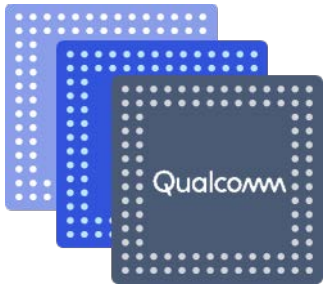
Qualcomm® Hexagon™ Vector eXtensions (HVX) Overview for SDM670/SDM710

Qualcomm Technologies, Inc.

80-PD126-49 Rev. C

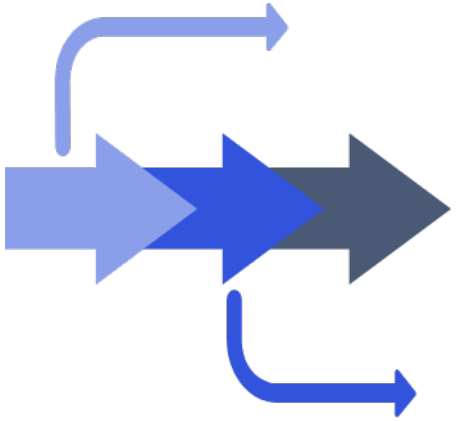


Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

Revision History

Revision	Date	Description
A	September 2017	Initial release
B	December 2017	Updated slide 7- Snapdragon DSP Evolution – Leverages Low-power Processing
C	April 2018	Updated title and other minor SDM710 changes

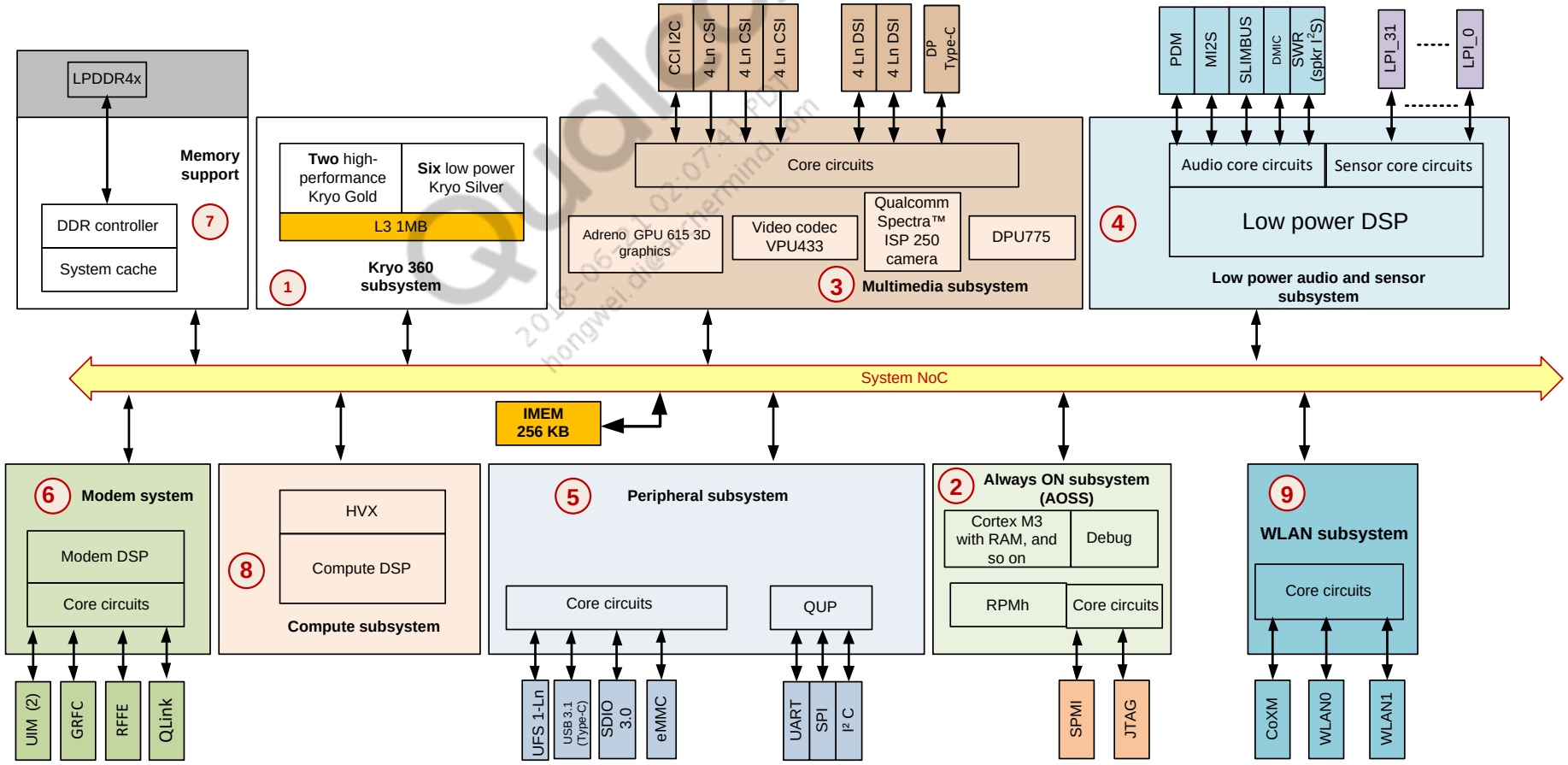
Contents



Introduction

SDM670/SDM710 Architecture Overview

Major architectural blocks:
1. Kryo subsystem
2. Resource and power management system (RPM)
3. Multimedia subsystem (MMSS)
4. Low power audio and sensor subsystem
5. Peripheral subsystem (PSS)
6. Modem system
7. Memory support and the bus system that supports all major blocks
8. Compute subsystem
9. WLAN subsystem



Snapdragon DSP Evolution – Leverages Low-power Processing

SDM660

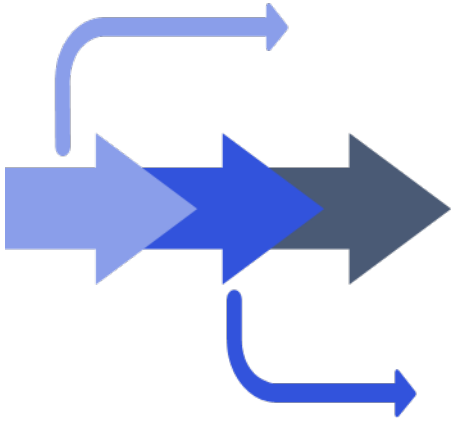
- Compute DSP
  - 787 MHz, 512 kB L2 cache
  - Dual-HVX
  - Imaging and machine learning

Shared
- Audio/Voice DSP
  – Shared with Sensors
- 1000 MHz Turbo
- Low-power island
- 512 kB L2 cache
- Always-aware hub

SDM670/SDM710

- Compute DSP
  - 1200 MHz, 512 kB L2 cache, 256 kB VTCM
  - Dual-HVX
  - Imaging and machine learning
  - Scatter/gather
  - Enhanced CNN support

Shared
- Audio/Voice DSP
  – Shared with Sensors
- 1200 MHz Turbo
- Low-power island
- 512 kB L2 cache
- Always-aware hub

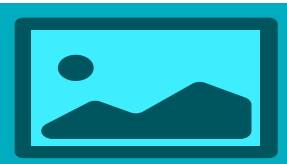


Hexagon Compute DSP (cDSP)

Hexagon cDSP – HVX Use Cases

- cDSP in SDM670/SDM710 ensures improved image processing, computer vision (CV), machine learning, and camera streaming

Image enhancement – Camera, Still, Video modes



CV and extended reality

Video

Camera streaming

Machine learning

Hexagon DSP Evolution – Application and cDSP

	Hexagon V60 (SDM660)	Hexagon V60 HVX (SDM660)	Hexagon V65 (SDM670/SDM710)	Hexagon V65 HVX (SDM670/SDM710)
Threads/core	4 threads	+2, 128 byte vector threads or +4, 64 byte vector threads	4 threads	+2, 128 byte vector threads or +4, 64 byte vector threads
Processor clock	748 MHz nominal (998 MHz turbo)	+650 MHz nominal (+787 MHz turbo)	950 MHz nominal (1200 MHz turbo)	+950 MHz nominal (+1200 MHz turbo)
Arithmetic	Fixed and floating point	Fixed point	Fixed and floating point	Fixed point
Caches	• L1 instruction: 16 K • L1 data: 32 K • L2: 512 K	Same L2 as core	• L1 instruction: 16 K • L1 data: 16 K • L2: 512 K	• Same L2 as core • 256 kB VTCM
Performance • Peak OPS • Dhrystone • Coremark	• 8 MMAC/MHz (16-bit fixpt) 16 MMAC/MHz (8-bit fixpt) • 4 MFLOP (SP)/MHz • 7.2 DMIPS/MHz • 14.4 C'mark/MHz	Dual-HVX • +64 MMAC/MHz (16-bit fixpt) • +256 MMAC/MHz (8-bit fixpt) • +256 Mops/MHz (16-bit fixpt)	• 8 MMAC/MHz (16-bit fixpt) 16 MMAC/MHz (8-bit fixpt) • 4 MFLOP (SP)/MHz • 7.2 DMIPS/MHz • 14.4 C'mark/MHz	Dual-HVX • +64 MMAC/MHz (16-bit fixpt) • +512 MMAC/MHz (8-bit fixpt) • +256 Mops/MHz (16-bit fixpt)
Architecture	Simultaneous multithreading (SMT)	–	SMT	–
Instruction/system enhancements	• Multimedia enhanced • Coprocessor capability • Automatic power collapse (APC)	• HVX • Dual-clusters	• Multimedia enhanced • Coprocessor capability • APC	• HVX • Dual-clusters • Scatter/gather, DMA, Universal bandwidth compression (UBWC)

Note: The new additions/changes in SDM670/SDM710 are indicated in green.

Frequency Evolution – cDSP

Voltage setting vs. maximum frequency	MSM8996	SDM660	SDM670/SDM710[1]
Turbo	825	787	1200
Nominal	650	650	950
SVS_L1	–	–	800
SVS	500	480	580
LowSVS	300	300	440
MinSVS	144	144	280

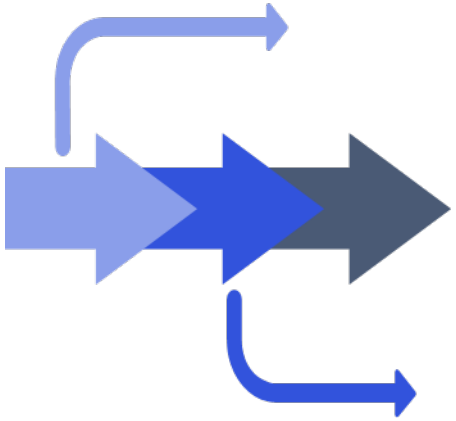
[1] Voltage frequencies are still under evaluation and they are subjected to change.
New instructions, vector TCM, and scatter-gather allow for comparable tasks at lower frequency requirements.
Values are subject to change.

cDSP Feature Summary

Feature	Description
UBWC and DMA 1.0	• UBWC provides bandwidth compression between the cDSP and DDR for HVX. • DMA 1.0 provides a DMA block to access UBWC frames in the DDR and present them to the cDSP as linear frames. This feature allows the cDSP to alter the linear frames, and then transfer them back to the DDR in UBWC frames. • The DMA firmware driver provides a high-level wrapper mode API to enable UBWC support for UBWC-to-linear and linear-to-UBWC data conversion for HVX use cases.
Vector TCM (VTCM)	Local TCM (256 kB) added in Hexagon V65.
Scatter-gather support	• New HVX instructions and usage examples for scatter-gather support. • Supports nonlinear vectors.
HVX contexts and concurrency	RTOS management of HVX contexts and concurrency: • HVX contexts are not reserved or locked; applicable only for 128-byte mode, for 64-byte mode locks are reserved. • RTOS can pre-empt and context-save HVX contexts. • 128-byte mode is the default, and algorithms call HVX instructions.
Migrating to 128 byte mode for future forward compatibility	64-byte/51- bit mode to be deprecated on chipsets later than SDM670/SDM710. Hence, migrating to 128-byte mode is essential.
dspCV_init is deprecated	• dspCV_init is deprecated and dspCV are not linked from application processor. • Link to dspCV is on DSP side, and worker pool is available (created in a static constructor) together with HVX APIs.

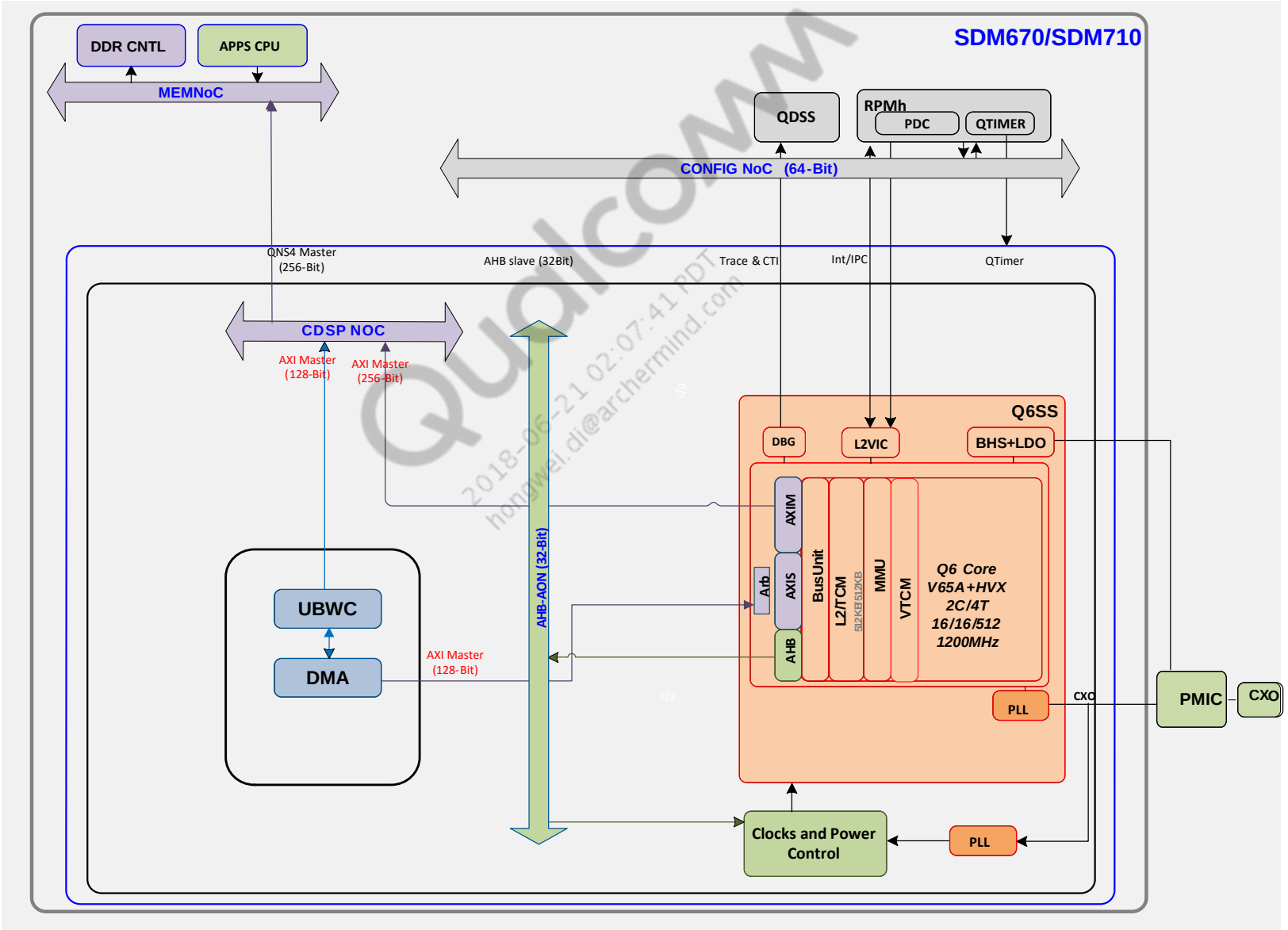
cDSP Feature Summary (cont.)

