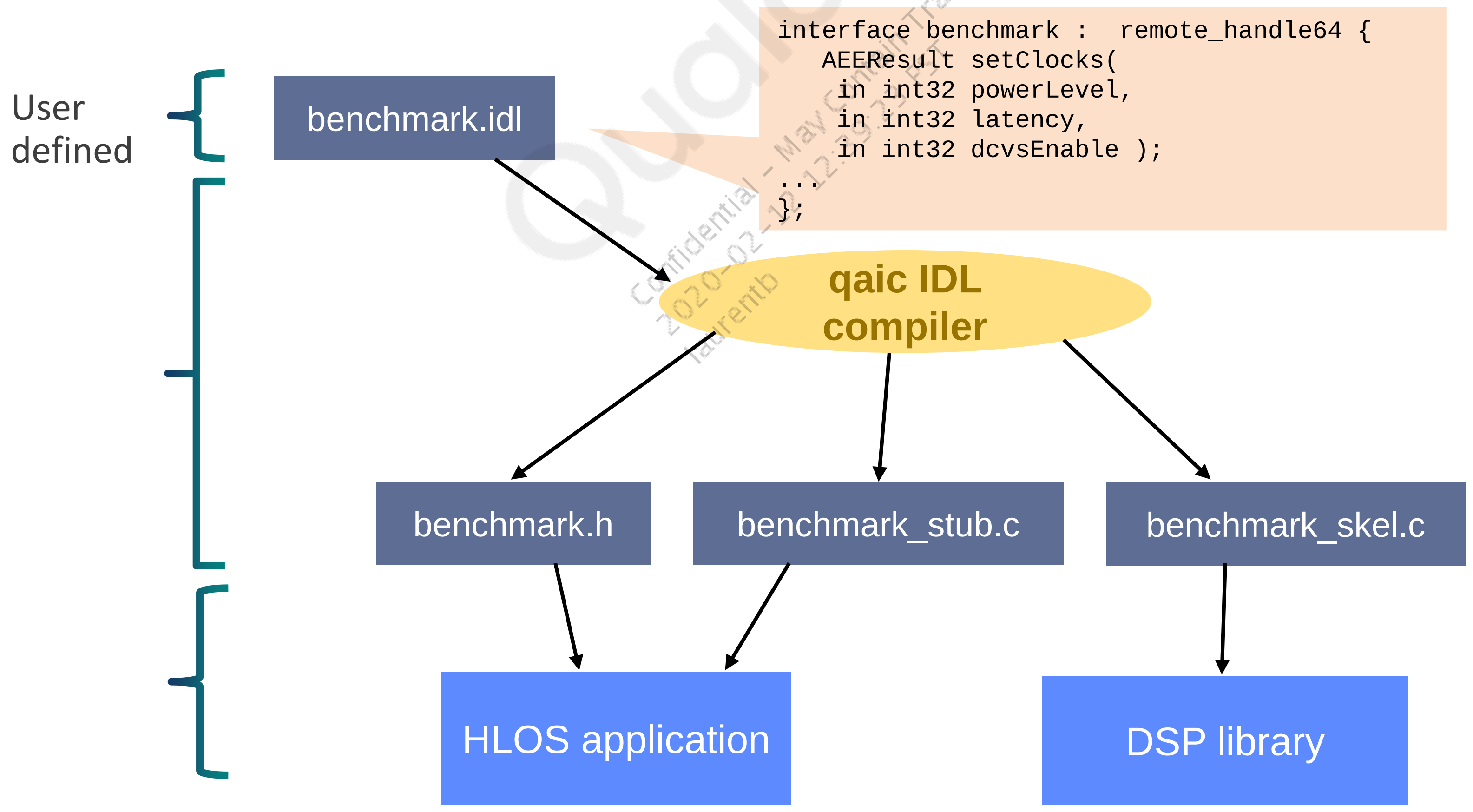
🟩 IDL compiler generated

🟦 RPC software

Note: IDL compiler is distributed with Hexagon SDK.

FastRPC – IDL (Interface Definition Language)

- IDL file defines interfaces that are remoted to CPU and implemented on DSP
- Qaic compiles IDL files into headers, stubs, and skels
- For more information on IDL compiler, see Hexagon SDK documentation: `${HEXAGON_SDK_ROOT}/docs/Tools_IDL Compiler.html`



```
interface benchmark :  remote_handle64 {
   AEEResult setClocks(
    in int32 powerLevel,
    in int32 latency,
    in int32 dcvsEnable );
...
};
```

FastRPC – Cache IO Coherency

- IO coherency is a one-way feature
  - Only cDSP can snoop into CPU cache, not vice versa
  - CPU is not required to flush and invalidate its cache, while cDSP must flush and invalidate its cache
- Without IO coherency
  - CPU and DSP must flush and invalidate their respective caches to maintain cache coherency for each FastRPC invocation
  - Results in additional latency to FastRPC call
- For store operations
  - cDSP writes data to its own cache, invalidates CPU cache lines, and flushes data to main memory
  - When data is accessed, CPU refreshes its cache from main memory
- cDSP snoops into CPU cache for cached buffer loads and stores before accessing main memory

FastRPC – Cache IO Coherency (cont.)

- If a cache miss occurs:
  - cDSP goes to main memory
  - For larger buffers, cDSP goes to CPU cache and then to main memory, which results in increased latency to main memory (DDR) for IO-coherent buffers
    - Caused by snooping
    - Adds to latency on algorithm side
- IO coherency is useful for buffers allocated as cached by CPU
  - Does not provide any benefit for uncached buffers allocated by CPU
- IO coherency is enabled through FastRPC framework and can be enabled/disabled as needed
- CPU scheduling might impact FastRPC latencies due to time taken for CPU wake-up
- Advantages:
  - Reduces time spent on CPU cache clean or invalidate operations
  - Developers are not required to perform cache maintenance operations to ensure coherency

FastRPC – Disable Cache IO Coherency

- Clients can disable IO coherency for ION buffer by registering buffer as non-coherent
  - Use `remote_register_buf_attr()` to register buffer as non-coherent

```
void remote_register_buf_attr(void* buf, int size, int fd, int attrs);
   buf - virtual address of the buffer
   size - size to of the buffer
   fd  - the file descriptor, callers can use -1 to deregister.
   attr - map buffer as coherent or non-coherent

#define FASTRPC_ATTR_NON_COHERENT (2)
```

- Another option is to modify `rpcmem_alloc()` to pass flag to map buffer as non-coherent

```
flags = ION_FLAG_CACHED | RPCMEM_HEAP_NONCOHERENT;
rpcmem_alloc(heapid, flags, size);
```

FastRPC – RPC Latency Control or QoS API

- Votes to disable CPU power collapse when RPC activity is found
- Reduces FastRPC latency
- Usage example with multi-domain feature:

> **Note:** Handle h must be acquired first using `remote_handle64_open`, before invoking `remote_handle64_control`.

```
#include “remote.h”

struct remote_rpc_control_latency data;
data.enable = 1
remote_handle64_control(h, DSPRPC_CONTROL_LATENCY, (void*)&data, sizeof(data));
```

- `remote_handle64_control()` added and exported from libadsprpc.so or libcdsprpc.so library

FastRPC – RPC Latency Control or QoS API (cont.)

- If not using multi-domain feature, libadsprpc.so or libcdsprpc.so linked to the application will determine which domain (adsp/cdsp) QoS is invoked
- Usage example when not using multi-domain feature:

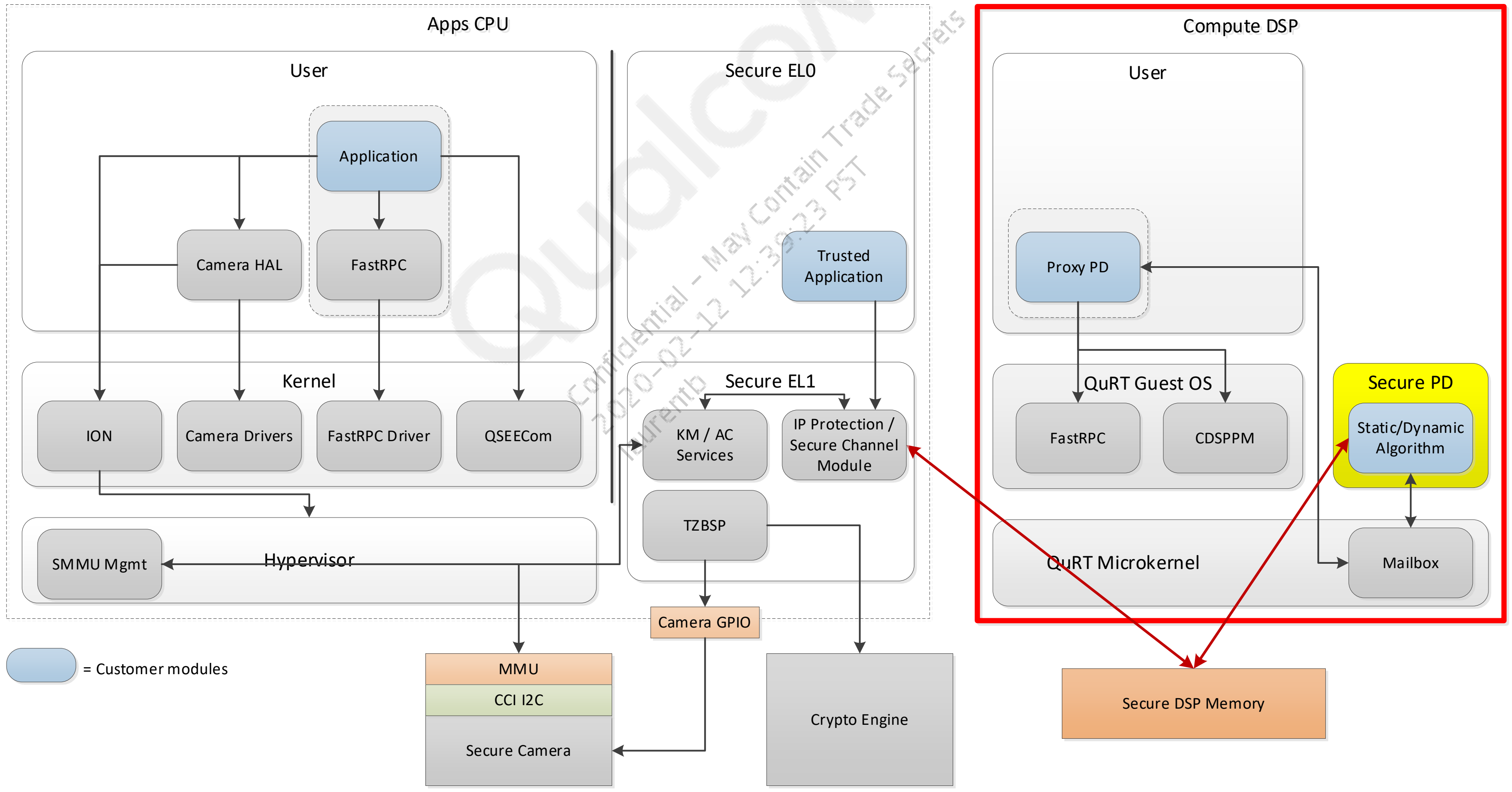
```
#include “remote.h”

Struct remote_rpc_control_latency data;
data.enable = 1
remote_handle_control(DSPRPC_CONTROL_LATENCY, (void*)&data, sizeof(data));
```

Improving FastRPC Latency

- Shorter workloads
  - Stay on CPU or combine multiple calls into one call
- Longer workloads
  - FastRPC performance dominated by CPU wakeup/interrupt latency
  - Disabling CPU power collapse greatly improves performance
- Quality of FastRPC service can be influenced by power vs. latency tradeoffs
  - Power manager QoS votes can disable CPU power collapse
  - Performance lock APIs can set minimum CPU frequencies
  - Profiling/optimizing CPU scheduler
- Using secondary boot image reduces latency added due to log messages

Secure PD

- Static – Algorithm shared object compiled as part of DSP image in SecurePD
- Dynamic – Algorithm shared objects can be loaded dynamically in SecurePD



Note: See *Dynamic Loading in Secure PD* (80-PR150-1) and *Qualcomm® Hexagon™ Secure cDSP Software Overview for SM8150* (80-PK723-1).