Feature	Description
New Hexagon Access DCVS v2 APIs for clock and bus voting	• Vote clocks via HAP_power with DCVS_v2. • Default clock values for a compute session are set even if client does not vote for clocks or bus.
dspCV_concurrency is deprecated	No need for dspCV_concurrency usage.
SDK 3.x	• Provides examples and a development environment for UBWC and scatter-gather functionality. • SDK 3.3 is currently supported for SDM670/SDM710 development.
Camera streaming – four tap points	• Additional four tap points in the camera pipeline to leverage HVX camera streaming functionality for preview and video stream use cases. • Data is inserted and output at multiple points in an IFE pipeline.
Dedicated bus for camera streaming on the cDSP	• Enables preprocessing on camera streams before ISP. • Supports a dual camera use case involving two sensors and ISPs.
Dedicated high-speed bus for data transfer from the DDR to the cDSP	cDSP is directly connected to the DDR.
I/O coherency	Low RPC latency with input/output coherence support.
FastCV software development kit with HVX acceleration	FastCV APIs are accelerated using HVX.



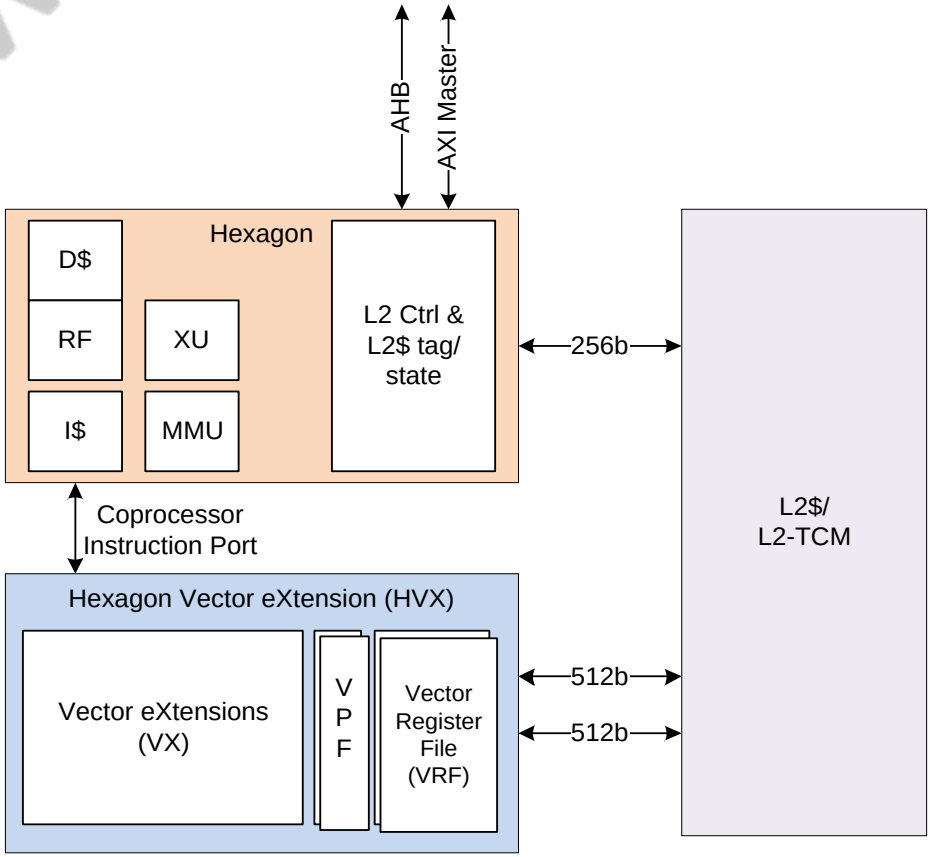
cDSP Hardware Architecture

cDSP Hardware Block Diagram

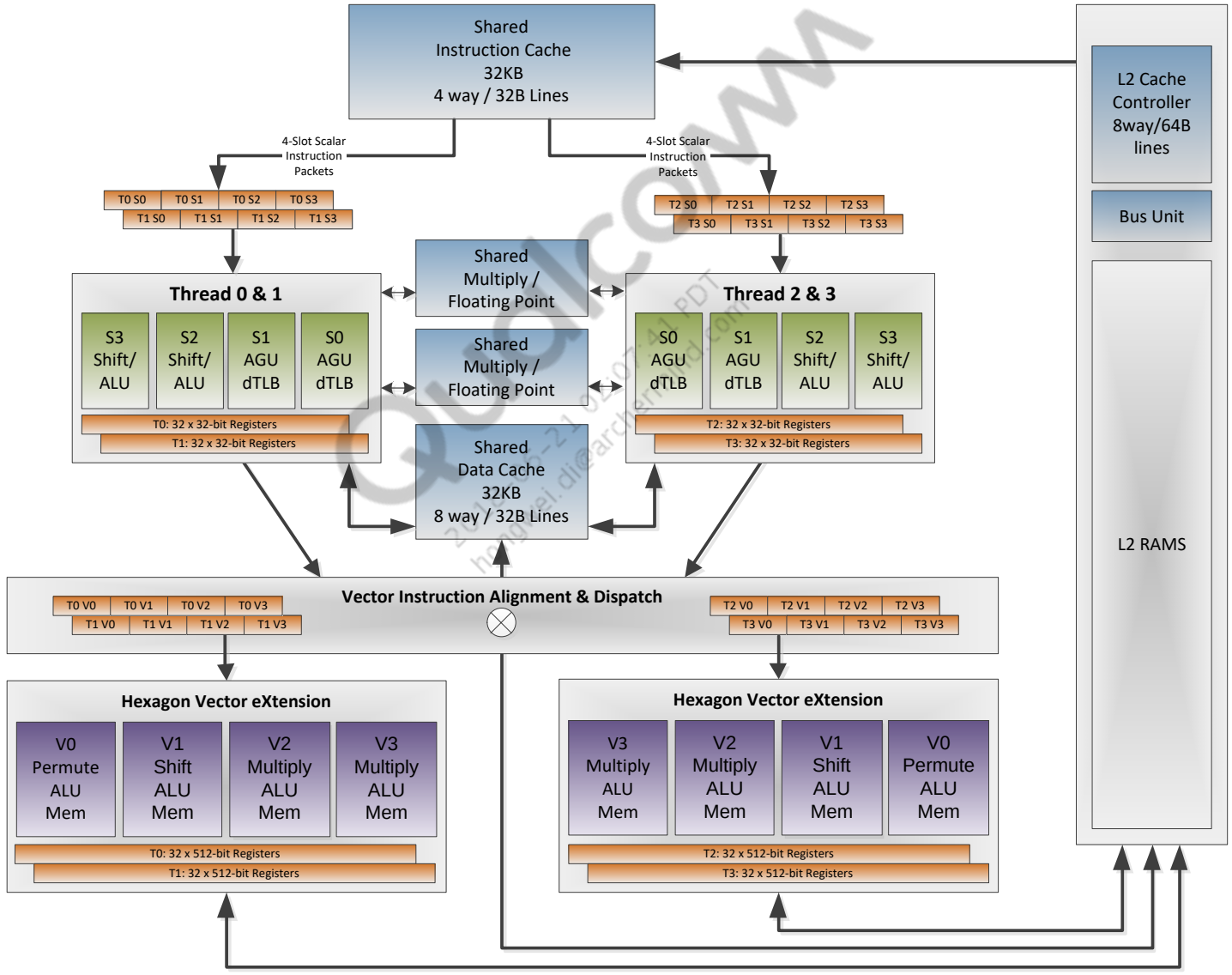


Hexagon cDSP with HVX

- Interleaved multithread VLIW DSP
- Combines best of DSP with general-purpose processor ease-of-programming
- Provides excellent control code as well as signal processing
- Focuses on throughput and efficiency instead of
  peak single-thread performance
- HVX is an optional component of the Hexagon DSP
  - Adds wide-vector (512-bit or 1024-bit) SIMD support
- Instruction extensions to the Hexagon v60 processor architecture support vector-operations on data up to 1024 bits wide, which are implemented in a coprocessor
- Suitable for high-performance imaging, compute, and machine learning applications

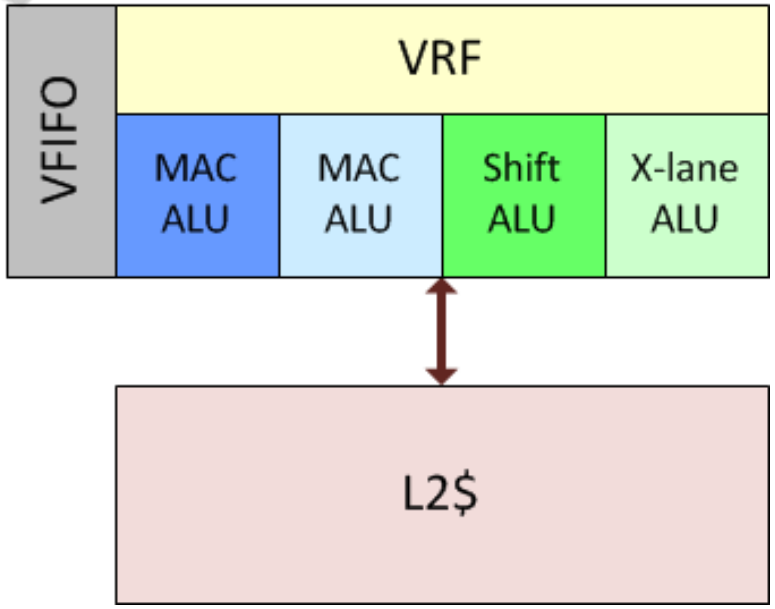


Hexagon cDSP with HVX (cont.)



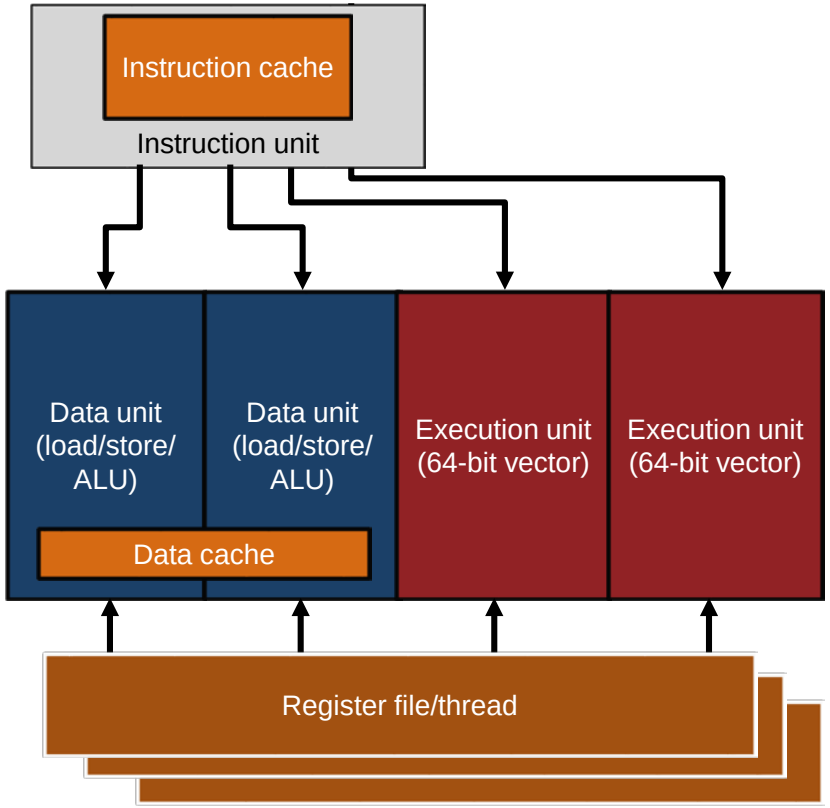
Hexagon cDSP with HVX (cont.)