HVX Use Cases

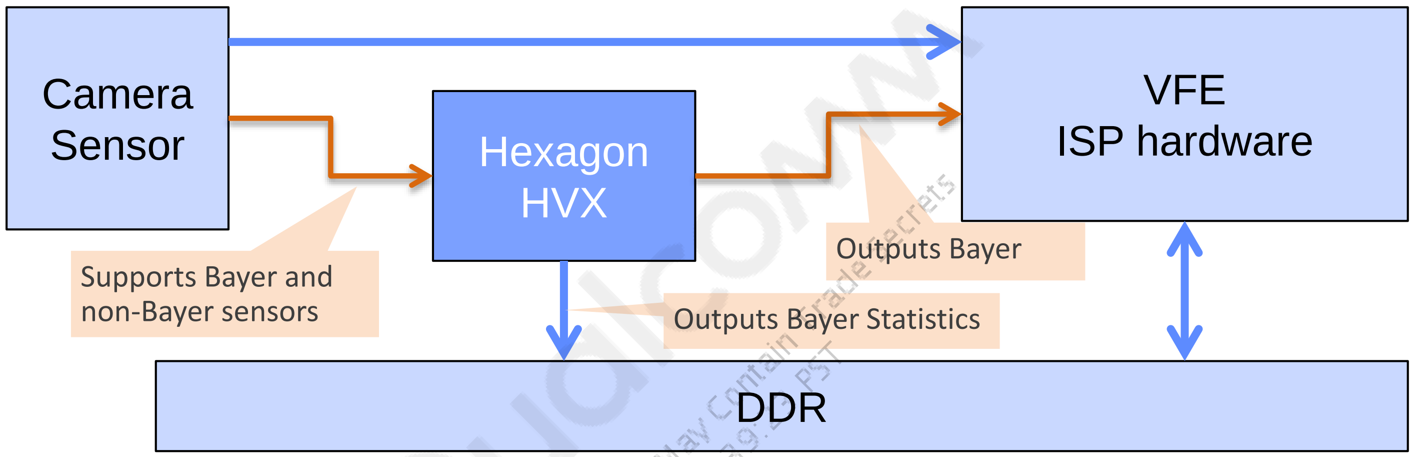
FastCV Overview

- An API and library that enable mobile devices to run Computer Vision (CV) applications efficiently
- Enables acceleration of CV applications through hardware
- Analogous to OpenGL ES in rendering domain; a clean modular library

FastCV value	Details
Smaller scope	APIs are most widely used APIs are most computationally intense APIs are suitable for heterogeneous core optimization
Optimized for embedded or mobile	More granular API Better power and performance

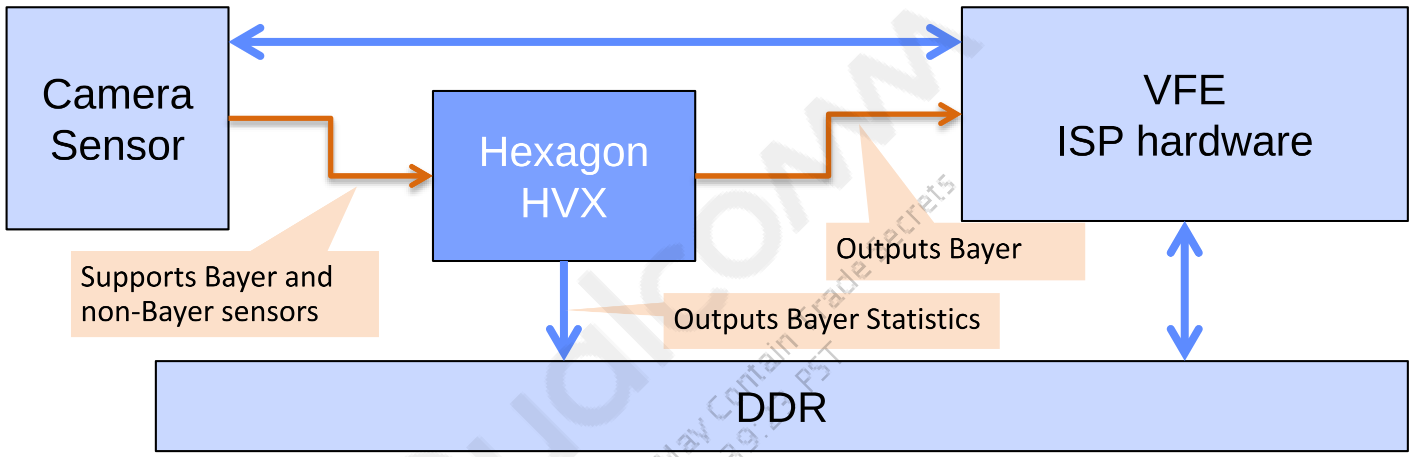
- Detailed list of APIs is available in Hexagon SDK documentation in *Computer Vision* section – *HVX optimized functions*
- For more information on FastCV support, see *Getting Started in the FastCV SDK (https://developer.qualcomm.com/software/fast-cv-sdk/getting-started)*

Camera Streaming Overview



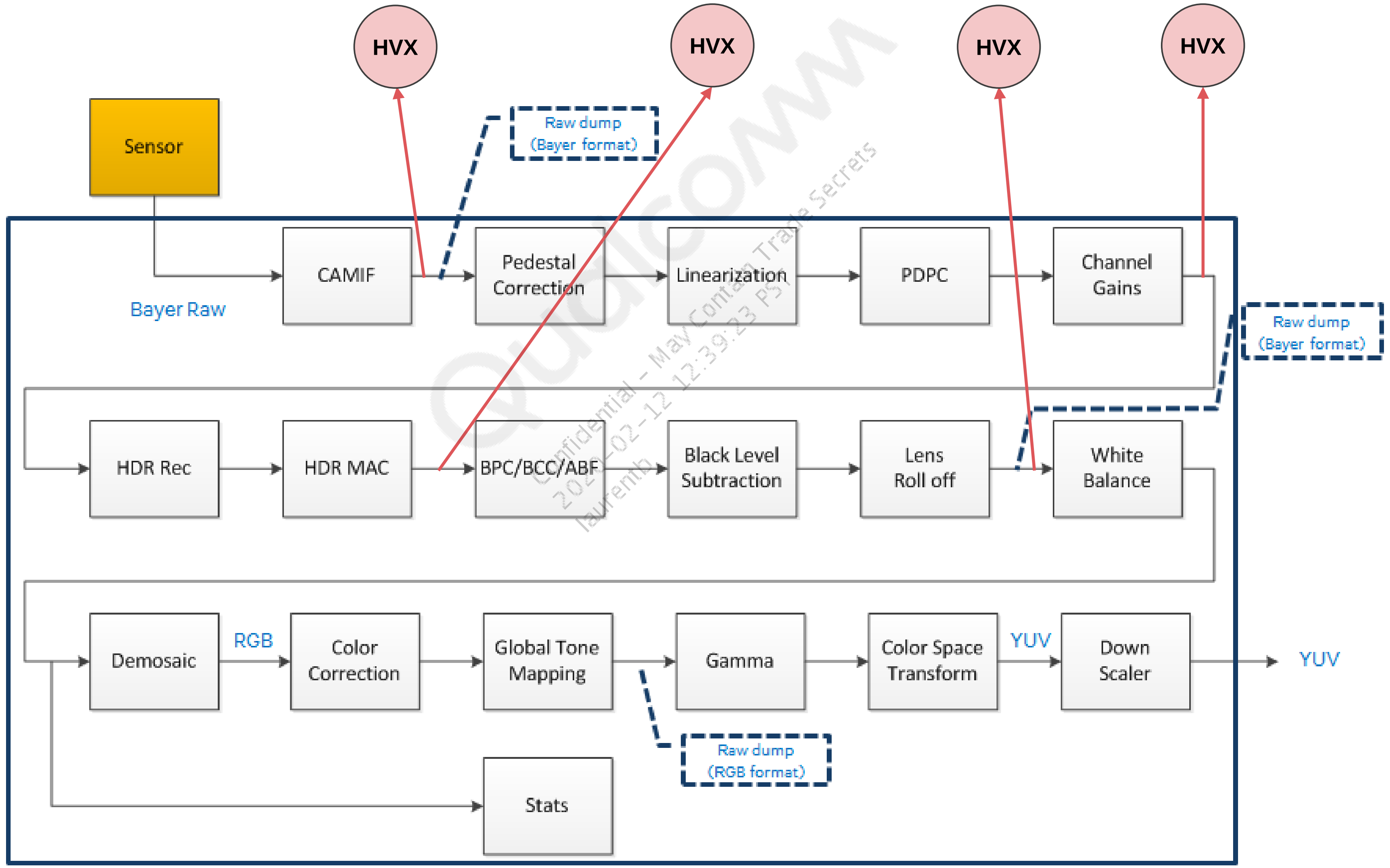
- Special bus enables access
- Use cases:
  - OEM differentiating camera feature
    - Opportunity to reduce BOM by removing external chips
  - Variation in camera sensors (non-Bayer, different resolution) requires programmability
- Direct streaming interface avoids DDR memory accesses
  - Hexagon cDSP can be inserted between camera sensor and VFE (QTI ISP)

Camera Streaming Overview (cont.)

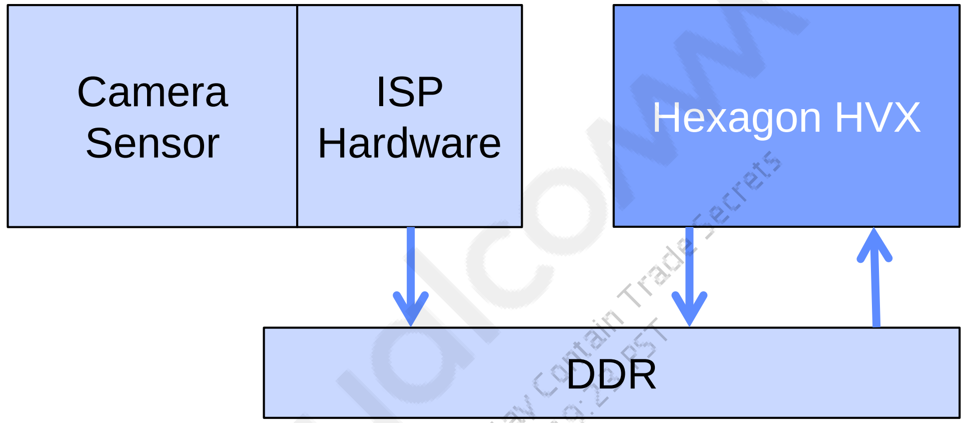


- Example algorithm/use case:
  - Custom Bayer statistics for auto-focus/auto-exposure
    - Bayer focus (region line max, sharpness, sum, num)
    - Bayer exposure statistics (num, sum)
- For details, see `${HEXAGON_SDK_ROOT}/docs/Camera streaming`

Camera Streaming – HVX Tap Points



Camera Memory-to-Memory Pre/Postprocessing



- Use case:
  - Memory-to-memory before ISP or after ISP
- Example application:
  - Wavelet-based denoiser
  - CV
- For more details, see `${HEXAGON_SDK_ROOT}/examples/common/benchmark_v65`

HVX Customization Workflow

Stages of HVX Customization

Evaluate	Define/Design	Integrate	Testing/Optimization	Manufacture
		Integration	**Power and Performance Optimization**	**Stability and Signing**
Customer downloads latest Hexagon SDK and LLVM toolchain (HVX examples are provided) for evaluation of use case	Customer identifies feature API to implement on HVX; starts developing code using SDK while referring to SDK example code	Customer integrates and modifies HAL code to invoke HVX-optimized functionality	Customer optimizes power and performance	Customer resolves stability issues and signs module
Latest Hexagon SDK, toolchain, and examples are available on [CreatePoint]or at [developer.qualcomm.com]	Customer develops standalone test application that exercises custom HVX-optimized functionality; verifies application on a target device. Before verification, target device must be signed using *testsig*. Customer includes performance profiling in this process.	Customer performs additional end-to-end testing of feature using real-life use cases and profiles for power and performance requirements.	Customer can use *sysmon* tool to profile performance.	Customer can use *adspcrashman* to resolve stability issues (see stability and debugging documentation). Static hashing should be used for mass production signing of module.

cDSP – Development Environment

Development Tools

- Hexagon SDK
  - Available for Linux and Windows development platforms
  - Eclipse IDE for build, debugging, and profiling examples on simulation and target
  - Command-line build environment using makefiles
  - IDL compiler – to remote APIs
  - Documentation – in-depth technical articles on all things cDSP and pointers to PDF documents
  - Examples – Designed for compute, neural network uses cases that can be compiled for simulator or target. Implemented in C, HVX Compiler intrinsics, and assembly
  - Tools
    - LLVM Compiler Toolchain – Provides example kernels that can be modified, built, profiled, and debugged on PC
    - Simulator – For profiling DSP code performance using DSP OS – QuRT simulation
    - Debugger - Target and simulator
    - Profiling – Target: Hexagon Trace Analyzer, Sysmon. Simulator: Hexagon Profiler
    - Signing – Scripts for generating development test signatures and signing production shared objects. Shared objects are authenticated prior to loading on cDSP
  - Library to help port Floating-point arithmetic.
- Halide compiler
  - Released as part of SDK and separately on CreatePoint. Latest releases with bug fixes or enhancements can become available earlier on CreatePoint than as part of SDK
  - Provides build environment to compile examples implemented in Halide

Note: Use latest CreatePoint tool because Hexagon development tools version might be updated during chipset lifecycle.

Hexagon SDK – Getting Started

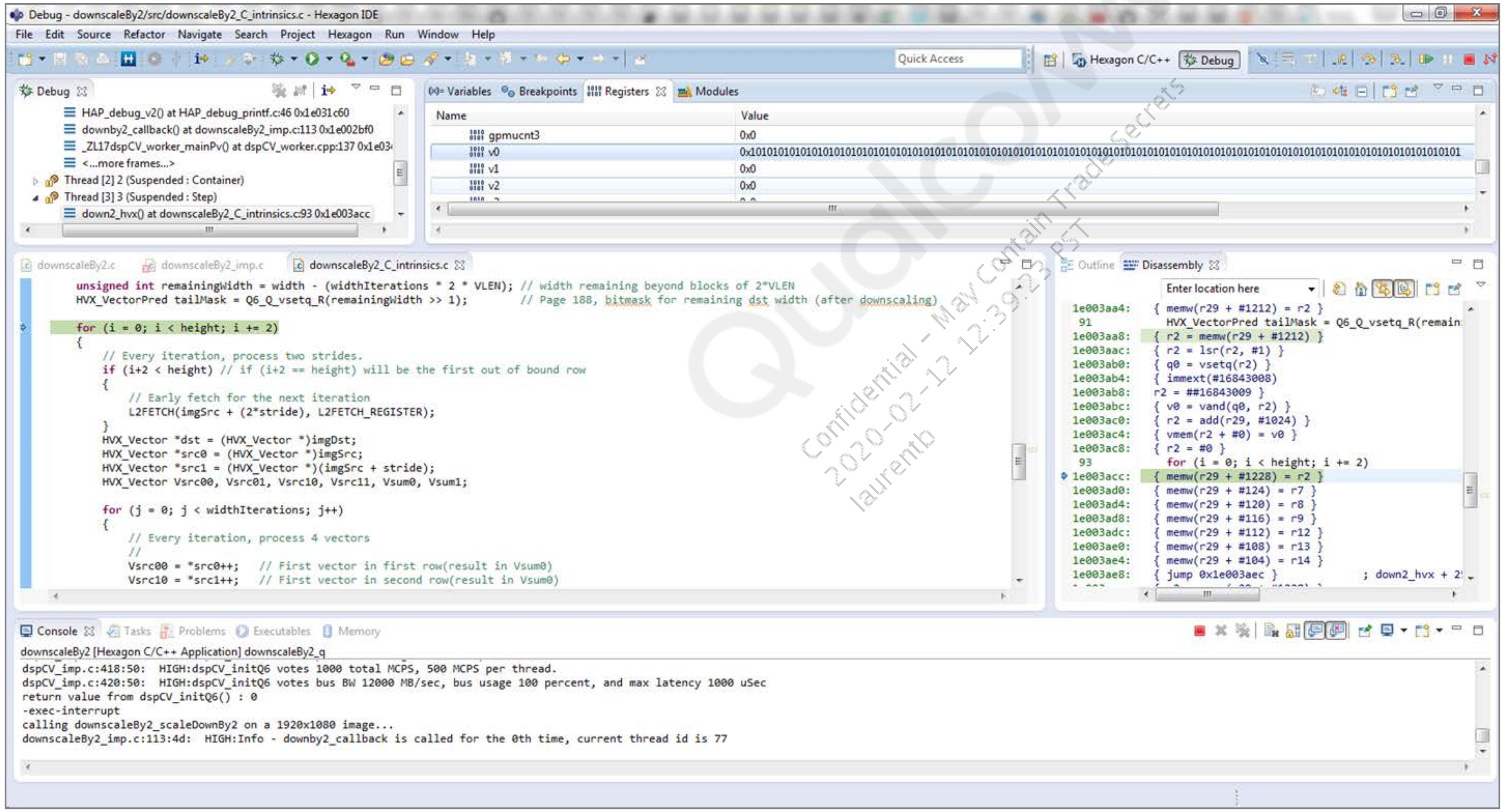
- Use SDK to build code for PC simulation and standalone execution on target
- SDK target verification involves dynamically linked shared objects loaded on cDSP and Arm processor
- After verifying on target, integrate Arm functionality in HLOS
- Documentation: `${HEXAGON_SDK_ROOT}/docs`
- Example code: `${HEXAGON_SDK_ROOT}/examples`
  - `${HEXAGON_SDK_ROOT}/examples/common/benchmark_v65` demonstrates RTOS, HVX, and VTCM use
  - Domain-specific examples available in their respective folders
  - To ensure FastRPC and development signing work properly, run calculator example provided in `${HEXAGON_SDK_ROOT}/examples/common/calculator`

Hexagon SDK – Getting Started (cont.)

- For Hexagon v65 and later, start with benchmark_v65 example, which demonstrates multiple computational kernels for HVX and best practices: `${HEXAGON_SDK_ROOT}/examples/common/benchmark_v65`
  - Test executable and stub.so (Arm implementation): `${HEXAGON_SDK_ROOT}/examples/common/benchmark_v65/android_Release/ship`
    Skel.so (cDSP implementation) `${HEXAGON_SDK_ROOT}/examples/common/benchmark_v65/hexagon_Release_dynamic_toolv83_v65/ship`
  - IDL: `${HEXAGON_SDK_ROOT}/examples/common/benchmark_v65/inc`
  - Stub-generated code: `${HEXAGON_SDK_ROOT}/examples/common/benchmark_v65/hexagon_Release_dynamic_toolv83_v65`
  - Test src (Arm): `${HEXAGON_SDK_ROOT}/examples/common/benchmark_v65/src_app`
  - DSP assembly src: `${HEXAGON_SDK_ROOT}/examples/common/benchmark_v65/asm_src`
  - DSP C intrinsics src: `${HEXAGON_SDK_ROOT}/examples/common/benchmark_v65/src_dsp`
- For descriptions of command line parameters and usage instructions, see `${HEXAGON_SDK_ROOT}/examples/common/benchmark_v65/hexagon_Release_dynamic_toolv83_v65 /readme.txt`
- To build and load it to target, run `${HEXAGON_SDK_ROOT}/examples/common/benchmark_v65/hexagon_Release_dynamic_toolv83_v65 /benchmark_v65_walkthrough.py`

Hexagon SDK – Eclipse IDE

- Bundled within Hexagon SDK to assist with development of HVX standalone applications



```
HAP_debug_v2() at HAP_debug_printf.c:46 0x1e031c60
downby2_callback() at downscaleBy2_imp.c:113 0x1e002bf0
_ZL17dspCV_worker_mainPv() at dspCV_worker.cpp:137 0x1e03
<...more frames...>
Thread [2] 2 (Suspended : Container)
Thread [3] 3 (Suspended : Step)
    down2_hvx() at downscaleBy2_C_intrinsics.c:93 0x1e003acc
```

Name	Value
gpmucnt3	0x0
v0	0x1010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101
v1	0x0
v2	0x0

downscaleBy2.c | downscaleBy2_imp.c | downscaleBy2_C_intrinsics.c

```c
    unsigned int remainingWidth = width - (widthIterations * 2 * VLEN); // width remaining beyond blocks of 2*VLEN
    HVX_VectorPred tailMask = Q6_Q_vsetq_R(remainingWidth >> 1);        // Page 188, bitmask for remaining dst width (after downscaling)

    for (i = 0; i < height; i += 2)
    {
        // Every iteration, process two strides.
        if (i+2 < height) // if (i+2 == height) will be the first out of bound row
        {
            // Early fetch for the next iteration
            L2FETCH(imgSrc + (2*stride), L2FETCH_REGISTER);
        }
        HVX_Vector *dst = (HVX_Vector *)imgDst;
        HVX_Vector *src0 = (HVX_Vector *)imgSrc;
        HVX_Vector *src1 = (HVX_Vector *)(imgSrc + stride);
        HVX_Vector Vsrc00, Vsrc01, Vsrc10, Vsrc11, Vsum0, Vsum1;

        for (j = 0; j < widthIterations; j++)
        {
            // Every iteration, process 4 vectors
            //
            Vsrc00 = *src0++;   // First vector in first row(result in Vsum0)
            Vsrc10 = *src1++;   // First vector in second row(result in Vsum0)
```

```
1e003aa4:   { memw(r29 + #1212) = r2 }
 91           HVX_VectorPred tailMask = Q6_Q_vsetq_R(remain
1e003aa8:   { r2 = memw(r29 + #1212) }
1e003aac:   { r2 = lsr(r2, #1) }
1e003ab0:   { q0 = vsetq(r2) }
1e003ab4:   { immext(#16843008)
1e003ab8:   r2 = ##16843009 }
1e003abc:   { v0 = vand(q0, r2) }
1e003ac0:   { r2 = add(r29, #1024) }
1e003ac4:   { vmem(r2 + #0) = v0 }
1e003ac8:   { r2 = #0 }
 93           for (i = 0; i < height; i += 2)
1e003acc:   { memw(r29 + #1228) = r2 }
1e003ad0:   { memw(r29 + #124) = r7 }
1e003ad4:   { memw(r29 + #120) = r8 }
1e003ad8:   { memw(r29 + #116) = r9 }
1e003adc:   { memw(r29 + #112) = r12 }
1e003ae0:   { memw(r29 + #108) = r13 }
1e003ae4:   { memw(r29 + #104) = r14 }
1e003ae8:   { jump 0x1e003aec }           ; down2_hvx + 2
```

downscaleBy2 [Hexagon C/C++ Application] downscaleBy2_q

```
dspCV_imp.c:418:50:  HIGH:dspCV_initQ6 votes 1000 total MCPS, 500 MCPS per thread.
dspCV_imp.c:420:50:  HIGH:dspCV_initQ6 votes bus BW 12000 MB/sec, bus usage 100 percent, and max latency 1000 uSec
return value from dspCV_initQ6() : 0
-exec-interrupt
calling downscaleBy2_scaleDownBy2 on a 1920x1080 image...
downscaleBy2_imp.c:113:4d:  HIGH:Info - downby2_callback is called for the 0th time, current thread id is 77
```

For details see: `${HEXAGON_SDK_ROOT}/docs/eclipse_first_project.html`
`${HEXAGON_SDK_ROOT}/docs/eclipse_projects.html`

Profiling

There are three types of profiling:

- Programmatic
- Simulator
- Target

Profiling (cont.)

Programmatic

- Profiling functions
  - HAP_perf_get_time_us() – Gets the current microseconds
  - HAP_perf_get_pcycles() – Gets the current 64-bit processor cycle count
  - Exported in `${HEXAGON_SDK_ROOT}/incs/HAP_perf.h`
- Function can be added before and after a function or piece of code to be profiled in developed Hexagon code
- For use see:
  `${HEXAGON_SDK_ROOT}/examples/common/benchmark_v65/src_dsp/gaussian7x7_imp.c`

Profiling (cont.)

Simulator

- Profiling flags –`profile`, `--timing`, `--packet_analyze`, to generate profiling data and timing analysis data files, can be added to QEXE_EXEC_SIM_OPTIONS in SDK example makefile, hexagon.min
  - For information:
    - `${HEXAGON_SDK_ROOT}/ docs/Platforms_Simulator.html` see *Advanced Profiling on Simulator* section
    - *Qualcomm® Hexagon™ gprof Profiler User Guide* (80-N2040-29)
    - *Qualcomm® Hexagon™ Profiler User Guide* (80-N2040-10)
    - These documents are released as part of Hexagon SDK and located at: `${HEXAGON_SDK_ROOT}/tools/HEXAGON_Tools/<latest toolchain>/Documents`

Profiling (cont.)

Target

- **sysMon** – Provides DSP load and statistics, such as pCPP, MPPS, L2 Cache, and AXI bus, for system-level profiling. Provided as Android app and command line executable in SDK or cDSP build
  - **sysMon_DSP_Profiler_V2.apk** is Android UI application for profiling aDSP/cDSP/sDSP work loads untethered. Provides limited profiling options compared to sysMonApp

    `${HEXAGON_SDK_ROOT}/docs/sysMon_DSP_Profiler_V2.html` provides more information
  - **sysMonApp** – Android executable, which can be used from the cmd prompt using adb shell to profile DSP subsystem. Device must be connected over USB to PC. Tool provides more profiling options, such as setting clock or locking cache

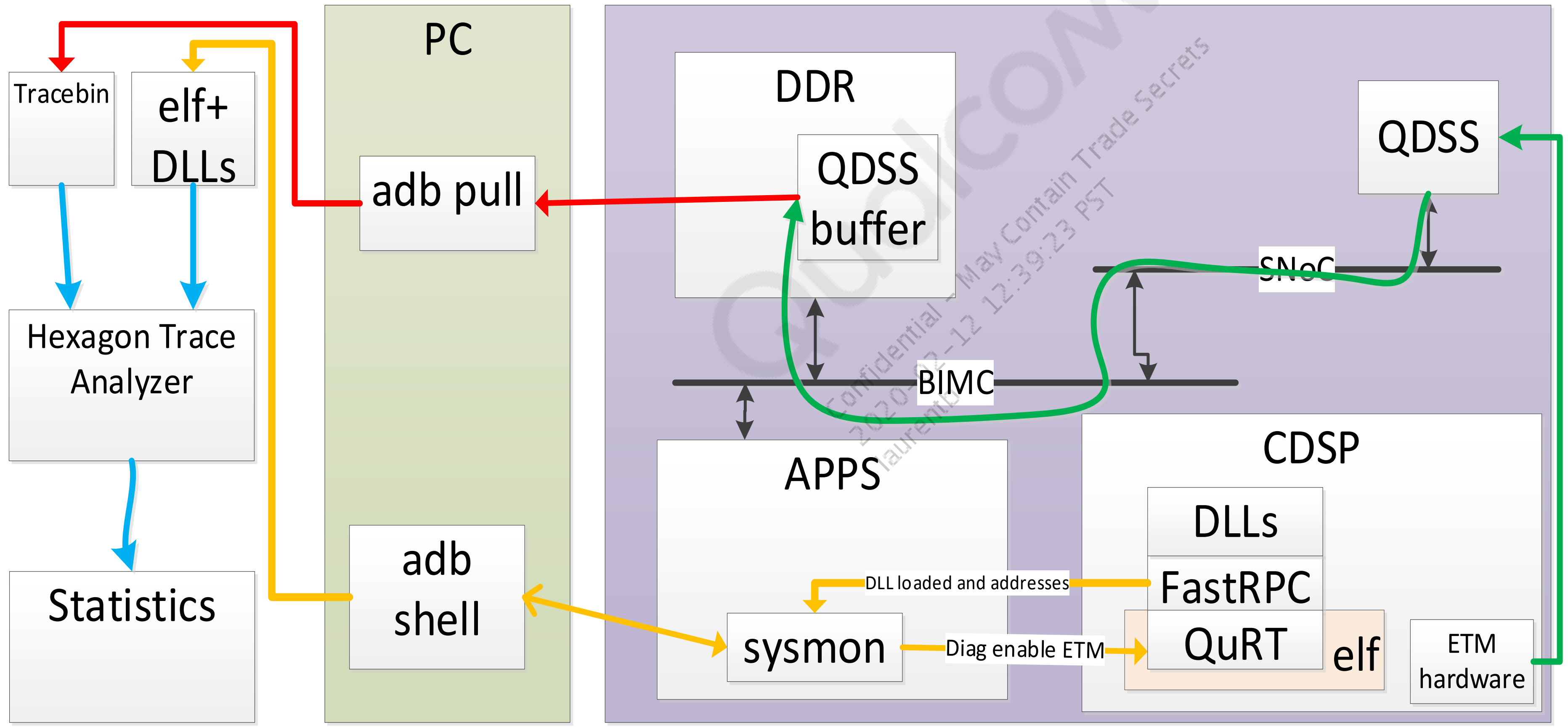
    For more information, see `${HEXAGON_SDK_ROOT}/docs/sysMonApp.html`
- **Hexagon trace analyzer**
  - Software trace analysis tool that provides greater granularity at function and packet level
  - Processes Hexagon ETM (Embedded Trace Macrocell) traces to derive flow of each processor thread
  - Valuable tool for giving insights into code execution, allowing in-depth analysis and optimization