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



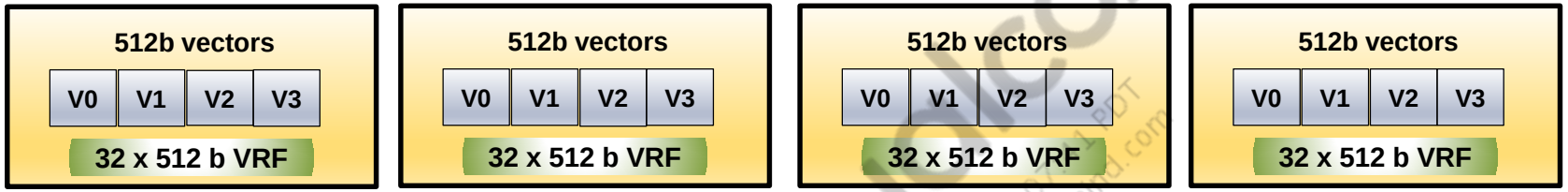
cDSP Hexagon Scalar Core

- Orthogonal four-issue VLIW: 1 to 4+ instructions per cycle
- Dual 64-bit load/store units: 8-bit/16-bit/32-bit/64-bit load/store, 32-bit scalar arithmetic
- Dual 64-bit vector units: 16-bit/32 bit/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

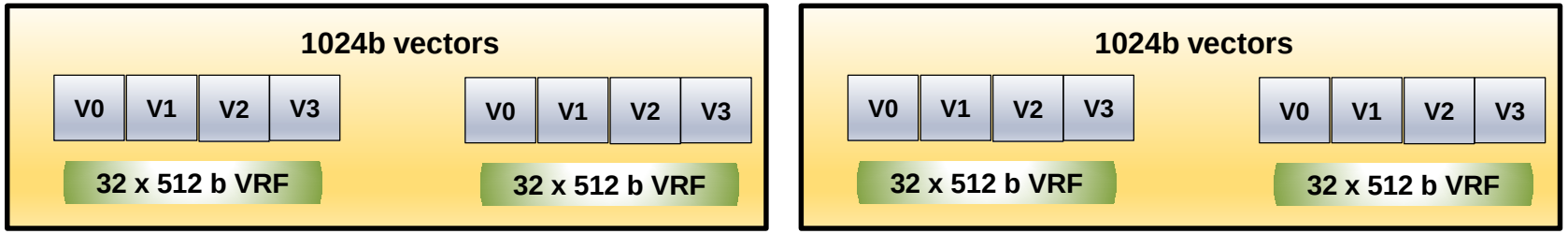


HVX Use Cases – Choose Vector Width

- Vectors are configured as 512-bit[1] (64-byte)
  - Four vector contexts available; one for each hardware thread
  - All hardware threads perform HVX tasks to maintain full vector processing capability



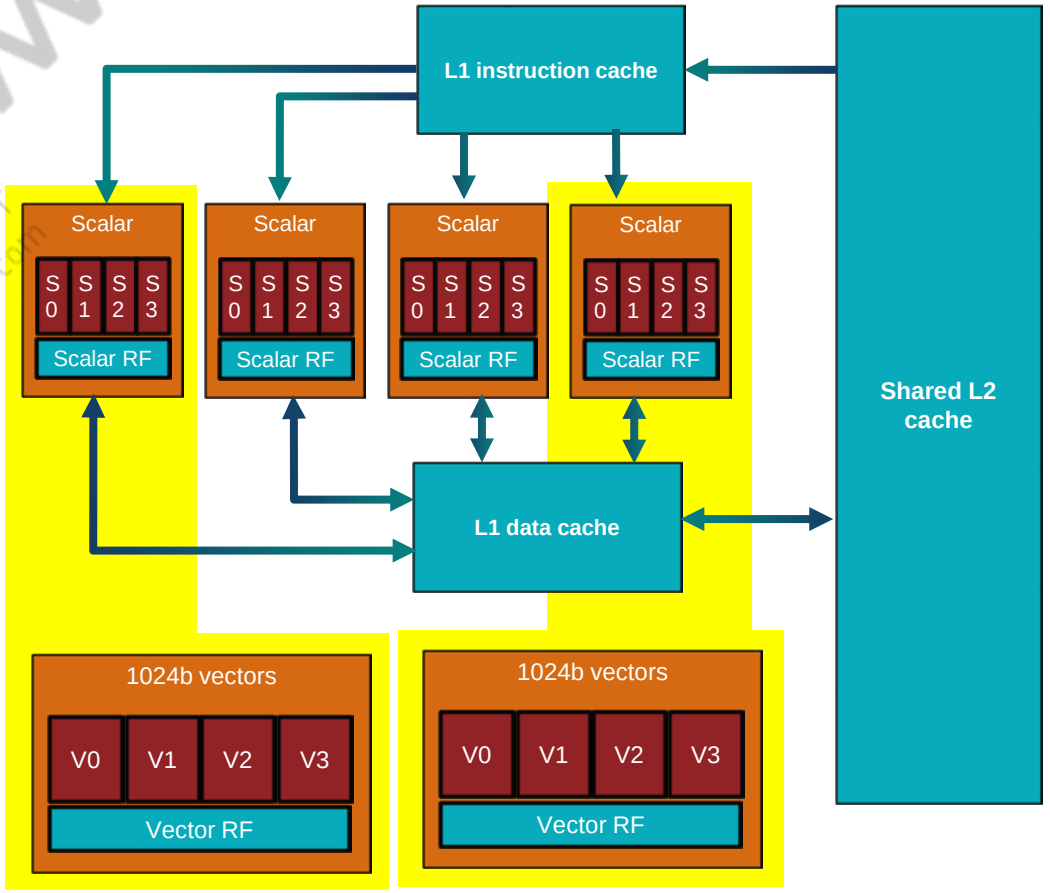
- Vectors are also configured as 1024-bit (128-byte)
  - Two vector contexts are available
  - Two threads can keep vector resources busy
  - Not all algorithms scale well to 128-byte vectors



[1] The 512-bit mode or vector width will not be supported in future chipsets, so algorithms with 1024-bit vector width are to be developed for forward compatibility.

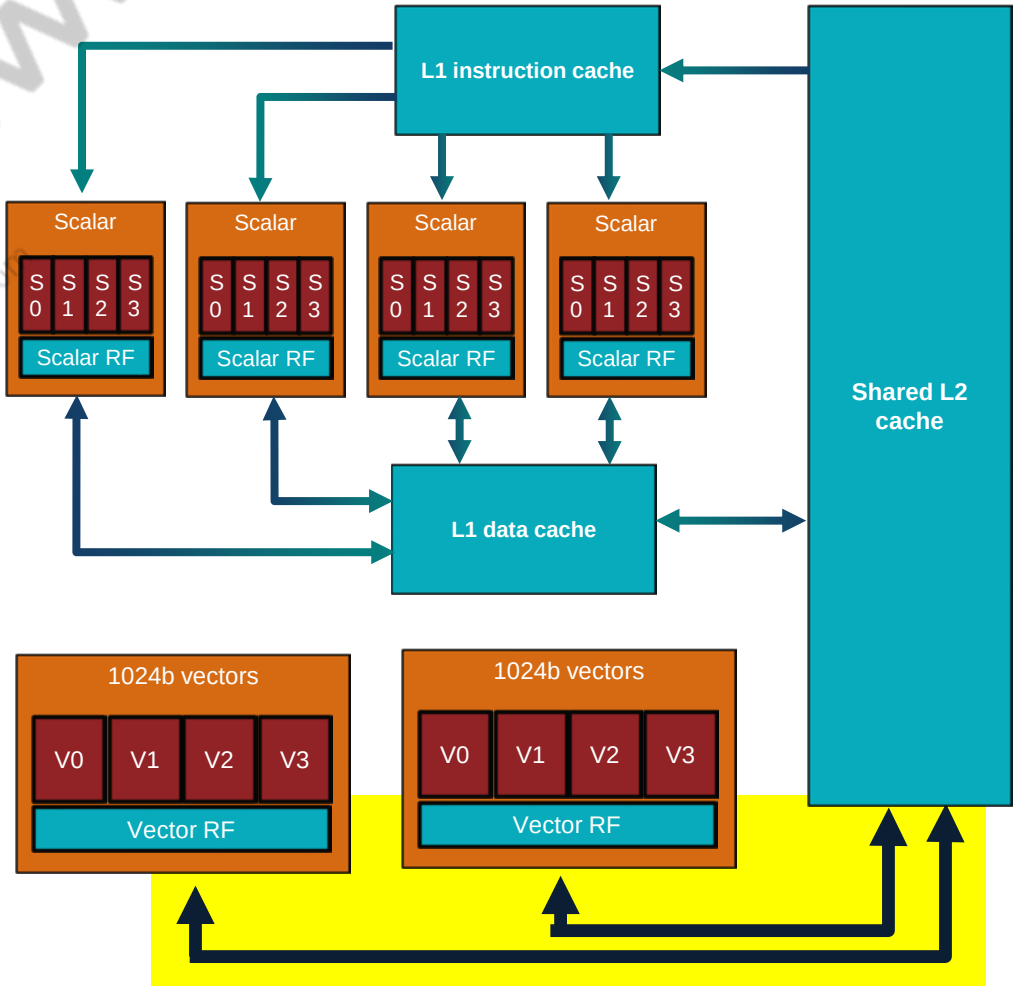
HVX Architecture – Threading Model

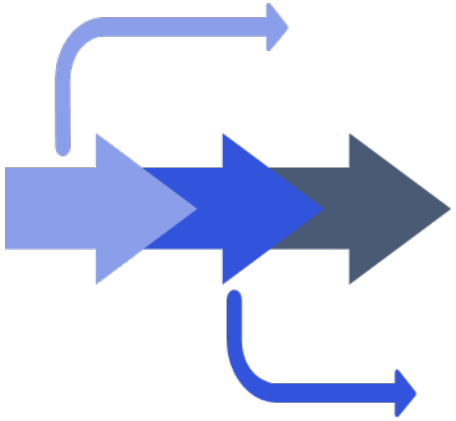
- Four parallel scalar threads, each with four-way VLIW and shared L1/L2
  - 600 MHz peak performance per thread
  - 2.4 GHz total peak scalar performance 1024-bit mode
  - Two HVX contexts, controllable by any two scalar threads
  - Other two threads perform scalar work in parallel
    - 600 MHz peak performance per thread
    - 1200 MHz total peak vector performance



HVX Architecture – Memory

- Vector units support various load/store instructions
  - Unaligned
  - Per-byte conditional
- 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/L2 is hardware coherent
- Streaming prefetch from DDR to L2

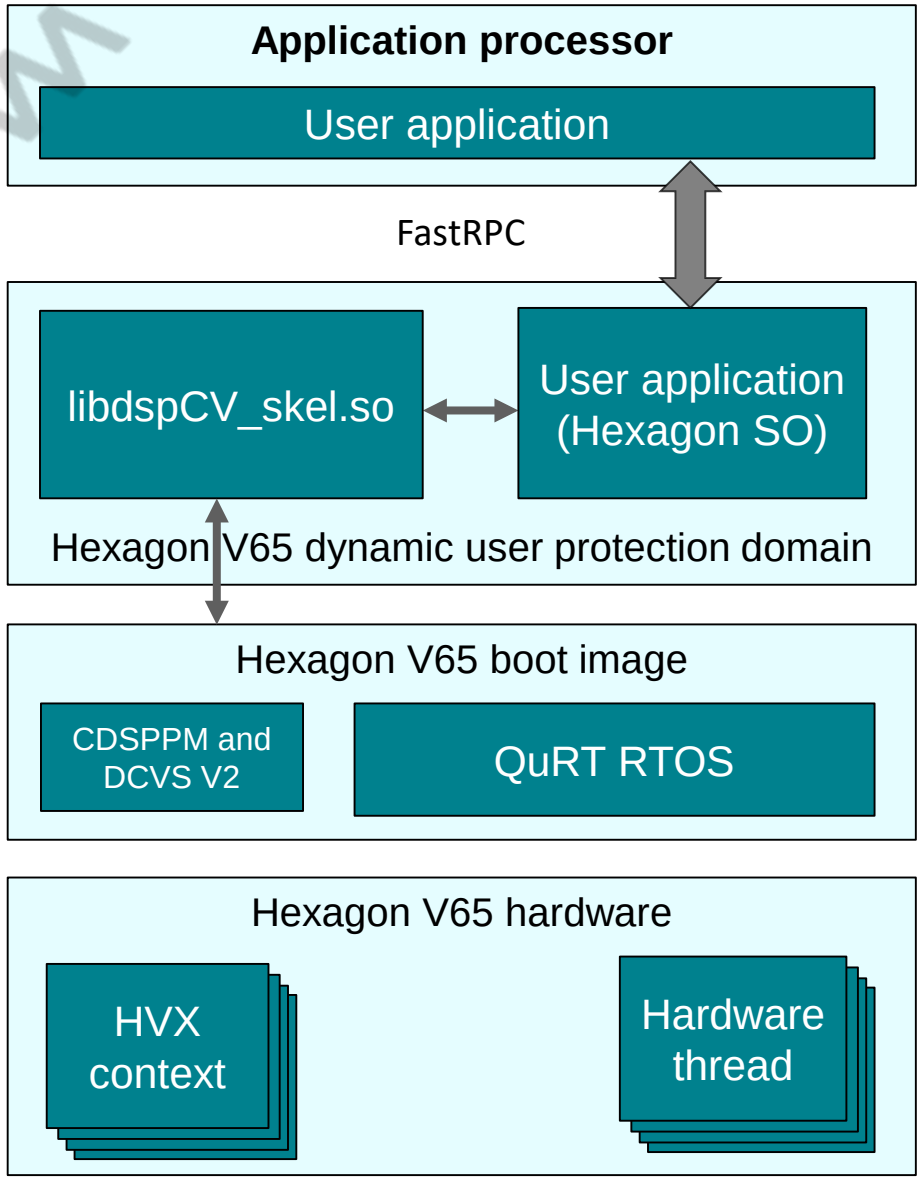




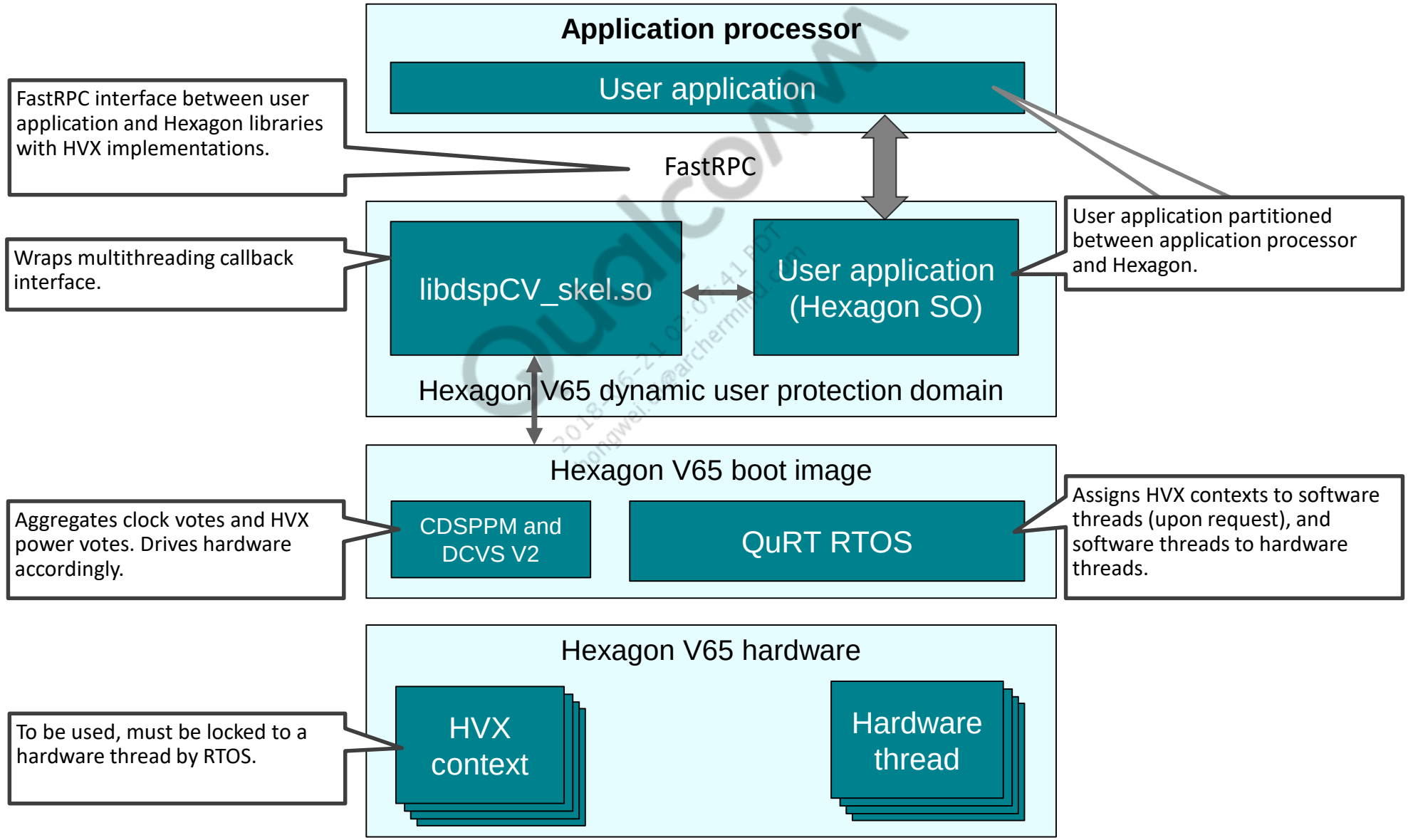
cDSP Software Architecture

cDSP Software Architecture

- Basic compute software architecture is 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 is automatically locked to software thread when that thread issues HVX instructions
    - HVX register set is saved and restored by QuRT software during context switch
  - libdspCV_skel.so provides only multithreading support directly called from DSP side
  - DCVS v2 APIs are provided for clock and power management