    For more information, see `${HEXAGON_SDK_ROOT}/docs/hexagon_trace_analyzer_doc.html`

Hexagon Trace Analyzer

- Hexagon Trace Analyzer
  - Uses output of Q6 ETM hardware to trace every instruction and number of cycles
  - Near zero impact on runtime performance
  - Use in Mission mode, no instrumentation required
  - Supported for cDSP
  - Available with Linux SDK
  - Works with development devices (debug fuse not blown) only
- Outputs
  - Flat image analysis
    - Per-instruction/function/directory/task statistics
    - Per-task call trees -> Flamegraphs
  - Temporal image analysis
    - Catapult
      - Timeline with per-task function call trees

Q6 ETM Support in Qualcomm Modems



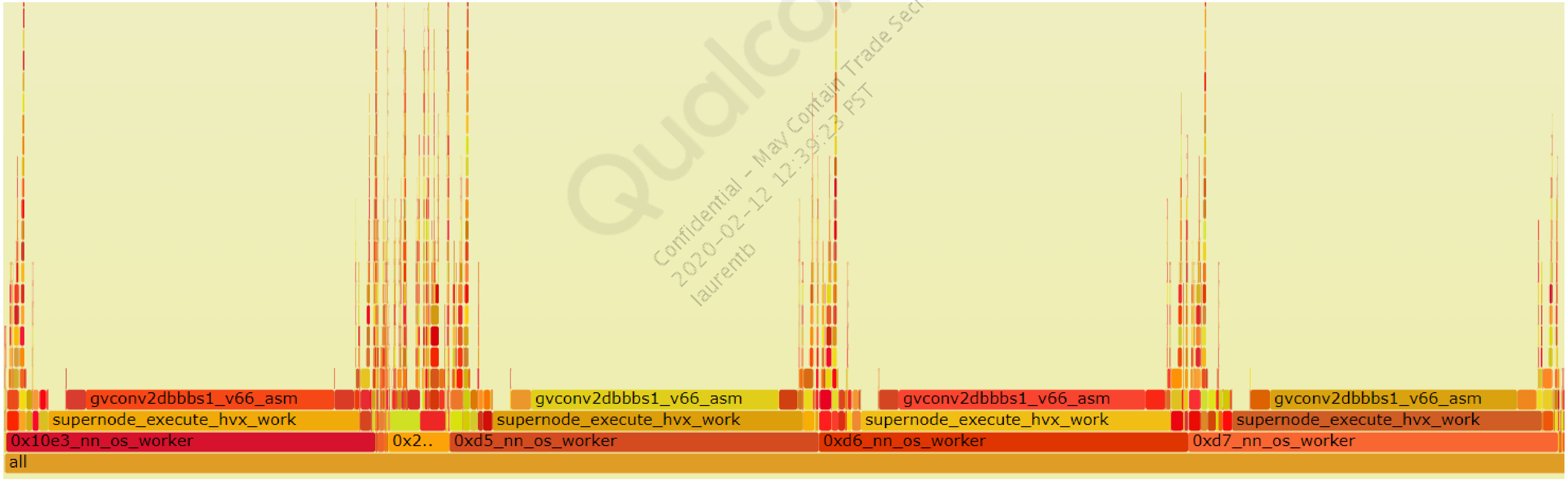
Prerequisites

- Hexagon Trace Analyzer executable
- Linux machine
- Python 2.7
- sysMonApp from SDK release, to enable ETM tracing and get DLL load addresses, required for parsing data
- adb drivers
- Docker

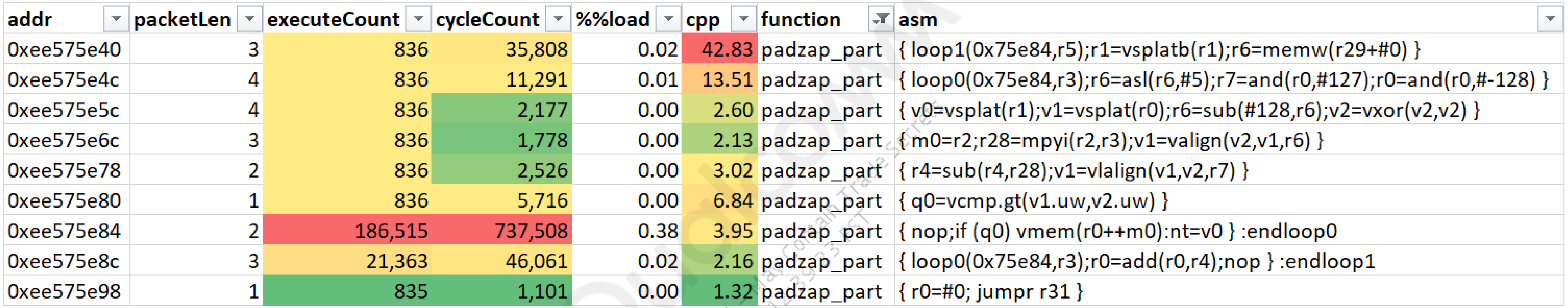
  Running Hexagon Trace Analyzer does not strictly need Docker, but Docker allows you to set up consistent virtualized Linux environment. Example Docker file and configuration, plus other required tools (FlameGraph and Catapult), are included
- FlameGraph (https://github.com/brendangregg/FlameGraph) and Catapult (https://github.com/catapult-project/catapult) are required for some visualization. To get these without Docker, install tools under directory containing hexagon-trace-analyzer

Flamegraphs

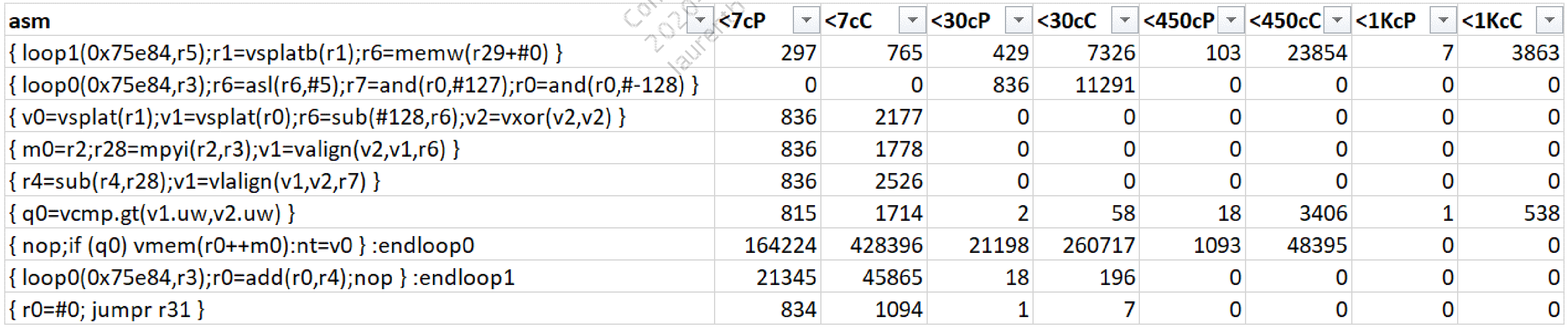
- For each task, shows the call tree
- Width is proportional to cycles consumed by that function



perPacketStats

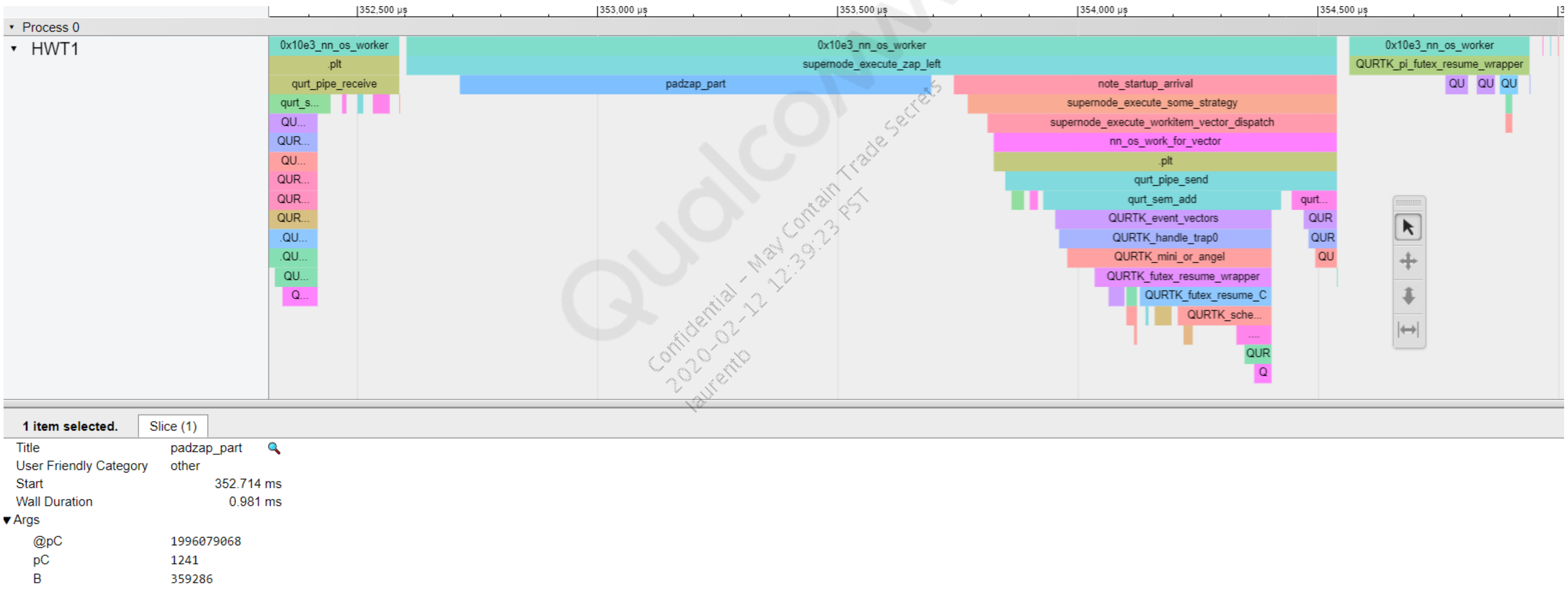


addr	packetLen	executeCount	cycleCount	%%load	cpp	function	asm
0xee575e40	3	836	35,808	0.02	42.83	padzap_part	{ loop1(0x75e84,r5);r1=vsplatb(r1);r6=memw(r29+#0) }
0xee575e4c	4	836	11,291	0.01	13.51	padzap_part	{ loop0(0x75e84,r3);r6=asl(r6,#5);r7=and(r0,#127);r0=and(r0,#-128) }
0xee575e5c	4	836	2,177	0.00	2.60	padzap_part	{ v0=vsplat(r1);v1=vsplat(r0);r6=sub(#128,r6);v2=vxor(v2,v2) }
0xee575e6c	3	836	1,778	0.00	2.13	padzap_part	{ m0=r2;r28=mpyi(r2,r3);v1=valign(v2,v1,r6) }
0xee575e78	2	836	2,526	0.00	3.02	padzap_part	{ r4=sub(r4,r28);v1=vlalign(v1,v2,r7) }
0xee575e80	1	836	5,716	0.00	6.84	padzap_part	{ q0=vcmp.gt(v1.uw,v2.uw) }
0xee575e84	2	186,515	737,508	0.38	3.95	padzap_part	{ nop;if (q0) vmem(r0++m0):nt=v0 } :endloop0
0xee575e8c	3	21,363	46,061	0.02	2.16	padzap_part	{ loop0(0x75e84,r3);r0=add(r0,r4);nop } :endloop1
0xee575e98	1	835	1,101	0.00	1.32	padzap_part	{ r0=#0; jumpr r31 }



asm	<7cP	<7cC	<30cP	<30cC	<450cP	<450cC	<1KcP	<1KcC
{ loop1(0x75e84,r5);r1=vsplatb(r1);r6=memw(r29+#0) }	297	765	429	7326	103	23854	7	3863
{ loop0(0x75e84,r3);r6=asl(r6,#5);r7=and(r0,#127);r0=and(r0,#-128) }	0	0	836	11291	0	0	0	0
{ v0=vsplat(r1);v1=vsplat(r0);r6=sub(#128,r6);v2=vxor(v2,v2) }	836	2177	0	0	0	0	0	0
{ m0=r2;r28=mpyi(r2,r3);v1=valign(v2,v1,r6) }	836	1778	0	0	0	0	0	0
{ r4=sub(r4,r28);v1=vlalign(v1,v2,r7) }	836	2526	0	0	0	0	0	0
{ q0=vcmp.gt(v1.uw,v2.uw) }	815	1714	2	58	18	3406	1	538
{ nop;if (q0) vmem(r0++m0):nt=v0 } :endloop0	164224	428396	21198	260717	1093	48395	0	0
{ loop0(0x75e84,r3);r0=add(r0,r4);nop } :endloop1	21345	45865	18	196	0	0	0	0
{ r0=#0; jumpr r31 }	834	1094	1	7	0	0	0	0

Catapult – Call Tree of Task on Hardware Thread



1 item selected. Slice (1)

Title	padzap_part
User Friendly Category	other
Start	352.714 ms
Wall Duration	0.981 ms

▼Args

@pC	1996079068
pC	1241
B	359286

Debugging Tools and Methods

- Debug logs
- Remote debugger
- User process crash log
- Debugging on Hexagon simulator using Eclipse IDE or command line
- Debugging on target device, crash and stability Issues
- Utility APIs - qprintf

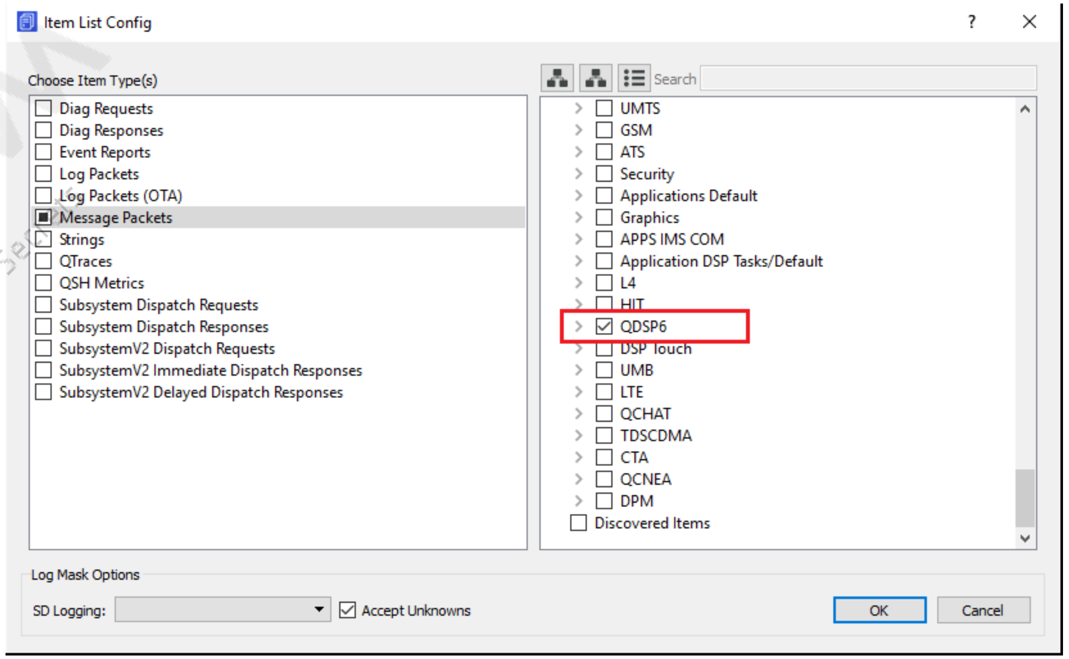
Debugging – Debug Logs

- Debug Logs (on target)
  - `FARF()` or `printf()` – Diagnostic messages on DSP can be sent using FARF or printf
  - Runtime FARF – Enable messages typically disabled in compilation; less overhead
  - Logcat – Regular FARF messages on DSP can be directed to logcat

    When enabled, FARF messages on DSP show up in logcat stream with tag **adsprpc**
  - The following diagnostic tools can be used to view and collect DSP messages:
    - QXDM Professional™ Tool (QXDM Professional) – For debug messages; available on CreatePoint
    - Mini-DM – Available as part of Hexagon SDK installation, `${HEXAGON_SDK_ROOT}/tools/mini-dm`
  - For details, see `${HEXAGON_SDK_ROOT}/docs/Debugging_Message Logs.html`

Debugging – Debug Logs (cont.)

- QXDM Professional
  - F3 - Messages View shows DSP log message
  - In QXDM Professional, enable DSP log mask:
    1. Press **F3**.
    2. Right-click in the **Messages View** window.
    3. Select **Configure** > **Message Packets**.
    4. Select **QDSP6**.



Item List Config

Choose Item Type(s)

- [ ] Diag Requests
- [ ] Diag Responses
- [ ] Event Reports
- [ ] Log Packets
- [ ] Log Packets (OTA)
- [■] Message Packets
- [ ] Strings
- [ ] QTraces
- [ ] QSH Metrics
- [ ] Subsystem Dispatch Requests
- [ ] Subsystem Dispatch Responses
- [ ] SubsystemV2 Dispatch Requests
- [ ] SubsystemV2 Immediate Dispatch Responses
- [ ] SubsystemV2 Delayed Dispatch Responses

Search

- [ ] UMTS
- [ ] GSM
- [ ] ATS
- [ ] Security
- [ ] Applications Default
- [ ] Graphics
- [ ] APPS IMS COM
- [ ] Application DSP Tasks/Default
- [ ] L4
- [ ] HIT
- [✓] QDSP6
- [ ] DSP Touch
- [ ] UMB
- [ ] LTE
- [ ] QCHAT
- [ ] TDSCDMA
- [ ] CTA
- [ ] QCNEA
- [ ] DPM
- [ ] Discovered Items

Log Mask Options

SD Logging: [✓] Accept Unknowns

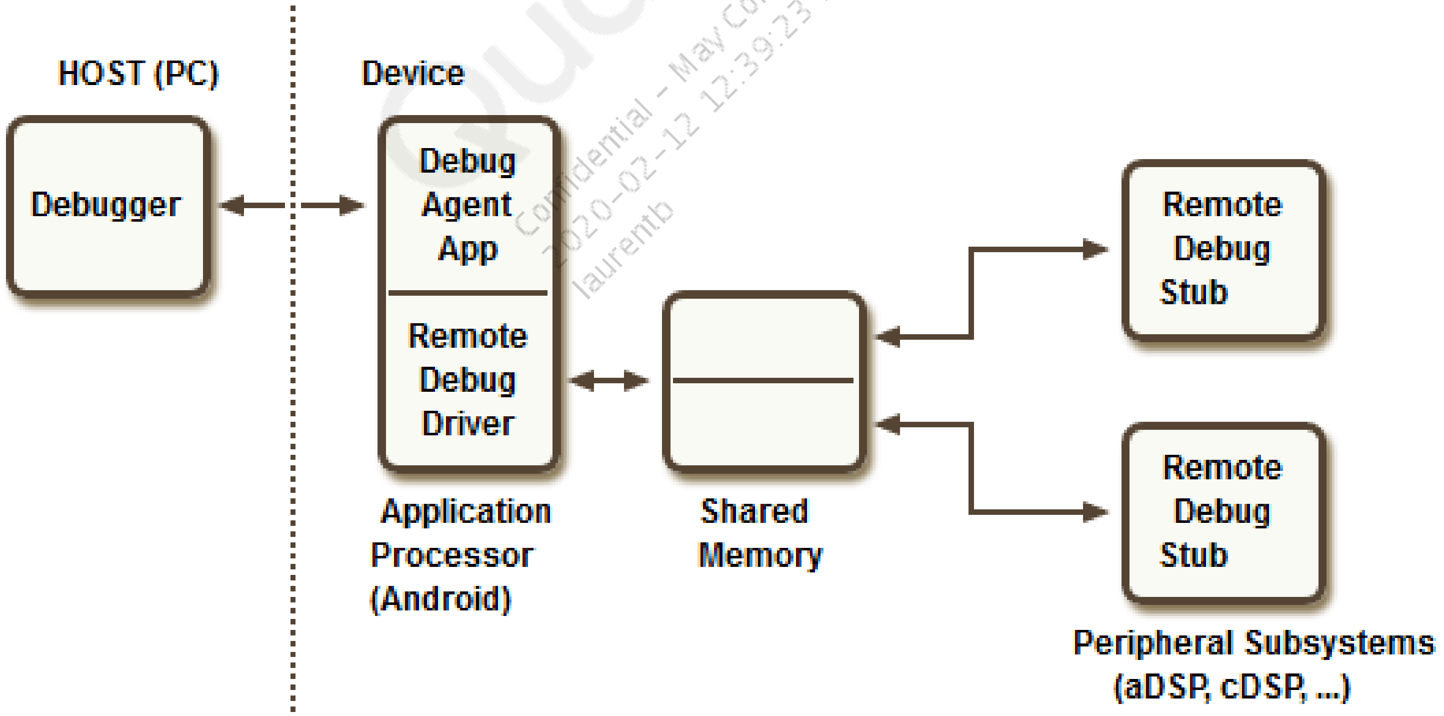
OK Cancel

- Min-DM
  - Min-DM shows same log as QXDM Professional
  - Recommended if unfamiliar with QXDM Professional
  - In device manager or QPST, check comport number of USB diagnostic port
  - Close QXDM Professional and QPST, because Min-DM shares DIAG port

```
>mini-dm.exe --comport com120
<dm_serial_init> Opening port com120
<dm_serial_init> Port com120 opened!
mini-dm is waiting for a DMSS connection...
DMSS is connected. Running mini-dm...
```

Debugging – Remote Debugger

- Remote debugger
  - Provides reliable and accurate debug capability for debugging subsystem processor image without using hardware debugger
  - Uses LLDB running on host PC to communicate with remote stub running on cDSP
  - For setup and other details, see `${HEXAGON_SDK_ROOT}/docs/Debugging_Target.html`



Debugging – Remote Debugger (cont.)

- Host PC
  - Debugger: Debugger application (LLDB) running on host PC that communicates with remote stub
- Device
  - Debug agent: Software that runs on Android platform, providing connectivity from device to host PC
  - Remote debug driver: Character-based driver that debug agent uses to transport payload received from host to debug stub running on subsystem processor over shared memory and vice versa
  - Shared memory: Shared memory from SMEM pool accessible from application processor and subsystem processors
  - Remote debug stub: Privileged code that runs in subsystem processor kernels that receive debug commands from debugger running on host and acts on commands. Commands include reading and writing to registers and memory belonging to subsystem address space, setting breakpoints, single stepping, etc.

Debugging – Remote Debugger (cont.)

Overall flow

1. When debug application starts, shared memory-based transport channel is opened to being debugged (for example, aDSP or cDSP).
2. Debug agent application communicates with DSP to discover which processes are running and expose a port for each.
3. LLDB on host machine connects to port associated with DSP process that user intends to debug.
4. Using this port, LLBD debugs process on DSP, including setting breakpoints, reading registers, and querying threads.
5. When process reaches breakpoint, it is halted.

   Control is turned over to LLDB, enabling user to perform functions such as single stepping or to continue.

Debugging – User Process Crash log

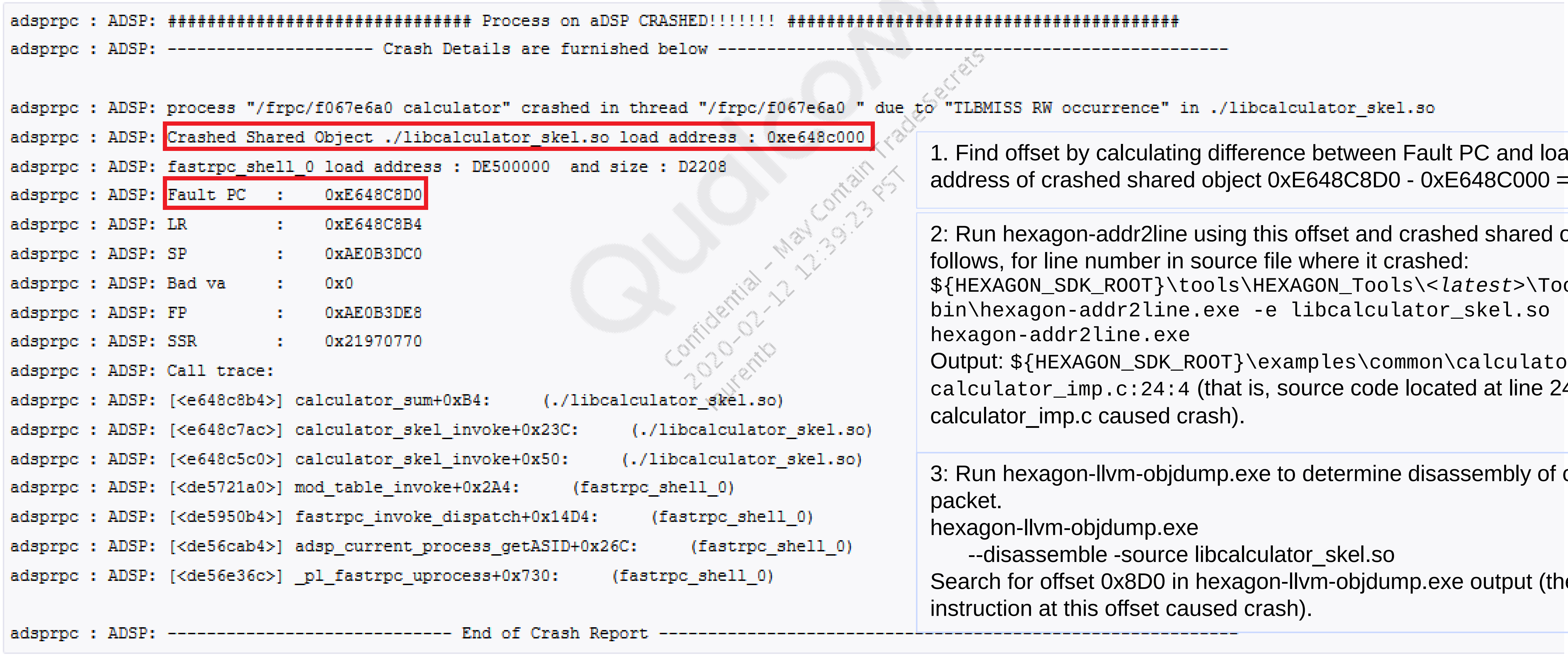
- Debugging user process exceptions

  When user process crashes on DSP, FastRPC exception handler collects required information from QuRT and flushes the messages to logcat

- Logcat details:
  - User process name
  - Thread name
  - Name of shared object and symbol offset (derived from PC during crash)
  - Kind of exception and details
  - Last known PC
  - Call trace

Debugging – User Process Crash log (cont.)