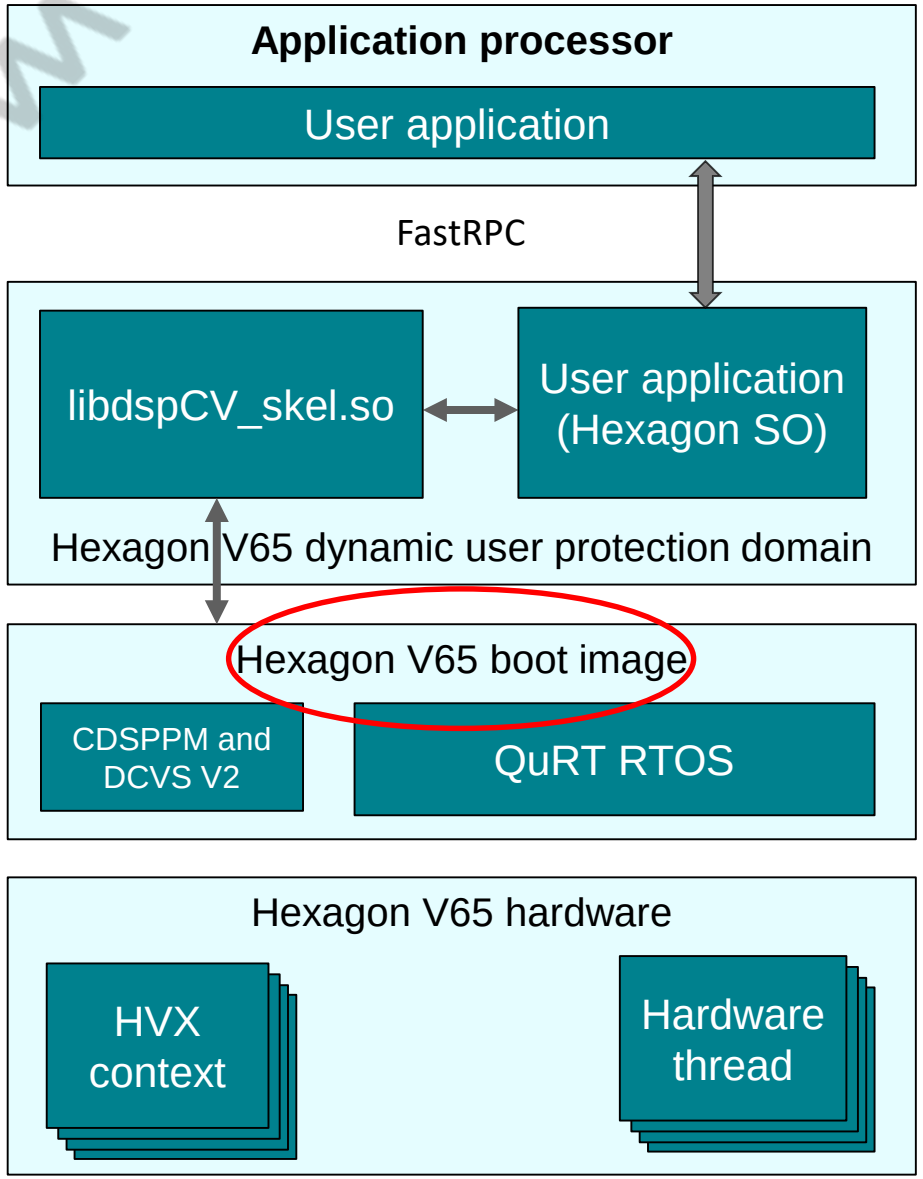


cDSP Software Architecture (cont.)



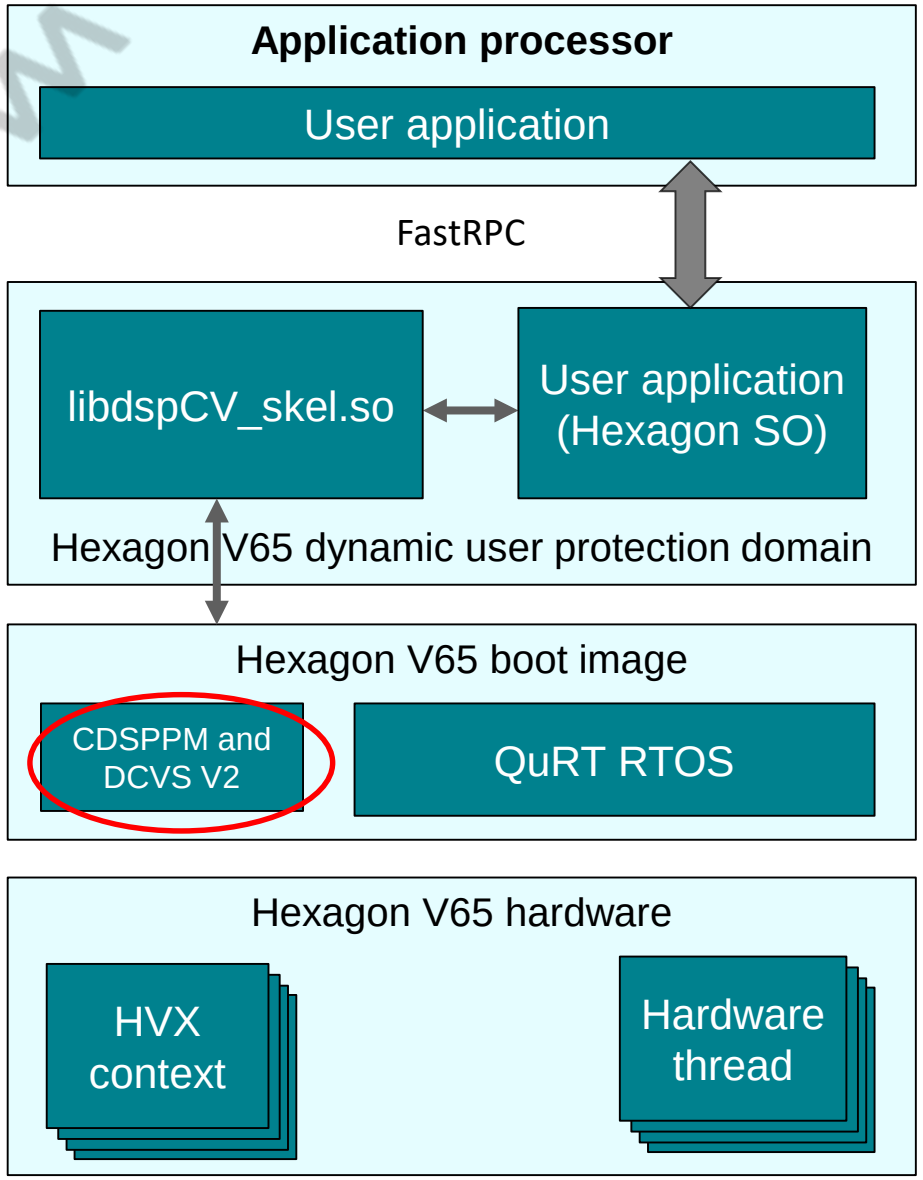
cDSP Software Architecture – QuRT RTOS

- 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 canceled)
  - Locks/unlocks HVX context to a requesting software thread, with specified 512-bit or 1024-bit mode
    - All active HVX contexts must be in same mode at a given time
    - If two threads request locks for two different modes, one is blocked until first thread unlocks
  - Queries for HVX target capabilities



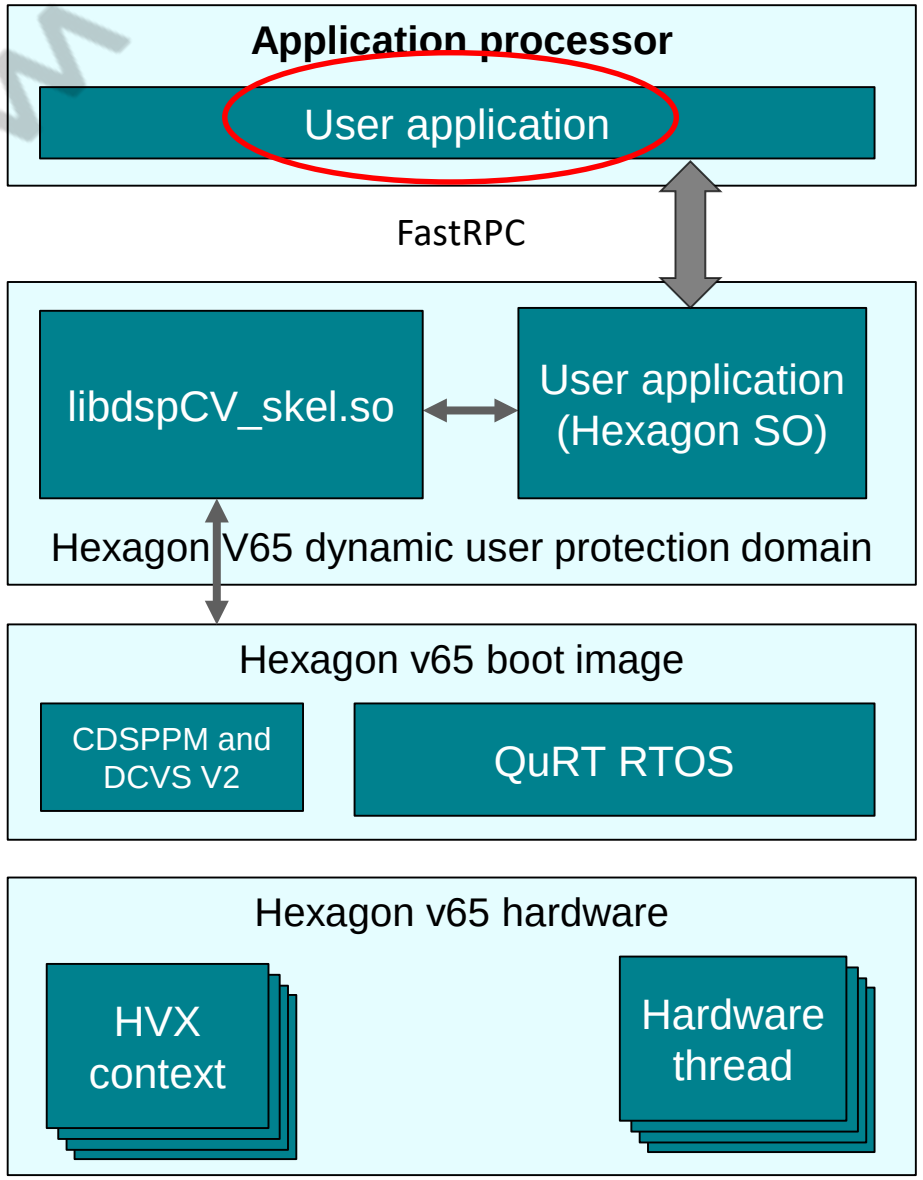
cDSP Software Architecture – cDSP Power Management (cDSPPM) and DCVS

- Aggregates client votes for Hexagon core clock and
  bus clock to DDR
- Aggregates client votes to power on/off HVX units
- Monitors load in real time, and adjusts clocks accordingly



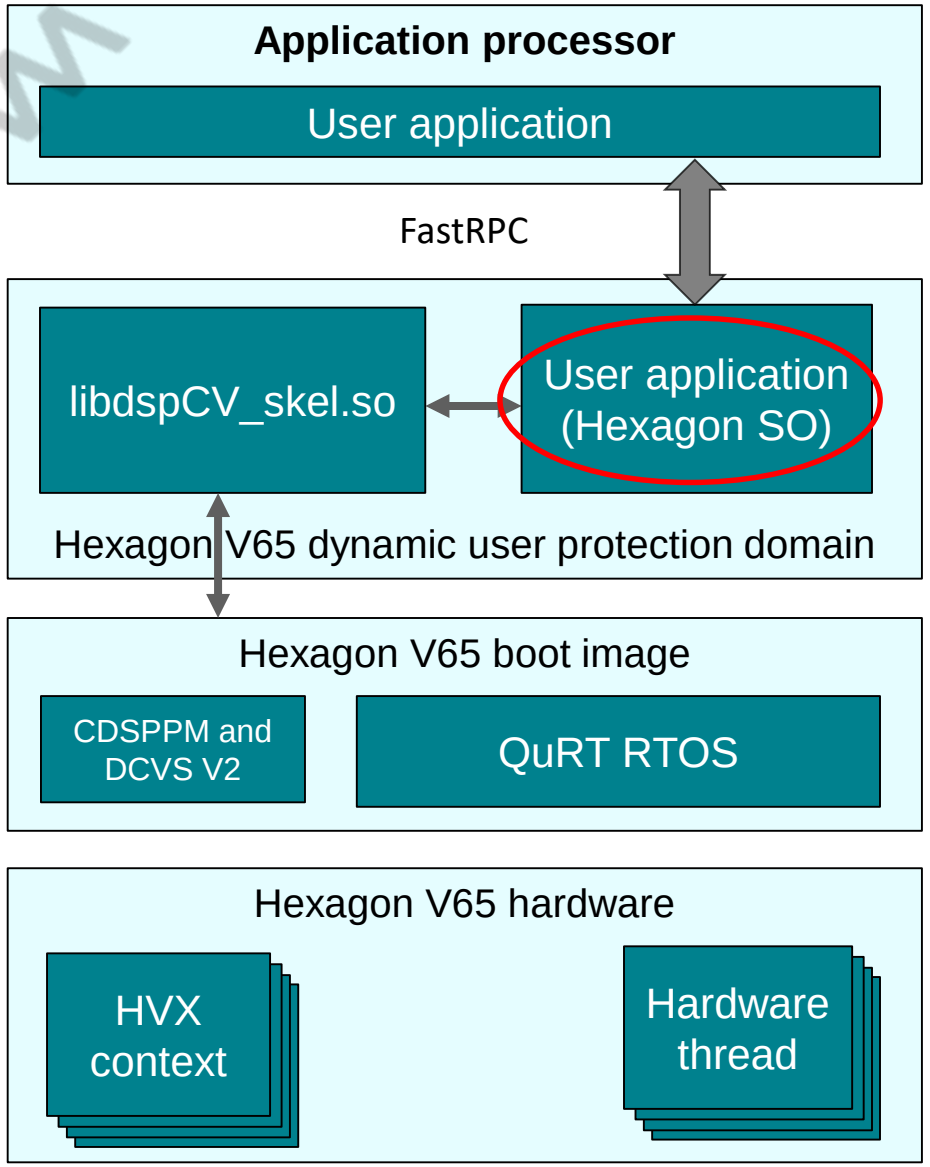
cDSP Software Architecture – User Application