- Example of the logcat o/p



```
adsprpc : ADSP: ################################ Process on aDSP CRASHED!!!!!!! ########################################
adsprpc : ADSP: ---------------------- Crash Details are furnished below ----------------------------------------------------


adsprpc : ADSP: process "/frpc/f067e6a0 calculator" crashed in thread "/frpc/f067e6a0 " due to "TLBMISS RW occurrence" in ./libcalculator_skel.so
adsprpc : ADSP: Crashed Shared Object ./libcalculator_skel.so load address : 0xe648c000
adsprpc : ADSP: fastrpc_shell_0 load address : DE500000  and size : D2208
adsprpc : ADSP: Fault PC    :     0xE648C8D0
adsprpc : ADSP: LR          :     0xE648C8B4
adsprpc : ADSP: SP          :     0xAE0B3DC0
adsprpc : ADSP: Bad va      :     0x0
adsprpc : ADSP: FP          :     0xAE0B3DE8
adsprpc : ADSP: SSR         :     0x21970770
adsprpc : ADSP: Call trace:
adsprpc : ADSP: [<e648c8b4>] calculator_sum+0xB4:     (./libcalculator_skel.so)
adsprpc : ADSP: [<e648c7ac>] calculator_skel_invoke+0x23C:     (./libcalculator_skel.so)
adsprpc : ADSP: [<e648c5c0>] calculator_skel_invoke+0x50:     (./libcalculator_skel.so)
adsprpc : ADSP: [<de5721a0>] mod_table_invoke+0x2A4:     (fastrpc_shell_0)
adsprpc : ADSP: [<de5950b4>] fastrpc_invoke_dispatch+0x14D4:     (fastrpc_shell_0)
adsprpc : ADSP: [<de56cab4>] adsp_current_process_getASID+0x26C:     (fastrpc_shell_0)
adsprpc : ADSP: [<de56e36c>] _pl_fastrpc_uprocess+0x730:     (fastrpc_shell_0)


adsprpc : ADSP: ------------------------------ End of Crash Report ------------------------------------------------------------
```

1. Find offset by calculating difference between Fault PC and load address of crashed shared object 0xE648C8D0 - 0xE648C000 = 0x8D0.

2: Run hexagon-addr2line using this offset and crashed shared object, as follows, for line number in source file where it crashed:
`${HEXAGON_SDK_ROOT}\tools\HEXAGON_Tools\<latest>\Tools\bin\hexagon-addr2line.exe -e libcalculator_skel.so  0x8D0`
`hexagon-addr2line.exe`
Output: `${HEXAGON_SDK_ROOT}\examples\common\calculator\src\calculator_imp.c:24:4` (that is, source code located at line 24 in calculator_imp.c caused crash).

3: Run hexagon-llvm-objdump.exe to determine disassembly of crashing packet.
hexagon-llvm-objdump.exe
    --disassemble -source libcalculator_skel.so
Search for offset 0x8D0 in hexagon-llvm-objdump.exe output (the instruction at this offset caused crash).

- For details, see `${HEXAGON_SDK_ROOT}/docs/Debugging_Exceptions.html`

Debugging on Hexagon Simulator Using Eclipse IDE or Command Line

- QuRT simulator tests are run with run_main_on_hexagon utility
- QuRT simulator tests are shared objects, not executables (for example, benchmark_q.so instead of benchmark_q), so simulator tests are run only for dynamic variants (for example, hexagon_Debug_dynamic_toolv82_v65), unlike static variants in older SDKs
- `run_main_on_hexagon does dlopen()` of simulator test obtains `main()` address using `dlsym()` on shared object handle and calls `main()` of simulator test
- With this simulation framework, simulator tests should not link rtld, test_util, atomic because these libs are built into run_main_on_hexagon

  All SDK examples are updated to adopt this new simulation framework. For earlier projects, ensure simulator test is not linking rtld, test_util, atomic
- To link QuRT libs to a simulator test, add <simulator_test_name>_OSTYPE = QURT in hexagon.min

  See `${HEXAGON_SDK_ROOT}/examples/common/benchmark_v65/hexagon.min`
- For debugging using Eclipse IDE, see:

  `${HEXAGON_SDK_ROOT}/docs/benchmark_v65_simulator_debug.html`
- For debugging using command line, see:

  `${HEXAGON_SDK_ROOT}/docs/benchmark_v65_cmdline_debug.html`
- For more information about simulator debugging, see:

  `${HEXAGON_SDK_ROOT}/docs/Debugging_Simulator.html`

Debugging on Target Device Crash and Stability Issues

- QPST – Crash dump collection
- adspcrashman – RAM dump parsing
- TRACE32 – Dump loading, scripts
- For more information, see *Qualcomm® Hexagon™ aDSP and cDSP Stability Debugging Guide* (80-N4597-2)

Utility APIs – qprintf

- SDK provides qprintf library
- Qprintf library provides different APIs to display scalar and vector register in various formats from C/C++ and Assembly code as log messages
- For more details on the library, see *Qualcomm® qprintf Library* (80-VB419-109) (also available at `${HEXAGON_SDK_ROOT}/libs/common/qprintf`)
- For examples on using the library, see the code example in: `${HEXAGON_SDK_ROOT}/examples/common/qprintf_example`

Signing

- Unsigned PD
- Development signing
  - Algorithm development
  - Internal testing
- Production signing
  - Sub CA
  - Static hashing

Unsigned PD

- Clients can request to offload signature-free dynamic shared objects to cDSP
- Signature-free dynamic shared object can run inside the unsigned PD on the cDSP
- Unsigned PD is a sandboxed, low-rights process that allows signature-free modules to run on cDSP
  - In the event of a compromise, sandbox prevents access to full system functionality and data
  - Unsigned PDs are designed to support general compute applications and have limited access to underlying drivers
- Unsigned PD available services
  - Thread creation and thread services
  - Timer creation and timer services
  - HVX contexts
  - Clock frequency controls
  - VTCM
  - Cache operations
  - Map HLOS memory allocated by corresponding HLOS application
- Unsigned PD limitations
  - Access to limited drivers (that is, UBWC/DMA and camera streamer are not available to unsigned PDs)
  - Thread priorities limited to predefined range

Unsigned PD (cont.)

- **Request signature-free offload**

  To request signature-free dynamic module offload, clients must make the following request:

```
#pragma weak remote_session_control
if (remote_session_control)
{
struct remote_rpc_control_unsigned_module data;
data.enable = 1;
data.domain = CDSP_DOMAIN_ID;
remote_session_control(DSPRPC_CONTROL_UNSIGNED_MODULE, (void*)&data, sizeof(data));
}
```

- Make request before calling any other FastRPC function
- A successful remote_session_control function allows clients to offload dynamic shared object to cDSP without signing

Development Signing

- Algorithm development
  - Testsig.so signature file must be generated and placed in device file system before standalone algorithm executable can be verified on target
  - Testsig.so is specific to each device and must be generated at least once for a device before target testing
  - After a device is signed, updated or new shared objects do not require device to be signed again
  - Works only on nonproduction target hardware platform or devices
- Internal testing
  - Sign algorithm shared object
  - Useful for internal mass testing before going into production because testsig.so generation not feasible for thousands of devices
  - Works only on nonproduction target hardware platform or devices

Development Signing – Algorithm Development

1. Set up the environment.

```
${HEXAGON_SDK_ROOT}>setup_sdk_env.cmd
Setting up the Hexagon SDK environment locally
Done
```

2. Run testsig.py.

```
${HEXAGON_SDK_ROOT}/tools/scripts>python testsig.py
```

`testsig-<serial num>.so` is generated and placed in `/system/vendor/lib/rfsa/adsp`

Development Signing – Internal Testing

- Elfsigner command to sign shared object:

```
python elfsigner.py -i INFILE -o [OUTDIR]
```

  - INFILE is the pathname of an ELF format shared object file to be signed
  - OUTDIR specifies the output folder pathname of signed output file

- Usage example:

```
python elfsigner.py -i dynmod.so -o signed/ Signs the input file
dynmod.so writes the signed output file to ./signed/dynmod.so

python elfsigner.py -i input/dynmod.so -o signed/ This command signs
the input file ./input/dynmod.so and writes the signed output file
to ./signed/dynmod.so
```

- Only works with nonproduction devices

Production Signing

- Static hashing method used for mass production signing of algorithm shared object so it works only with specific DSP firmware image being shipped
- Algorithm shared object is hashed, and hash key is stored in DSP firmware image upon recompilation
  - DSP firmware verifies hash key before loading algorithm shared object at runtime to authenticate
- DSP firmware must be recompiled when shared object implementation is changed

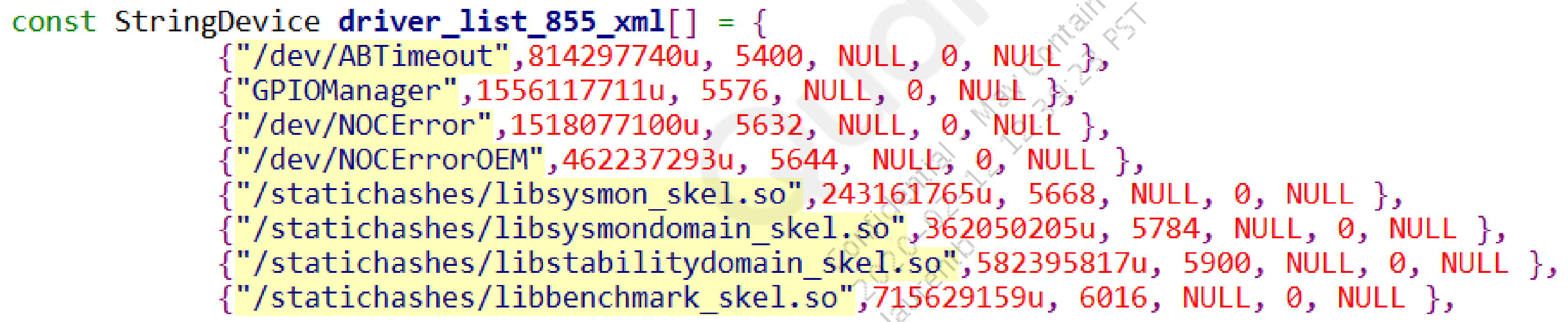
1. Set the OEM_ROOT path.
```
>set OEM_ROOT=<CDSP Build folder
>\cdsp_proc\hap\oem
```
2. Copy the skel files for hashing to:
```
dynamic_modules_hash:
<CDSP Build folder>\cdsp_proc\hap\oem\
build\dynamic_modules_hash
```
3. Compile cDSP build after replacing the SO file.
```
<CDSP Build folder>\cdsp_proc\build>
python build.py -c sdm855 -o all
```

Note: Code examples reflect the SDM855 chipset name.

Production Signing (cont.)

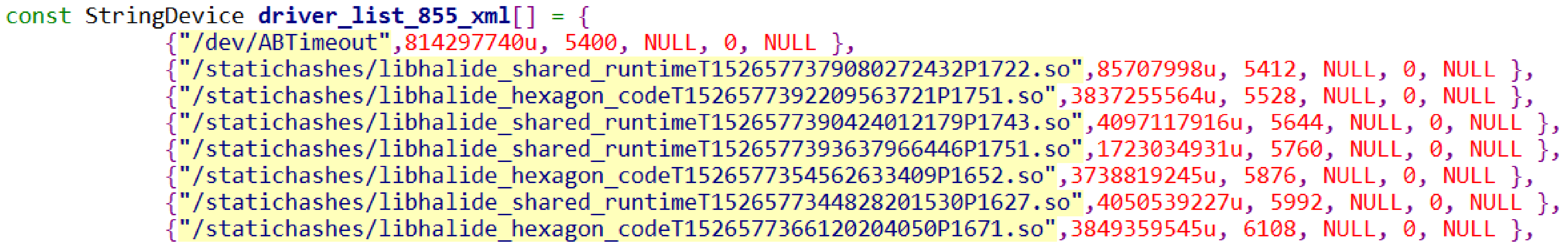
4. Verify that shared objects have been hashed correctly.
   a. Open: `<cdsp build root>\cdsp_proc\core\kernel\devcfg\build\core_cdsp_root_libs\qdsp6\855.cdsp.prod\ data\855_xml\devcfg.c`.
   b. Search for shared object name to confirm it appears with hash code.