- 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



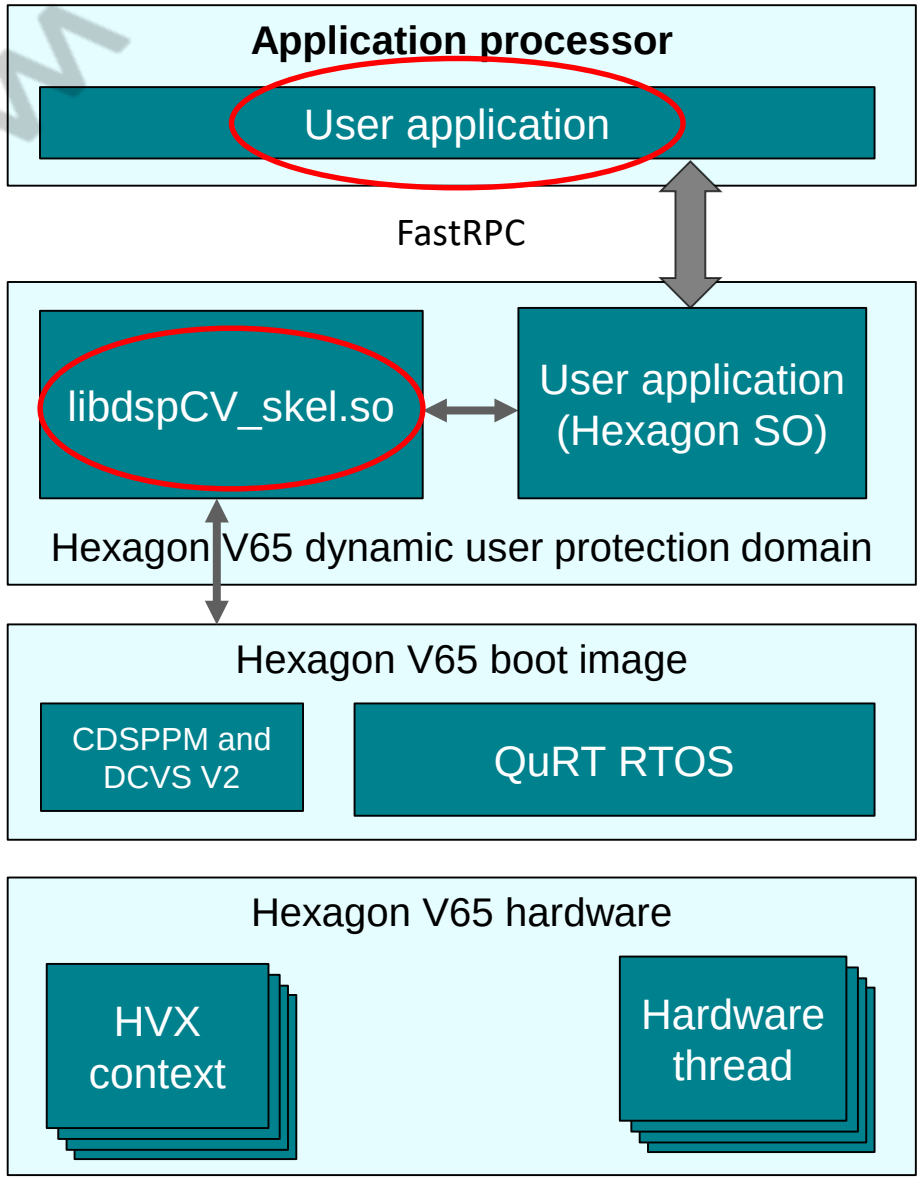
cDSP Software Architecture – User Application (cont.)

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



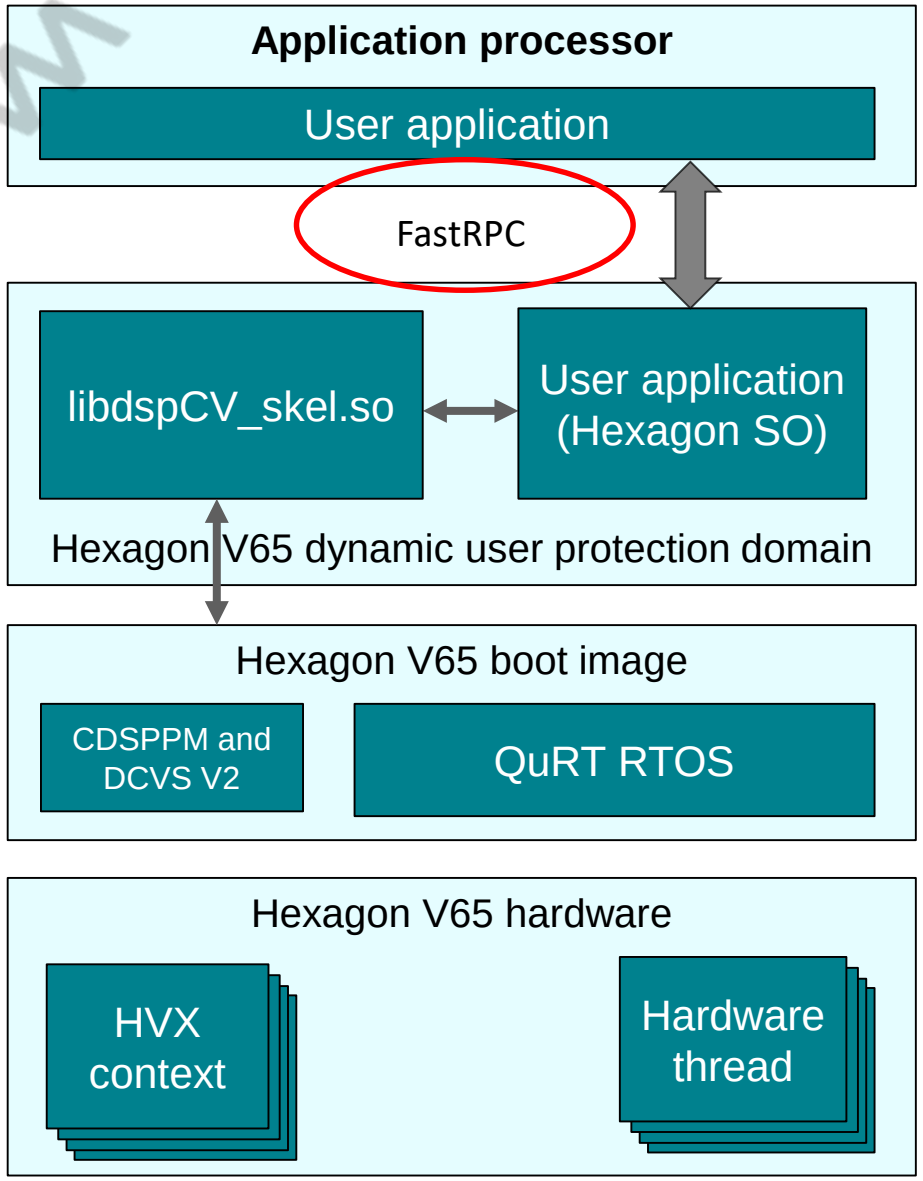
cDSP Software Architecture – libdspCV_skel.so

- libdspCV_skel.so library is Hexagon SDK deliverable
- Used by FastCV applications to implement multithreading
- Wraps required APIs from Hexagon boot image
  - Supplies worker threads and callback interface for user application multithreading



cDSP Software Architecture – FastRPC

- Provides User mode and Kernel mode drivers to enable the user application to call functions in Hexagon SO file
- Upon first RPC request from user application, FastRPC performs following steps:
  - Creates user process on cDSP that services this and subsequent calls from the call process
  - Marshals function arguments back and forth for each RPC call
  - Tears down cDSP process when application process dies
- Cache I/O coherency to reduce RPC latency



cDSP Power Management

- 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 to be guaranteed throughout voting use case lifecycle
  - CDSPPM 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
  - HAP_power APIs provide access for setting clock and bus votes together with other DCVS parameters

cDSP Power Management – DCVS v2 APIs

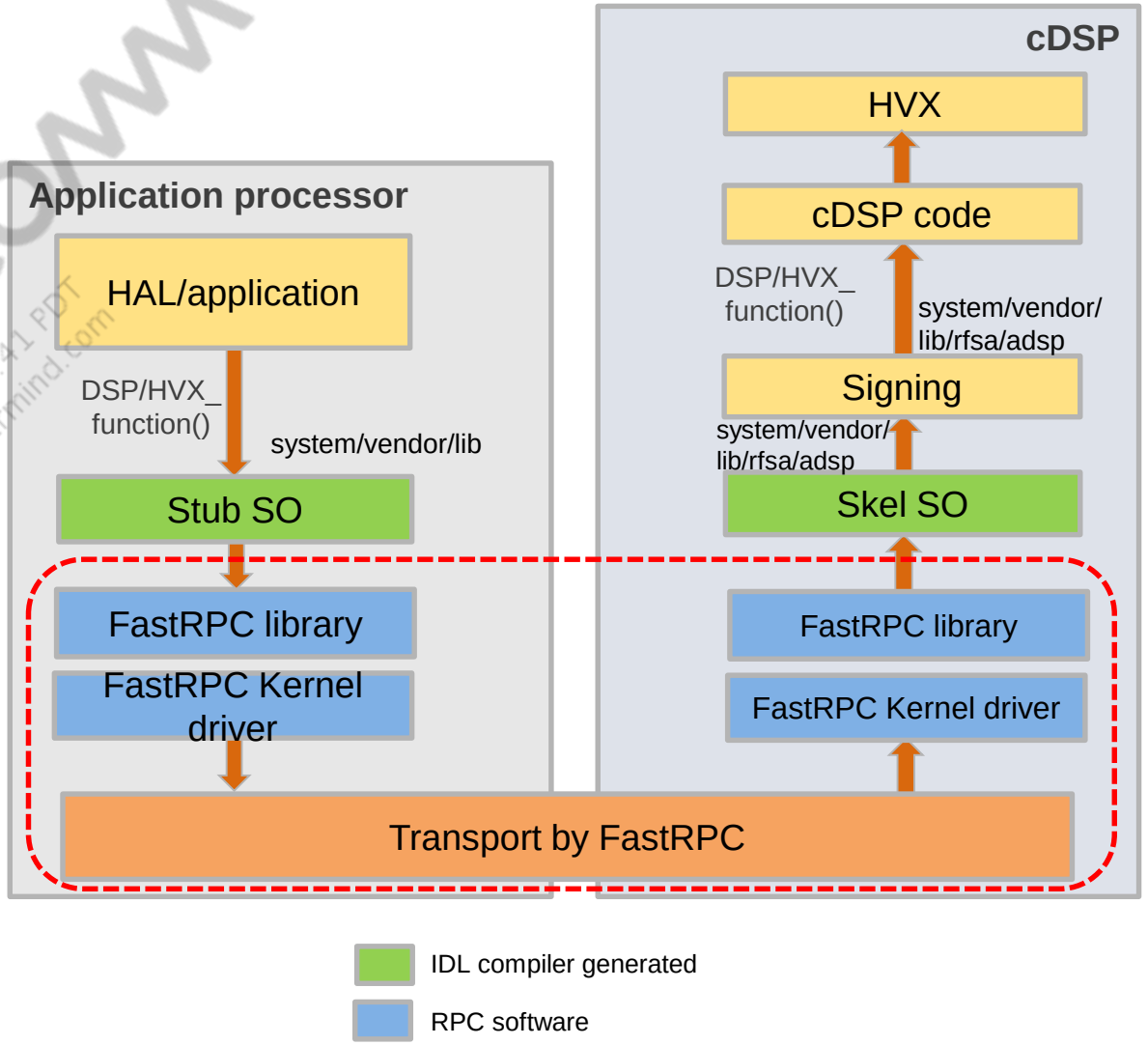
- Use cDSP DCVS v2 to set cDSP clocks
- Following DCVS v2 configurable parameters are used with HAP_power_set API:
  - DCVS enable/disable
  - DCVS options to instruct DCVS algorithm to use predefined set of thresholds and operation logic, based on the selected option
  - Set sleep latency vote in microseconds
  - DCVS parameters to set upper and lower DCVS thresholds, and vote for core and bus clocks using voltage corner

cDSP Power Management – DCVS v2 APIs (cont.)

- For DCVS v2 usage, see Hexagon SDK documentation:
  `<Hexagon_SDK>\<Hexagon SDK Version>\docs\Hap_power_set_dcvs_2.html`
- SDK example:
  `<Hexagon_SDK>\<Hexagon SDK Version>`
  `\examples\compute\benchmark\src_dsp\benchmark_imp.c`
- Function: `benchmark_setClocks()`

Dynamic Loading Data Flow

- Dynamic loading module feature is available in SDM670/SDM710
- Used for both camera streaming and memory-to-memory processing
- Dynamic loading use is not on cDSP; it is on ION heap on HLOS side next to SMMU



Note: IDL compiler is distributed with Hexagon SDK.

cDSP Software Architecture – Cache I/O Coherency

- I/O coherency is one-way feature
  - Only cDSP can snoop into CPU cache and not vice versa
  - CPU does not have to flush and invalidate its cache, while cDSP must flush and invalidate its cache
- Without I/O coherency:
  - CPU and DSP must flush and invalidate their respective caches to maintain cache coherency for each FastRPC invocation
  - Results in extra 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 Software Architecture – Cache I/O Coherency (cont.)

- cDSP snoops into CPU cache for cached buffer loads and stores before accessing main memory
- If cache miss occurs:
  - cDSP goes to main memory (DDR)
  - For larger buffers, cDSP goes to CPU cache and then to main memory, which results in increased latency to DDR for
    IO-coherent buffers
    - Caused by snooping
    - Adds to latency on algorithm side
- I/O coherency is useful for buffers that are allocated as cached by CPU
  - Does not provide any benefit for uncached buffers allocated by CPU
- I/O coherency is enabled through FastRPC framework and can be enabled or 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

cDSP Software Architecture – Disable Cache I/O Coherency

- Clients can disable I/O coherency for ION buffer by registering buffer as noncoherent buffer.
  - Use remote_register_buf_attr() API for registering buffer as noncoherent buffer.