Before compile:



```
const StringDevice driver_list_855_xml[] = {
            {"/dev/ABTimeout",814297740u, 5400, NULL, 0, NULL },
            {"GPIOManager",1556117711u, 5576, NULL, 0, NULL },
            {"/dev/NOCError",1518077100u, 5632, NULL, 0, NULL },
            {"/dev/NOCErrorOEM",462237293u, 5644, NULL, 0, NULL },
            {"/staticahshes/libsysmon_skel.so",243161765u, 5668, NULL, 0, NULL },
            {"/statichashes/libsysmondomain_skel.so",362050205u, 5784, NULL, 0, NULL },
            {"/statichashes/libstabilitydomain_skel.so",582395817u, 5900, NULL, 0, NULL },
            {"/statichashes/libbenchmark_skel.so",715629159u, 6016, NULL, 0, NULL },
```

After compile:



```
const StringDevice driver_list_855_xml[] = {
            {"/dev/ABTimeout",814297740u, 5400, NULL, 0, NULL },
            {"/statichashes/libhalide_shared_runtimeT1526577379080272432P1722.so",85707998u, 5412, NULL, 0, NULL },
            {"/statichashes/libhalide_hexagon_codeT1526577392209563721P1751.so",3837255564u, 5528, NULL, 0, NULL },
            {"/statichashes/libhalide_shared_runtimeT1526577390424012179P1743.so",4097117916u, 5644, NULL, 0, NULL },
            {"/statichashes/libhalide_shared_runtimeT1526577393637966446P1751.so",1723034931u, 5760, NULL, 0, NULL },
            {"/statichashes/libhalide_hexagon_codeT1526577354562633409P1652.so",3738819245u, 5876, NULL, 0, NULL },
            {"/statichashes/libhalide_shared_runtimeT1526577344828201530P1627.so",4050539227u, 5992, NULL, 0, NULL },
            {"/statichashes/libhalide_hexagon_codeT1526577366120204050P1671.so",3849359545u, 6108, NULL, 0, NULL },
```

Production Signing (cont.)

5. Replace cDSP firmware image on device (shared objects only work with this image).

```
set ADSPPATH= <This has to be cdsp build root folder location>
adb wait-for-device
adb root
adb wait-for-device
adb remount
adb shell mount -o rw,remount /firmware
adb shell mount -o rw,remount /dsp
adb shell rm /firmware/image/cdsp*
adb shell rm -r /dsp/cdsp/*
adb push %ADSPPATH%\cdsp_proc\obj\qdsp6v5_ReleaseG\845.cdsp.prod\LA\system\etc\firmware\. /firmware/image
adb push %ADSPPATH%\cdsp_proc\build\dynamic_modules\845.cdsp.prod\. /dsp/cdsp
adb push %ADSPPATH%\cdsp_proc\build\ms\servreg\845.cdsp.prodQ\cdspr.jsn /firmware/image
adb shell sync
sleep 1
adb reboot
adb wait-for-device root
adb shell input keyevent 82
adb wait-for-device remount
adb wait-for-device shell "cd /sys/devices/soc0; echo 12 > select_image && echo \"`cat image_version`\""
adb wait-for-device shell "cd /sys/devices/soc0; echo 16 > select_image && echo \"`cat image_version`\""
adb wait-for-device shell "cat /firmware/verinfo/ver_info.txt"
```

Note: The cDSP image folder location might change in an OEM environment. Check the folder location first.

Sub-CA

- CASS USB token (dongle) signing is replaced by sub-CA
- USB token is not renewed after expiration
- Customer should migrate to sub-CA and follow procedure listed in *Signing: Sub-CA* (80-PD867-113) for requesting and signing using sub-CA
- No new agreement is required for customers who have already signed CASS agreement to get the USB token
- New customers should follow up with Qualcomm Product Management team for CASS agreement details

Library to Help Port Floating-Point Arithmetic

- HVX supports only fixed-point arithmetic
- Qualcomm Fixed-Point Library
  - Provides assistance with converting floating-point operations to fixed-point operations
  - For more details see:

    `${HEXAGON_SDK_ROOT}/docs/images/80-VB419-69_Qualcomm_Fixed_Point_Library.pdf`

    `${HEXAGON_SDK_ROOT}/docs/images/80-VB419-107_qfxp_Library_Training.pdf`
- Qualcomm Math (qmath) Library
  - Provides an HVX vectorized alternative to floating-point operations that are not feasible to convert to fixed-point operations
  - Significantly faster than the scalar Hexagon floating-point unit
  - For more details, see:

    `${HEXAGON_SDK_ROOT}/docs/images/80-VB419-105_Qualcomm_Math_Library.pdf`

Halide – An Image Processing Language

- A domain-specific programming language for writing high-performance image processing code on modern processors

  Optimized image processing pipelines are difficult to write
- Halide depends on LLVM toolset to generate HVX binaries
  - Allows separation of algorithm from computational details
  - Enables rapid authoring and evaluation of optimized pipelines
- For more information, go to http://halide-lang.org

Why Halide?

- Addresses challenges in writing image algorithms with traditional languages
  - Fast image processing pipelines are difficult to write because they include:
    - Definition of pipeline stages
    - Optimization
    - Parallelism – vectorization, multi-threading
    - Packetization (DSP instruction level parallelism)
    - Memory hierarchy – tiling, fusion, prefetching, software pipelining
  - Traditional development environments make optimizing image processing kernels difficult
  - Programming languages
    - Parallelism, tiling, optimizations are difficult to express in traditional languages
    - Substantial time spent on expressing and debugging algorithm
  - Libraries
    - Optimized kernels such as OpenCV and OpenVX might not integrate properly into efficient pipelines due to lack of computation and storage reuse
    - Library calls add optimization barrier
- Addresses challenges in time and effort it takes to tune algorithms
  - Assembly is time-intensive and requires deep knowledge of low-level processor details
  - Compiler intrinsics require significant learning curve, and code might not be reusable across processors

Halide and HVX

- Halide supports many targets:
  - Hexagon/HVX
  - X86/SSE
  - Arm v7/NEON
  - CUDA
  - OpenCL
  - Native Client
- Halide:
  - Allows algorithm to be dispatched on Hexagon/HVX with a code directive
  - Is open source
  - Depends on LLVM toolset to generate Hexagon/HVX binaries
  - Allows programmers with little knowledge of HVX internals to focus on authoring and tuning image algorithms, which allows more programmers to program for HVX
  - Eases offloading of image kernels to HVX
  - Reduces effort required to prototype and gauge profitability of kernel running on HVX

Halide and HVX (cont.)

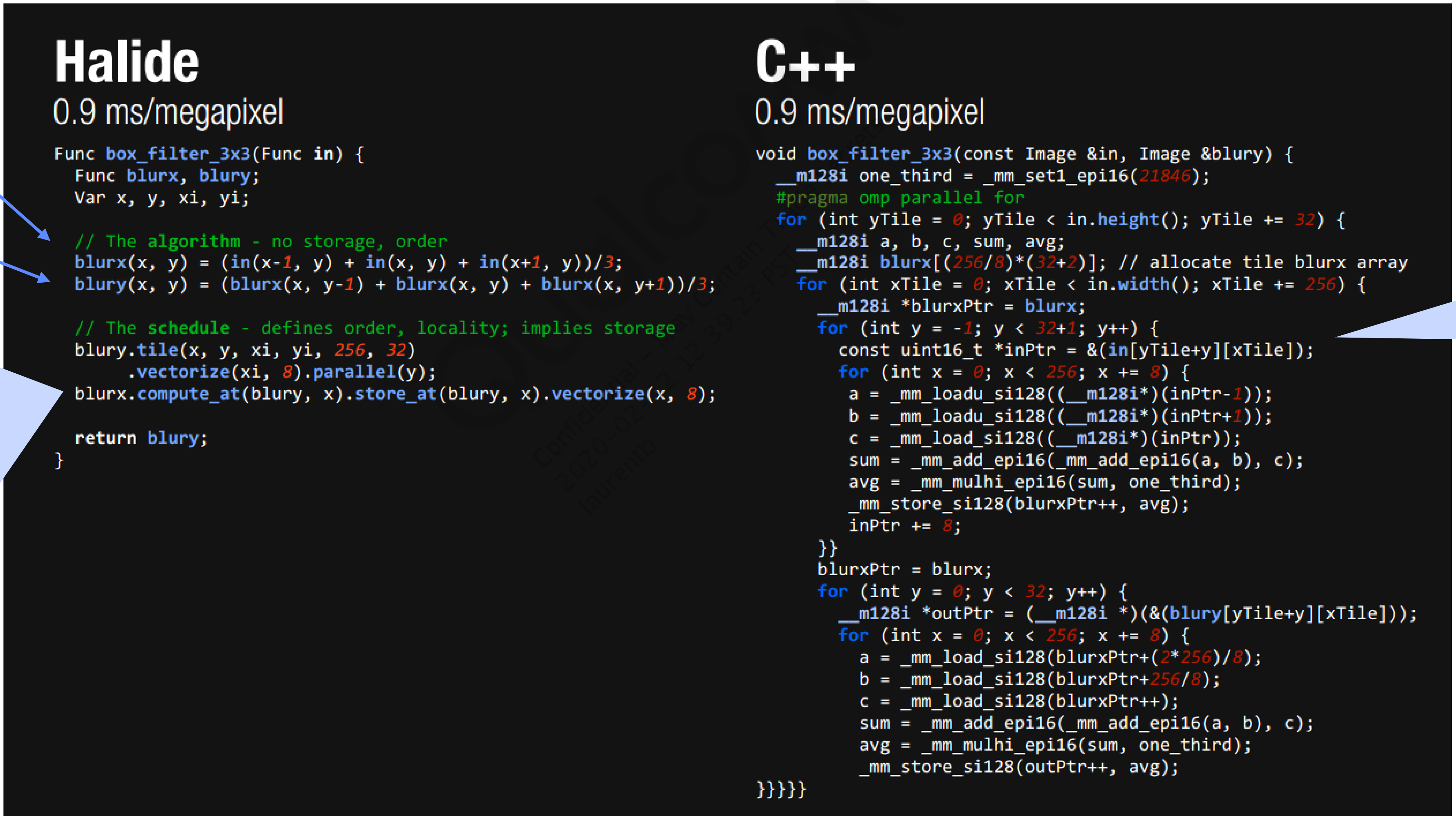
- Halide language is derivative of C/C++
- Halide on HVX uses QuIC LLVM compiler for code generation
  - QuIC LLVM compiler is tuned extensively for Hexagon/HVX
- Leverages optimizations in Hexagon/HVX LLVM compiler
  - Can shield programmer from low-level optimizations required for performance
- Halide programs consist of two major components:
  - Algorithm – Specifies what will be computed at pixel level
  - Schedule – Specifies how computation should be organized
- Separation is key and it enables:
  - Authoring an algorithm without complexity of computational organization
  - Tuning and experimenting with computational organization to achieve high performance
  - Generating highly optimized compiled code
    - Programmer guides compiler, explicitly specifying high-level optimizations

Programming in Halide

Separates *algorithm* from *schedule*

Simple syntax to restructure computation

Programmers can quickly restructure computation to find optimal performance



Halide

0.9 ms/megapixel

```
Func box_filter_3x3(Func in) {
  Func blurx, blury;
  Var x, y, xi, yi;

  // The algorithm - no storage, order
  blurx(x, y) = (in(x-1, y) + in(x, y) + in(x+1, y))/3;
  blury(x, y) = (blurx(x, y-1) + blurx(x, y) + blurx(x, y+1))/3;

  // The schedule - defines order, locality; implies storage
  blury.tile(x, y, xi, yi, 256, 32)
       .vectorize(xi, 8).parallel(y);
  blurx.compute_at(blury, x).store_at(blury, x).vectorize(x, 8);

  return blury;
}
```

C++

0.9 ms/megapixel

```
void box_filter_3x3(const Image &in, Image &blury) {
  __m128i one_third = _mm_set1_epi16(21846);
  #pragma omp parallel for
  for (int yTile = 0; yTile < in.height(); yTile += 32) {
    __m128i a, b, c, sum, avg;
    __m128i blurx[(256/8)*(32+2)]; // allocate tile blurx array
    for (int xTile = 0; xTile < in.width(); xTile += 256) {
      __m128i *blurxPtr = blurx;
      for (int y = -1; y < 32+1; y++) {
        const uint16_t *inPtr = &(in[yTile+y][xTile]);
        for (int x = 0; x < 256; x += 8) {
         a = _mm_loadu_si128((__m128i*)(inPtr-1));
         b = _mm_loadu_si128((__m128i*)(inPtr+1));
         c = _mm_load_si128((__m128i*)(inPtr));
         sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
         avg = _mm_mulhi_epi16(sum, one_third);
         _mm_store_si128(blurxPtr++, avg);
         inPtr += 8;
      }}
      blurxPtr = blurx;
      for (int y = 0; y < 32; y++) {
        __m128i *outPtr = (__m128i *)(&(blury[yTile+y][xTile]));
        for (int x = 0; x < 256; x += 8) {
          a = _mm_load_si128(blurxPtr+(2*256)/8);
          b = _mm_load_si128(blurxPtr+256/8);
          c = _mm_load_si128(blurxPtr++);
          sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
          avg = _mm_mulhi_epi16(sum, one_third);
          _mm_store_si128(outPtr++, avg);
}}}}}
```

Complex loop nest with vector intrinsics

Halide – HVX Benefits

- Halide makes high-performance image processing pipelines easy to write and optimize
- Halide takes care of following aspects of HVX programming:
  - Expresses algorithms in terms of operations on a pixel
  - Eliminates need to write in terms of intrinsics or assembly
  - Schedules instructions performed by compiler
  - Leverages QTI LLVM compiler technology
  - Allows for seamless heterogeneous computing using Hexagon offloading model
  - Eliminates need to use FastRPC or IDL
  - Simplifies work through use of a *hexagon()* scheduling clause

Compute Resource Manager for cDSP

Compute Resource Manager for cDSP

- Framework providing APIs to request and release compute resources (for example, HVX, VTCM, hardware threads, memory buses) and enable serialization
- Sharing compute resources while running applications concurrently causes overall performance to deteriorate. Framework allows participating applications to run workloads in serial or batch (one after the other) mode
- Nonparticipating clients (that is, clients not requesting serialization), can still run concurrently with a serialized batch
- VTCM Manager allocation APIs supported since SDM845 are still supported
- Compute resource APIs include option to allocate VTCM with requesting serialization, for best performance

Compute Resource Manager for cDSP – APIs

- Framework APIs defined in `${HEXAGON_SDK_ROOT}/incs/HAP_compute_res.h`
- Acquire/Release APIs
  - Acquire resources and/or participate in serialization

    ```
    unsigned int HAP_compute_res_acquire(compute_res_attr_t* res_info,
                                         unsigned int timeout_us)
    ```

    Accepts a prepared attribute structure (res_info) and returns a context id for successful request within provided timeout (microseconds). Returns 0 for failures. Attribute structure (res_info) should be prepared using helper APIs prior to calling this API
  - Release resources and not participate

    ```
    int HAP_compute_res_release(unsigned int context_id)
    ```

    Releases the resources assigned to the context bearing ID, context_id, assigned by a previous acquire call. Until released, resources remain assigned. User must call API after requested resource is no longer needed
- Helper APIs
  - Initializes the attribute structure pointed to by compute_res_attr_t* to the defaults

    ```
    int HAP_compute_res_attr_init(compute_res_attr_t* res_info)
    ```

    User must call API prior to setting specific resource property using helper APIs

    Default values: Don't participate in serialization, No VTCM request
  - Sets serialization option based on input (b_serialize) in the resource structure

    ```
    int HAP_compute_res_attr_set_serialize(compute_res_attr_t* res_info, unsigned char b_serialize)
    ```

Compute Resource Manager for cDSP – APIs (cont.)

- **VTCM allocation**
  - HAP_compute_res_attr_set_vtcm_param

```
int HAP_compute_res_attr_set_vtcm_param(compute_res_attr_t* res_info, unsigned int vtcm_size,
                                        unsigned char b_vtcmSinglePage)
```

    Sets VTCM request parameters in the provided resource attribute structure. User calls to request VTCM chunk in acquire call. `HAP_compute_res_attr_init()` API resets VTCM request attributes to 0 (no VTCM request) in the resource attribute structure

  - HAP_compute_res_attr_get_vtcm_ptr

```
void* HAP_compute_res_attr_get_vtcm_ptr(compute_res_attr_t* res_info)
```

    Retrieves VTCM memory allocation pointer after successful acquisition from provided resource attribute structure. API can be called after successful `HAP_compute_res_acquire()` call to retrieve allocated VTCM pointer

- For more details, see `${HEXAGON_SDK_ROOT}/docs/compute_resource_manager.html`

VTCM Manager

- Hexagon V66 includes local TCM, called *Vector TCM* (VTCM)
  - Performance is better than L2-TCM
    - About twice the bandwidth
    - About half the store-to-load latency
    - Lower power
  - Only visible to cDSP loads and stores
    - Not connected to UBWC or DMA
  - VTCM size is 256 kB for SM8250
- Use cases
  - New HVX scatter-gather instructions can only operate on VTCM memory
    - Typically, they perform HVX-based memcpy between L2 and VTCM before and after scatter-gather
    - Single page requests are mandatory and aligned to closest possible page size, such as 4 kB, 16 kB, 64 kB, 256 kB
  - Scratch buffers for camera streaming use cases that exceed 256 kB L2 requirement
  - Scratch or temporary memory for other algorithms that can benefit from local storage
- VTCM Manager provides APIs to allocate, free, and query VTCM availability
- APIs defined in `${HEXAGON_SDK_ROOT}/incs/HAP_vtcm_mgr.h`

VTCM Manager – APIs

- Query VTCM size defined on target:

  `int HAP_query_total_VTCM (unsigned int* page_size, unsigned int* page_count)`

- Query VTCM allocation status:

  `int HAP_query_avail_VTCM (unsigned int* avail_block_size, unsigned int* max_page_size, unsigned int* num_pages)`

- Request VTCM memory of specified size and single page requirement:

  `void* HAP_request_VTCM (unsigned int size, unsigned int single_page_flag)`

  Note: Single page requests are mandatory for scatter-gather operations because they must be contained within a single page of memory. If `single_page_flag` is set to 1, size is aligned to closest possible page size, such as 4 kB, 16 kB, 64 kB, 256 kB.

- Release a successful request for VTCM memory by providing pointer to previously allocated VTCM block:

  `int HAP_release_VTCM (void* pVA)`

- Request VTCM memory of size and single page requirement with a timeout option:

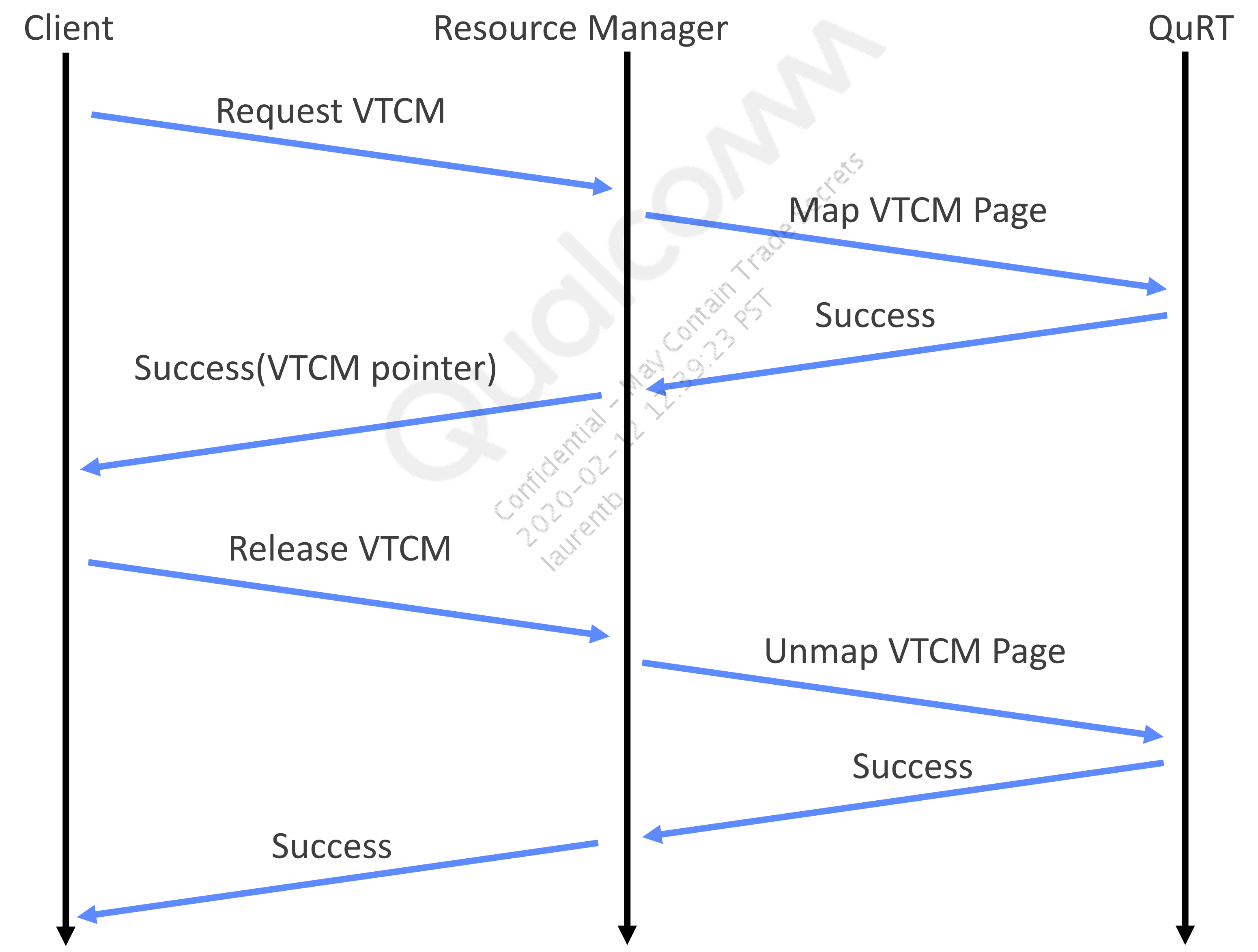
  `void* HAP_request_async_VTCM (unsigned int size, unsigned int single_page_flag, unsigned int timeout_us)`

  API can be used to wait until provided time out for request is fulfilled. Calling thread is suspended until requested VTCM memory is available or until the timeout, which ever happens first

  Note: Possibility of a deadlock when calling this API, if same thread is holding part of or entire VTCM memory prior to this call. This API is not supported from SecurePD and CPZ processes.

- For details, see `${HEXAGON_SDK_ROOT}/docs/VTCM Manager.html`

VTCM – Call Flow



Hexagon Training Resources

- Documentation and training videos are available on CreatePoint
- For a complete list, see *References*

References

References – Documents

Title	DCN
Qualcomm Technologies, Inc.	
Qualcomm® Hexagon™ V66 Programmer's Reference Manual	80-N2040-42
Qualcomm® Hexagon™ V66 HVX Programmer's Reference Manual	80-N2040-44
Hexagon Access Audio Dynamic Loading Customization and Debug Guide AVS.ADSP.2.7 and Later	80-NF772-37
Qualcomm® FastCV™ Integration	80-NE989-7
Qualcomm® Snapdragon™ Computer Vision Overview for MSM8996/MSM8998	80-NL315-9
HQV™/VPP Feature Description	80-P1881-1
FastRPC Training	80-N7039-1
Qualcomm® Hexagon™ aDSP and cDSP Stability Debugging Guide	80-N4597-2
Halide for HVX User Guide	80-PD002-1
Halide for Qualcomm® Hexagon™ Vector eXtensions (HVX)	80-PD002-2
Qualcomm® Hexagon™ gprof Profiler User Guide	80-N2040-29
Qualcomm® Hexagon™ Secure cDSP Software Overview for SM8150	80-PK723-1
Dynamic Loading in Secure PD	80-PR150-1

Note: The documents are available on CreatePoint.

References – Documents (cont.)

Title	DCN
Qualcomm Technologies, Inc.	
Qualcomm® Hexagon™ Profiler User Guide	80-N2040-10
Qualcomm® qprintf Library	80-VB419-109
Signing: Sub-CA	80-PD867-113

Note: The documents are available on CreatePoint.

References – Training Videos

Hexagon Processor	DCN
Hexagon Programming: Processor Architecture	VD80-VB419-41A (Part 1) VD80-VB419-41B (Part 2) VD80-VB419-41C (Part 3) VD80-VB419-41D (Part 4) VD80-VB419-41E (Part 5) VD80-VB419-41F (Part 6)
Hexagon Programming: Software Development Tools	VD80-VB419-42A (Part 1) VD80-VB419-42B (Part 2) VD80-VB419-42C (Part 3) VD80-VB419-42D (Part 4) VD80-VB419-42E (Part 5) VD80-VB419-42F (Part 6)
Hexagon Programming: Processor Architecture	VD80-VB419-44A (Part 1) VD80-VB419-44B (Part 2) VD80-VB419-44C (Part 3) VD80-VB419-44D (Part 4) VD80-VB419-44E (Part 5) VD80-VB419-44F (Part 6) VD80-VB419-44G (Part 7)

Hexagon Processor	DCN
Hexagon Programming: Assembly Programming	VD80-VB419-45A (Part 1) VD80-VB419-45B (Part 2) VD80-VB419-45C (Part 3) VD80-VB419-45D (Part 4) VD80-VB419-45E (Part 5)
Hexagon Programming: Memory System	VD80-VB419-60A (Part 1) VD80-VB419-60B (Part 2) VD80-VB419-60C (Part 3) VD80-VB419-60D (Part 4) VD80-VB419-60E (Part 5) VD80-VB419-60F (Part 6)

Note: The training videos are available on CreatePoint.

References – Training Videos (cont.)

HVX	DCN
Hexagon Core Review	VD80-P4366-1
HVX Architecture Review	VD80-P4366-2A (Part 1) VD80-P4366-2B (Part 2)
HVX Tools	VD80-P4366-3
Simple Example	VD80-P4366-4A (Part 1) VD80-P4366-4B (Part 2)
Unusual Features	VD80-P4366-5A (Part 1) VD80-P4366-5B (Part 2)
Exploring HVX Using 3x3 Convolution	VD80-P4366-6A (Part 1) VD80-P4366-6B (Part 2)
Microarchitecture	VD80-P4366-7A (Part 1) VD80-P4366-7B (Part 2)
Assembly Programming	VD80-P4366-8A (Part 1) VD80-P4366-8B (Part 2)
Assembly 3x3 Convolution Example	VD80-P4366-9

HVX	DCN
Advanced Algorithms	VD80-P4366-10A (Part 1) VD80-P4366-10B (Part 2) VD80-P4366-10C (Part 3)
Advanced Instructions – Data,	VD80-P4366-11A (Part 1) VD80-P4366-11B (Part 2) VD80-P4366-11C (Part 3)
Advanced Instructions – Arithmetic	VD80-P4366-12A (Part 1) VD80-P4366-12B (Part 2) VD80-P4366-12C (Part 3)
OS Concerns	VD80-P4366-13
HVX Limitations	VD80-P4366-14

Halide	DCN
Halide Programming	VD80-PG213-1A (Part 1) VD80-PG213-1B (Part 2) VD80-PG213-1C (Part 3) VD80-PG213-1D (Part 4)

Note: The training videos are available on CreatePoint.