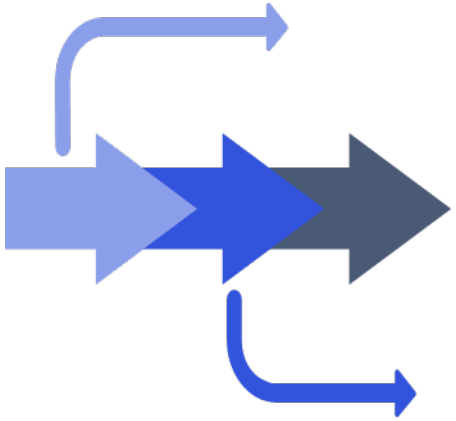
```
void remote_register_buf_attr(void* buf, int size, int fd, int attrs);
buf - virtual address of the buffer
size - size to of the buffer
fd  - the file descriptor, callers can use -1 to deregister.
attr - map buffer as coherent or non-coherent
 #define FASTRPC_ATTR_NON_COHERENT (2)
```

  - Another option is to modify rpcmem_alloc() to pass flag to map buffer as noncoherent

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

Recommendations

- Shorter workloads
  - Stay on CPU, or combine multiple calls into one call
- Longer workloads
  - FastRPC performance is dominated by CPU wakeup/interrupt latency
  - Disabling CPU power collapse greatly improves performance
- Quality of FastRPC service is influenced by power vs. latency tradeoffs
  - Performance lock APIs can set minimum CPU frequencies



cDSP Use Cases

HVX Customization Areas – Plan of Record (PoR) Features

Area	PoR feature	Customization
Camera	Camera streaming framework, JPEG rotation	Ability to implement custom Bayer statistics for auto-focus/ auto-exposure and other such examples using streaming framework.
CV (memory-to-memory)	FastCV HVX-optimized APIs	Detailed list is available in the Hexagon SDK documentation about FastCV tool.
Video postprocessing (memory to memory)	FRC	Frame rate conversion
Snapdragon Computer Vision Engine[1] (SCVE)	• Touch-to-track • Ghost-free panoramic stitching • Image segmentation • Text recognition	SCVE Android APK available for feature evaluation.
Machine Learning	Snapdragon Neural Processing Engine (SNPE), nn_lib	• SNPE SDK available through https://developer.qualcomm.com/ or CreatePoint. nn_lib is available at: https://source.codeaurora.org/quic/hexagon_nn/nnlib • Tensor flow integration at: https://github.com/tensorflow/tensorflow/tree/master/tensorflow/contrib/hvx

[1] Available for an additional license fee.

FastCV Overview

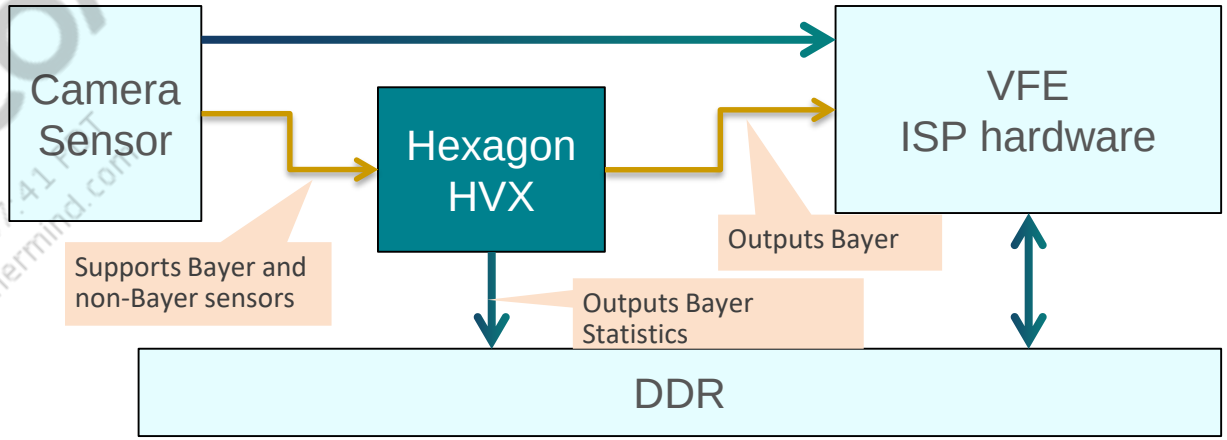
- API and library that enables mobile devices to run CV applications efficiently
- Enables acceleration of CV applications through hardware
- Analogous to OpenGL ES in rendering domain, and is clean modular library

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 *Snapdragon Computer Vision Overview* (80-NL315-9)

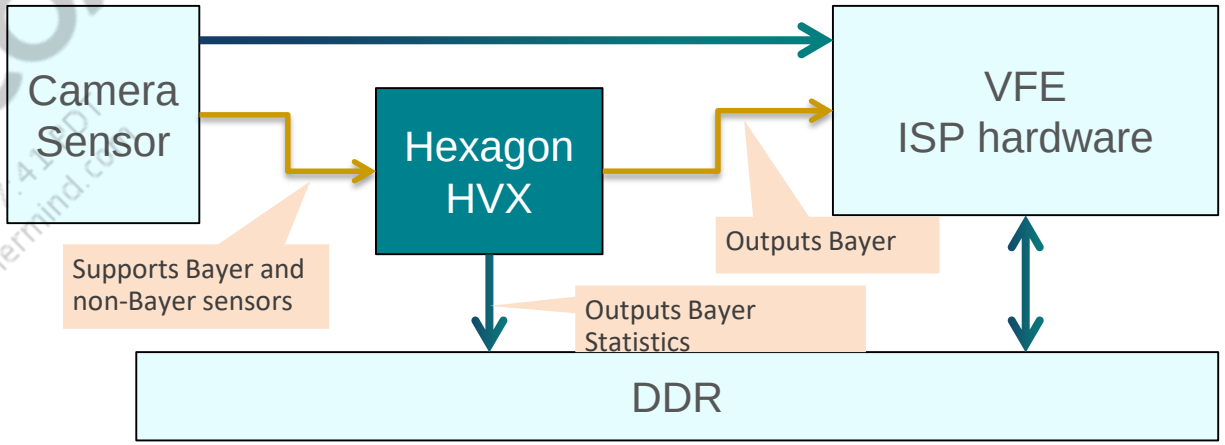
Camera Streaming Overview

- Use cases of special bus that enables access:
  - 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 is inserted between camera sensor and VFE (QTI ISP)

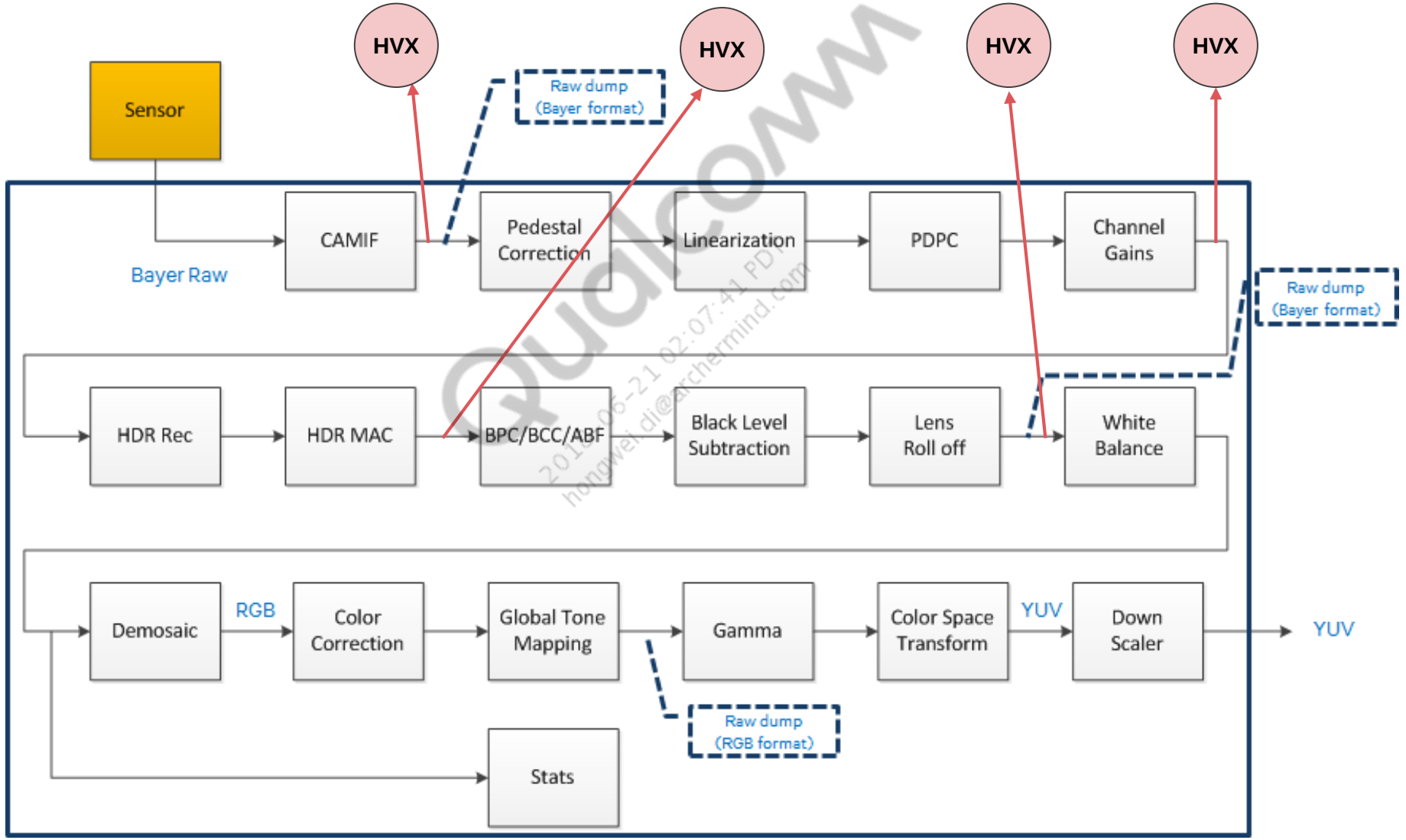


Camera Streaming Overview (cont.)

- 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 *examples/camera_streaming* in *Hexagon SDK 3.3*



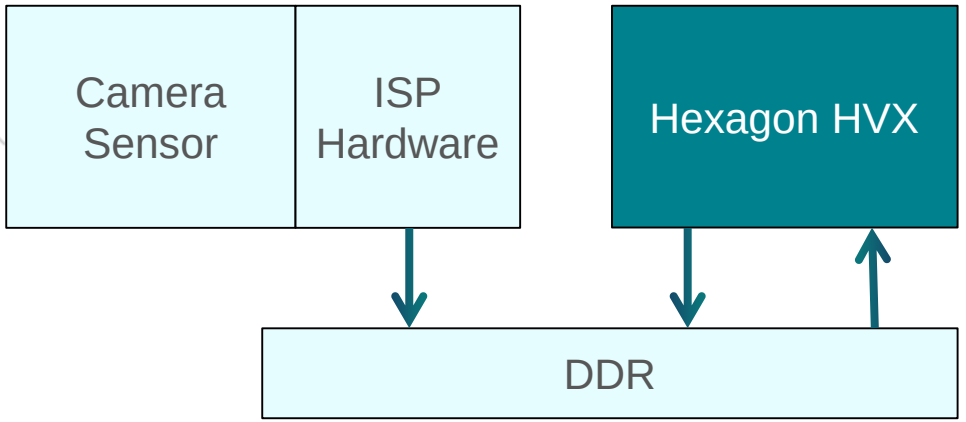
Camera Streaming – HVX Tap Points



Note: Details on how to enable HVX tap points will be updated in later revisions of this document.

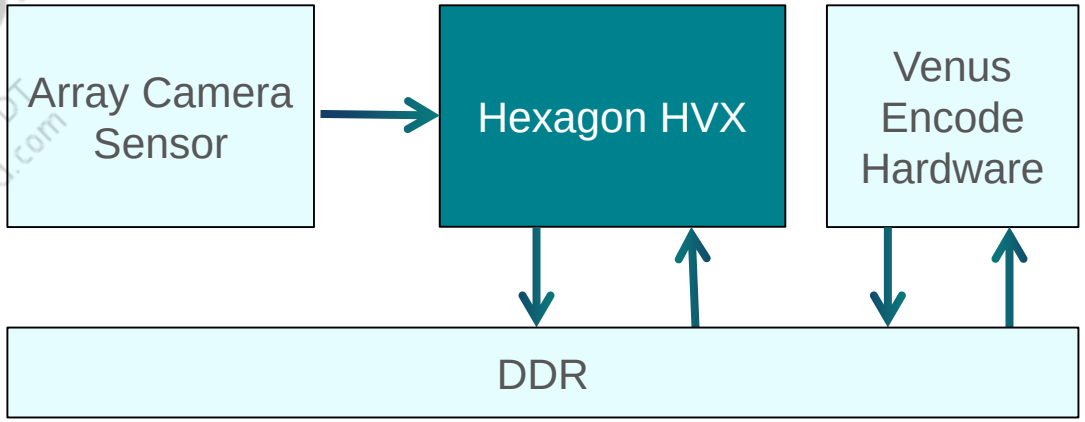
Camera Memory-to-Memory Pre/Postprocessing

- Use case:
  - Memory to memory before ISP or after ISP
- Example application:
  - Wavelet-based denoiser
  - CV
- For details, see *examples/camera_streaming* in *Hexagon SDK 3.3*



Computational Camera

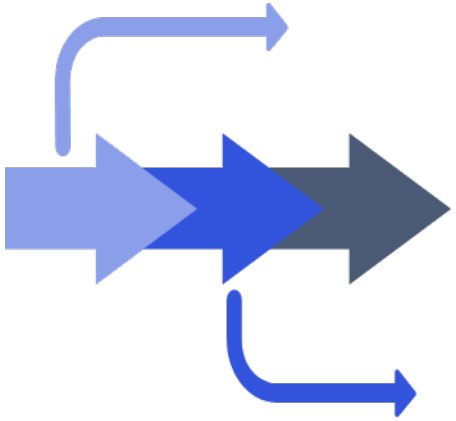
- Next generation use cases such as Augmented Reality and Virtual Reality
  - Computational photography using array cameras
- Nontraditional camera cannot use ISP hardware
  - All ISP functions are performed in the software



Customization Workflow

HVX customization stages

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 is 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 is used for mass production signing of module.



HVX Development Environment

HVX Development Environment and Resources

- Development tools
  - SDK 3.3 is used for SDM670/SDM710 HVX development.
  - Hexagon SDK 3.3 Variants
    - Hexagon600_SDK.WIN.3.3 files
    - Hexagon600_SDK.LNX.3.3 files
  - Hexagon tools
    - Starting from version 8.x.08, C++14 compiler is supported
    - Starting from version 8.x.09, Halide compiler support is added for automatic generation of HVX executable code
    - For SDK 3.3, default tool chain supported is 8.1.03
  - All tools are available on CreatePoint
- Resources
  - *Hexagon V65 HVX Programmer Reference Manual* (80-N2040-41)
  - *Hexagon V65 Programmer Reference Manual* (80-N2040-39)

Note: Use the latest tools from CreatePoint because the Hexagon Development tools version might be updated during the lifecycle of the chipset.

Resources – Hexagon Training Videos

- Following training videos are available on CreatePoint:

Course	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)

Resources – Hexagon Training Videos (cont.)

Course	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)

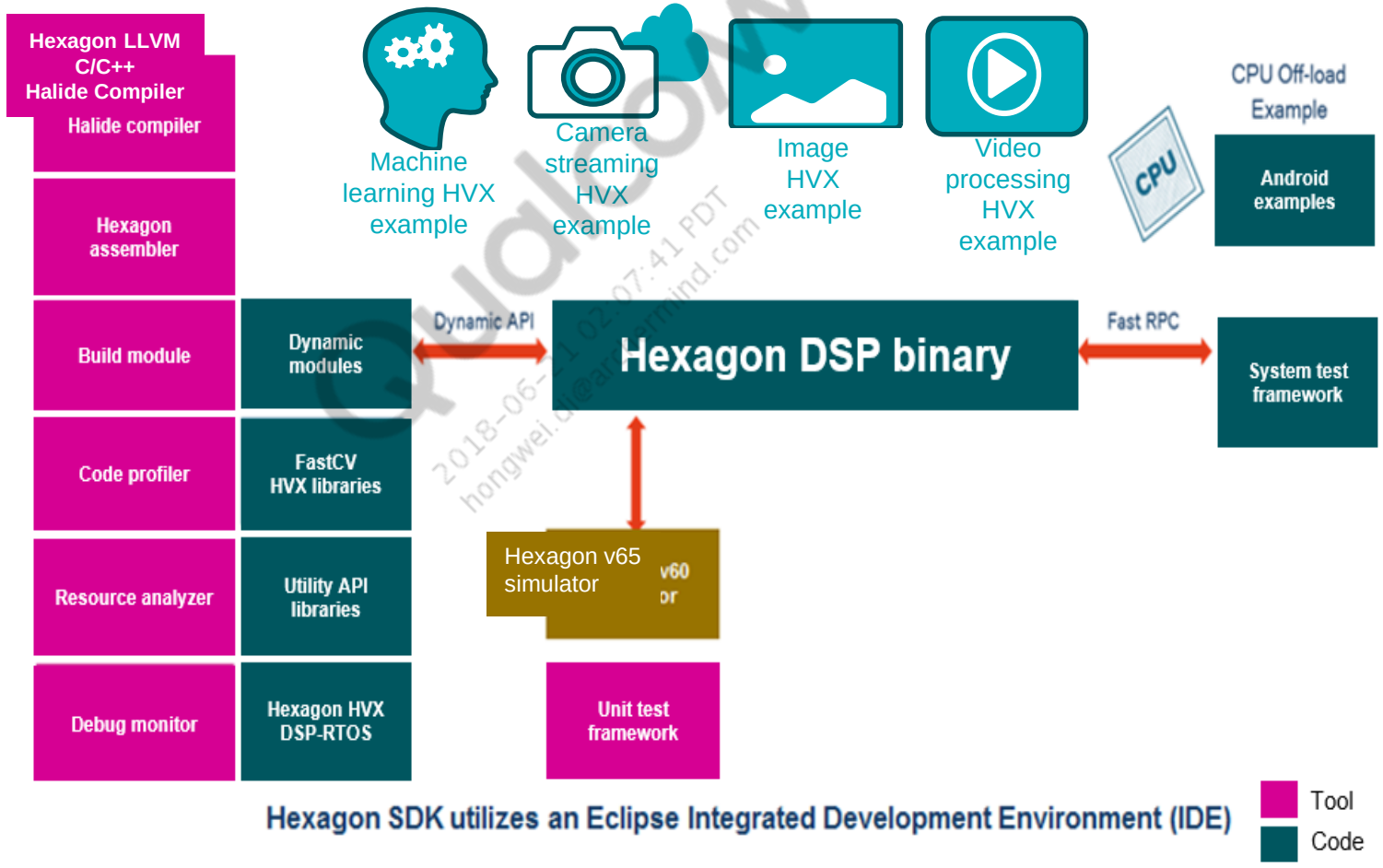
Resources – HVX Training Videos

Course	DCN
HVX Programming: Hexagon Core Review	VD80-P4366-1
HVX Programming: Architecture Review	• VD80-P4366-2A (Part 1) • VD80-P4366-2B (Part 2)
HVX Programming: HVX Tools	VD80-P4366-3
HVX Programming: Simple Example	• VD80-P4366-4A (Part 1) • VD80-P4366-4B (Part 2)
HVX Programming: Unusual features	• VD80-P4366-5A (Part 1) • VD80-P4366-5B (Part 2)
HVX Programming: Exploring HVX Using 3 x 3 convolution	• VD80-P4366-6A (Part 1) • VD80-P4366-6B (Part 2)
HVX Programming: Micro-architecture	• VD80-P4366-7A (Part 1) • VD80-P4366-7B (Part 2)
HVX Programming: Assembly programming	• VD80-P4366-8A (Part 1) • VD80-P4366-8B (Part 2)
HVX Programming: Assembly 3 x 3 convolution example	VD80-P4366-9

Resources – HVX Training Videos (cont.)

Course	DCN
HVX Programming: Advanced algorithms	• VD80-P4366-10A (Part 1) • VD80-P4366-10B (Part 2) • VD80-P4366-10C (Part 3)
HVX Programming: Advanced instructions – Data	• VD80-P4366-11A (Part 1) • VD80-P4366-11B (Part 2) • VD80-P4366-11C (Part 3)
HVX Programming: Advanced instructions – Arithmetic	• VD80-P4366-12A (Part 1) • VD80-P4366-12B (Part 2) • VD80-P4366-12C (Part 3)
HVX Programming: OS concerns	VD80-P4366-13
HVX Programming: HVX Scope	VD80-P4366-14

Hexagon SDK – SDK for HVX



Hexagon SDK – SDK for HVX (cont.)

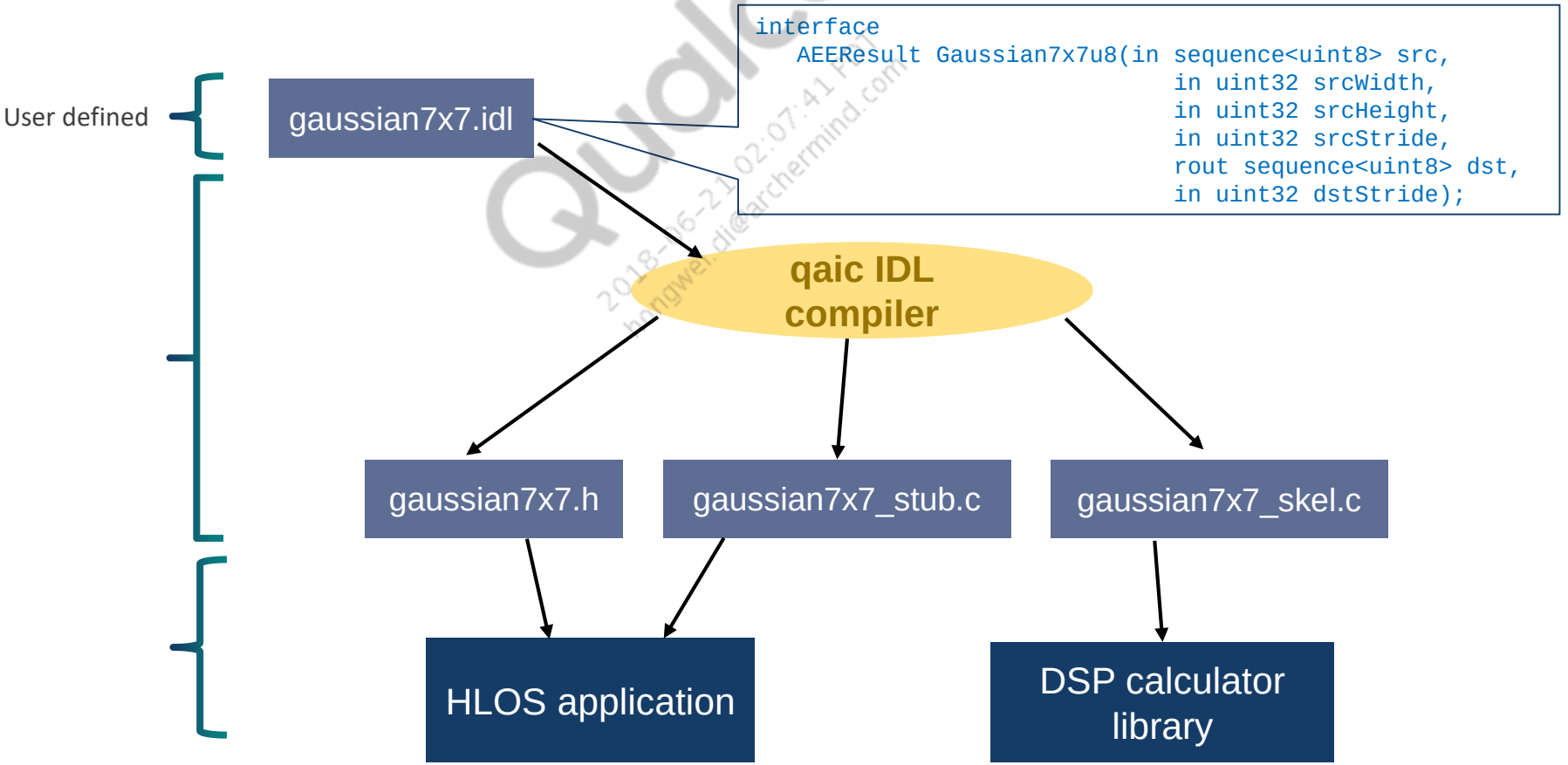
- Use SDK to build code for PC simulation and stand-alone execution on target
- SDK target verification involves dynamically linked shared objects that are loaded on cDSP and ARM processor
- After verifying on target, integrate the ARM functionality in HAL
- Documentation: `C:\Qualcomm\Hexagon_SDK\<Hexagon SDK Version>\docs`
- Example code: `C:\Qualcomm\Hexagon_SDK\<Hexagon SDK Version>\examples`
- FastCV, camera (image processing), machine learning, and common examples are available in their respective folders
  - To ensure that signing is handled correctly, start with calculator example in `C:\Qualcomm\Hexagon_SDK\<Hexagon SDK Version>\examples\common\calculator`
  - Then verify required examples in camera streaming, common, compute, and fastCV

Hexagon SDK – SDK for HVX (cont.)

- For example, Gaussian 7 × 7 is in `C:\Qualcomm\Hexagon_SDK\<Hexagon SDK Version>\examples\compute\gaussian7x7`
  - Test executable and stub.so (ARM implementation): `C:\Qualcomm\Hexagon_SDK\<Hexagon SDK Version>\examples\compute\gaussian7x7\android_Release\ship`
  - Skel.so (cDSP implementation): `C:\Qualcomm\Hexagon_SDK\<Hexagon SDK Version>\examples\compute\gaussian7x7\hexagon_Debug_dynamic\ship`
  - IDL: `C:\Qualcomm\Hexagon_SDK\<Hexagon SDK Version>\examples\compute\gaussian7x7\inc`
  - Stub-generated code: `C:\Qualcomm\Hexagon_SDK\<Hexagon SDK Version>\examples\compute\gaussian7x7\android_Release`
  - DSP-HVX assembly code implementation: `C:\Qualcomm\Hexagon_SDK\<Hexagon SDK Version>\examples\compute \gaussian7x7\asm_src`
  - Test app, Skel, and cDSP implementation in C and C Intrinsics: `C:\Qualcomm\Hexagon_SDK\<Hexagon SDK Version>\examples\compute\gaussian7x7\src`
  - Build makefile configuration: `C:\Qualcomm\Hexagon_SDK\<Hexagon SDK Version>\examples\compute\gaussian7x7\glue`

Hexagon SDK – IDL Diagram

- qaic compiles IDL files into headers, stubs, and skels
- For more information on the IDL compiler, see Hexagon SDK documentation, `C:\Qualcomm\Hexagon_SDK\<Hexagon SDK Version>\docs\Tools_IDL Compiler.html`.



Hexagon SDK – Simulator

1. Set up environment: `C:\Qualcomm\Hexagon_SDK\<Hexagon SDK Version>\setup_sdk_env.cmd`
2. Compile and simulate:

   ```
   make tree_clean V=hexagon_Release_toolv80_v60 VERBOSE=1
   make tree V=hexagon_Release_toolv80_v60 VERBOSE=1
   ```

   Note: For supported build variants, see the Hexagon SDK:

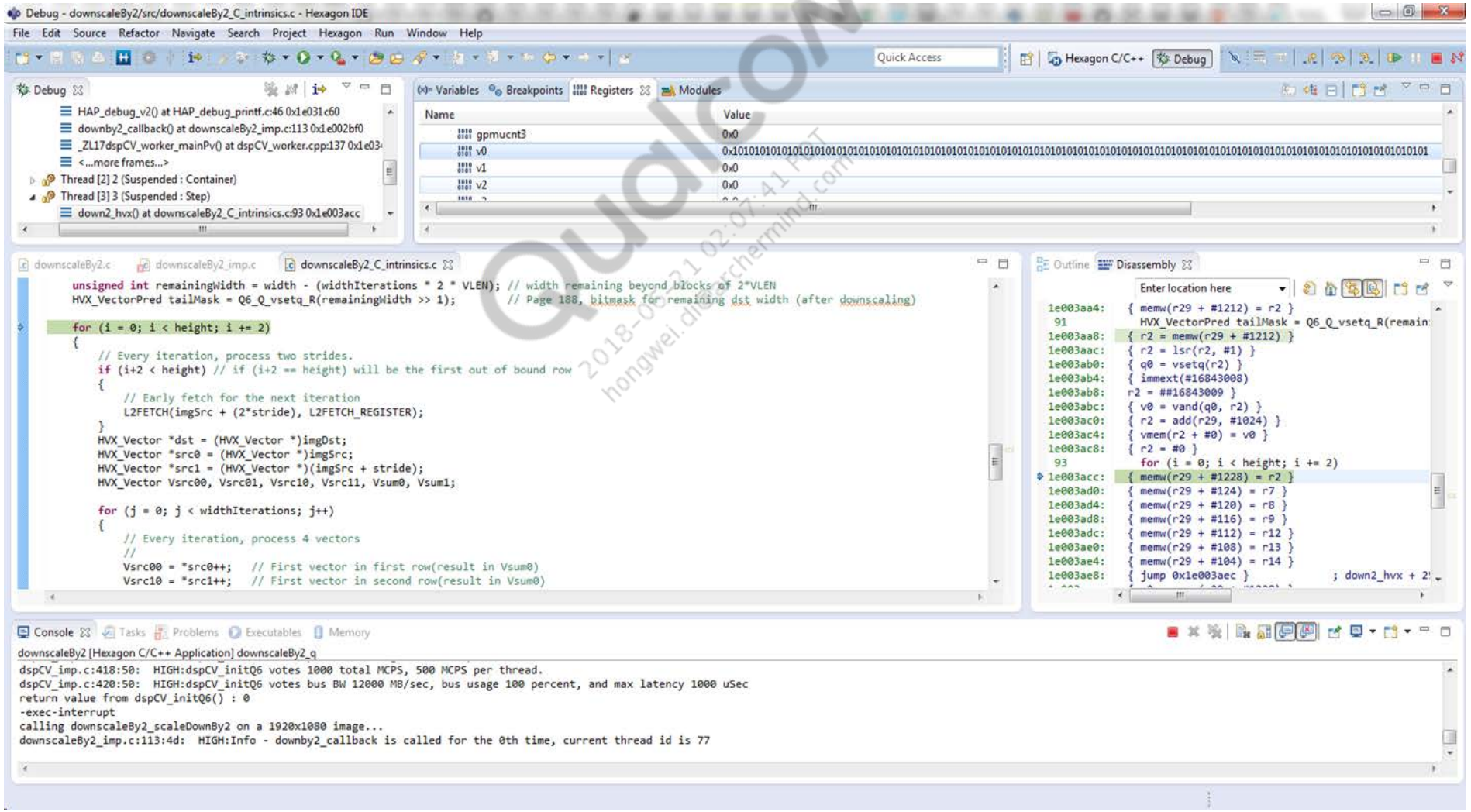
   Hexagon SDK version >Environments>Command Line>Variants section.

   `C:\Qualcomm\Hexagon_SDK\<Hexagon SDK Version>\docs\Environments_CommandLine.html`
3. LLDB debugging:

   ```
   C:\…\examples\compute\gaussian7x7\hexagon-lldb hexagon_Release_toolv80_v60\gaussian7x7_q -mv65 --dsp_clock 700
   ```

Hexagon SDK – IDE

- The Hexagon IDE is bundled within the Hexagon SDK to assist with the development of HVX standalone applications



Hexagon SDK – Test Target

1. Set up the target testing environment:
   C:\Qualcomm\<Hexagon SDK version>\3.3\setup_sdk_env.cmd
2. Compile.
   a) Standalone and stub:

```
make tree V=android_Release  VERBOSE=1
 adb push c:\…\android_Release\ship\gaussian7x7 /data
 adb shell chmod 777 /data/gaussian7x7
```

   b) Skel

```
make tree V=hexagon_Release_toolv80_v60 VERBOSE=1
adb push c:\...\hexagon_Release_toolv80_v60\ship\libgaussian7x7_skel.so /system/lib/rfsa/adsp
```

3. Execute.
   For details, see `adb shell /data/gaussian7x7` in Hexagon SDK documentation

Note: Before target testing, sign the device for the development phase as explained on slide 72.

Hexagon SDK – libcdsprpc.so Integration

- Hexagon 3.1 SDK examples require makefile change for SDM670/SDM710
- SDM670/SDM710 has dedicated cDSP for compute, libcdsprpc.so must be integrated instead of libadsprpc.so to call FastRPC framework
- To build the application Android binary for target, make following change in `android.min` file:

```
From gaussian7x7_DLLS += libadsprpc
To gaussian7x7_DLLS += libcdsprpc
```

Halide – Image-Processing Language

- Halide is domain-specific programming language used to write 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 on Halide, 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 these pipelines include:
  - Definition of pipeline stages
  - Optimization
  - Parallelism – Vectorization, multithreading
  - 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, OpenVX may not compose into efficient pipelines due to lack of computation and storage reuse
  - Library calls add an optimization barrier
- Addresses the challenges in the time and effort taken to tune algorithms
  - Assembly is time intensive and requires deep knowledge of low-level processor details
  - Compiler intrinsics require a significant learning curve and code may not be reusable across different processors

Halide and HVX

- Halide supports many targets, including HVX:
  - HVX
  - X86/SSE
  - ARM v7/NEON
  - CUDA
  - OpenCL
  - Native Client
- Halide allows algorithm to be dispatched on HVX with a code directive
- Halide is open source project
- Hexagon/HVX support is added to Halide project
- Halide depends on LLVM toolset to generate Hexagon/HVX binaries
- Halide allows programmers with little knowledge of HVX internals to focus on authoring and tuning image algorithms
  - Allows larger set of programmers to program for HVX
- Eases the offloading of image kernels to HVX; reduces the effort to prototype and gauge profitability of kernel running on HVX
- The Halide language is a derivative of C/C++, which many programmers are familiar with
- Halide on HVX uses the QuIC LLVM compiler for code generation. The QuIC LLVM compiler is tuned extensively for Hexagon and HVX
- Leverages optimizations in Hexagon/HVX LLVM compiler and is able to shield the programmer from low-level optimizations required for performance

Halide Programming

- Halide programs consist of two major components:
  - Algorithm: Specifies what will be computed at a pixel level
  - Schedule: Specifies how the computation should be organized
- Separation is a key to enable:
  - Authoring an algorithm without the complexity of computational organization
  - Tuning and experimenting with the computational organization to achieve high performance
  - Generating highly optimized compiled code guided by the programmer, explicitly specifying high-level optimizations

Halide Programming (cont.)

- For more information on Halide programming and Halide porting examples on HVX see: `<Hexagon SDK Root>\<Hexagon version>\tools\HEXAGON_Tools\<HEXAGON Tools version>\Tools\Halide or <HEXAGON_Tools Root>\<HEXAGON Tools version>\Tools\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

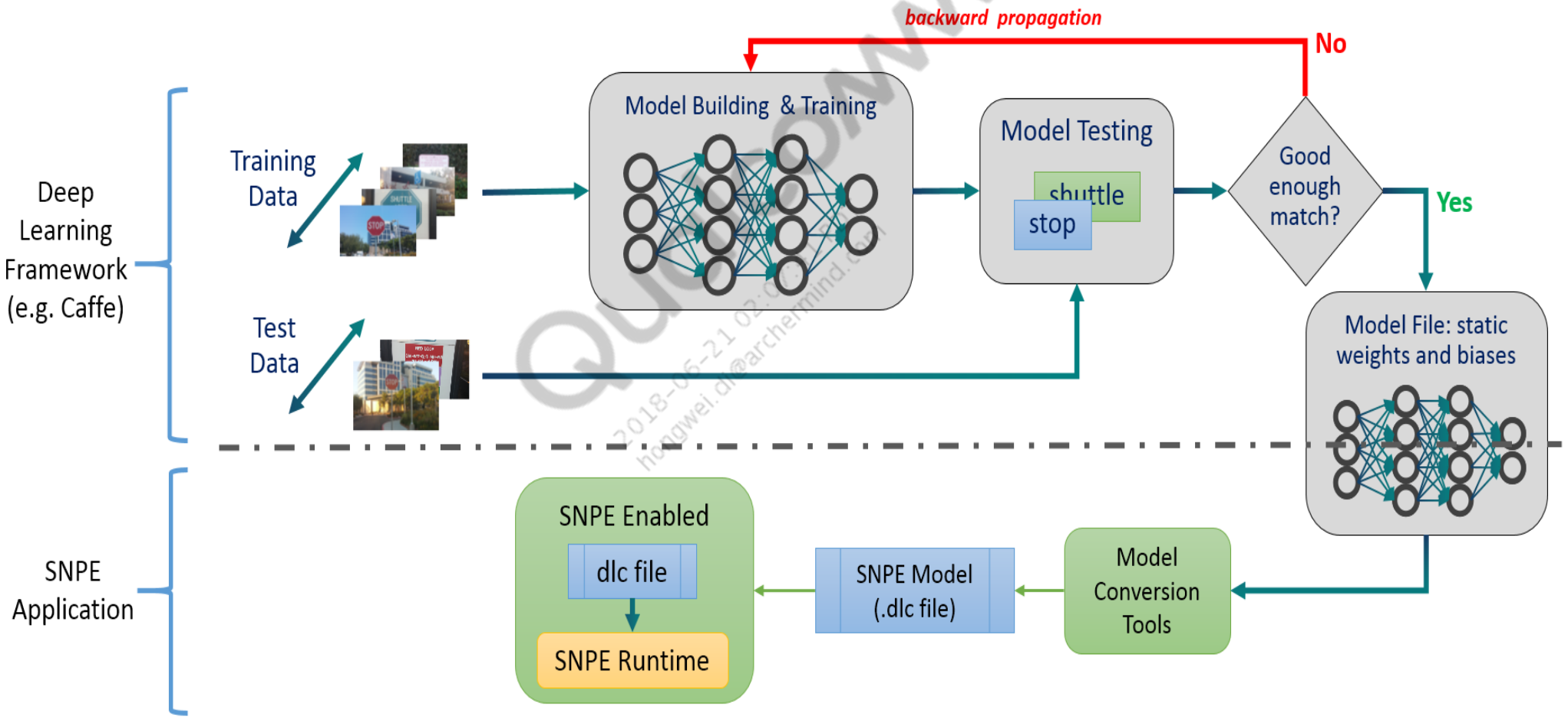
Halide – HVX Benefits

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

Snapdragon Neural Processing Engine (SNPE)

- SNPE is a Snapdragon software accelerated runtime for the execution of deep neural networks. The SNPE features are as follows:
  - Executes an arbitrarily deep neural network
  - Executes the network on the Snapdragon CPU, the Qualcomm® Adreno™ GPU, or the Hexagon DSP. DSP is only supported on Android.
  - Debugs the network execution on x86 Ubuntu Linux
  - Converts Caffe, Caffe2, and TensorFlow models to an SNPE Deep Learning Container (DLC) file
  - Quantizes DLC files to 8-bit fixed point for running on the Hexagon DSP
  - Debugs and analyzes the performance of the network with SNPE tools
  - Integrates a network into applications and other code via C++ or Java

SNPE – Model Workflow



SNPE – Model Workflow (cont.)

- Model training is performed on a deep learning framework (Caffe, Caffe2, and TensorFlow models are supported by SNPE.)
- After training is completed the trained model is converted into a DLC file that is downloaded into the SNPE runtime. This DLC file can then be used to perform forward inference passes using one of the Snapdragons accelerated compute cores.
- The basic SNPE workflow consists of only a few steps:
  - Convert the network model to a DLC file that is loaded by the SNPE.
  - Optionally quantize the DLC file for running on the Hexagon DSP.
  - Prepare input data for the model.
  - Load and execute the model using SNPE runtime.
- For more detailed information on SNPE and setup, download Snapdragon_NPE_SDK.LA.1.0 Installer
- from CreatePoint. Currently SNPE SDK support is limited to Ubuntu, specifically version 14.04.

Note: SNPE support on SDM670/SDM710 chipset will be available by end of November. However, SNPE can be explored on chipsets like SDM660, SDM845.

Development Signing

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

Development Signing – Algorithm Development

1. Set up the environment

```
c:\Qualcomm\Hexagon_SDK\<Hexagon SDK Version>>setup_sdk_env.cmd
Setting up the Hexagon SDK environment locally
Done
```

2. Run testsig.py:

```
c:\Qualcomm\Hexagon_SDK\<Hexagon SDK Version>\tools\scripts>testsig.py
```

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

Development Signing – Internal Testing

- Elfsigner command to sign the 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 pathname of the output folder of the signed output file.
- Usage example:

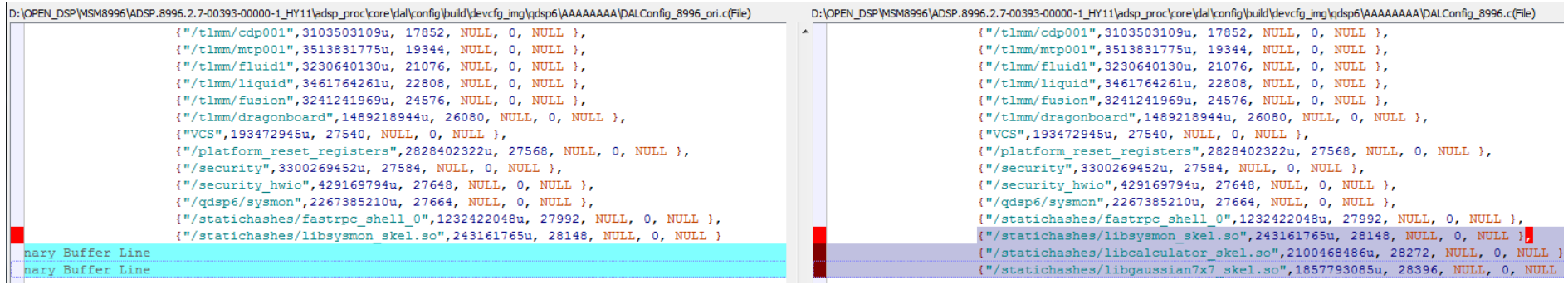
```
python elfsigner.py -i dynmod.so -o signed/ Signs the input file dynmod.so writes the signed
output file to ./signed/dynmod.so
python elfsigner.py -i input/dynmod.so -o signed/ This command signs the input file
./input/dynmod.so and writes the signed output file to ./signed/dynmod.so
```

- Only works with nonproduction devices

Production Signing

- Static hashing method is used for mass production signing of algorithm shared object so that it works only with specific DSP firmware image that is 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 run time to authenticate
- DSP firmware is recompiled when shared object implementation is changed

1. Set the OEM_ROOT path:`>set OEM_ROOT=D:\OPEN_DSP\SDM670\CDSP.._HY11\cdsp_proc\hap\oem`
2. Copy the skel files for hashing to dynamic_modules_hash: `<CDSP..._HY11>\cdsp_proc\hap\oem\build\dynamic_modules_hash`
3. Compile the cDSP build after replacing the SO file: `<CDSP..._HY11>\cdsp_proc\build>python build.py -c sdm670 -o all`
4. Verify that static hashing works properly by confirming that the SO file is added in DALConfig_<CHIPID>.c: `<CDSP..._HY11>\cdsp_proc\core\dal\config\build\ devcfg_img\qdsp6\AAAAAAAA\DALConfig_<CHIPID>.c`



Before compile

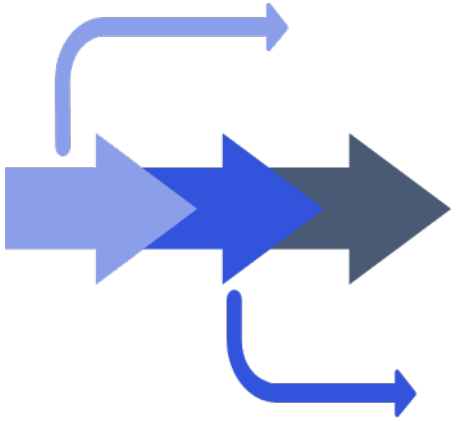
After compile

Production Signing (cont.)

5. Replace the cDSP binary in the target and reboot the target:

```
cd cdsp_proc\obj\qdsp6v5_ReleaseG\670.cdsp.prod\LA\system\etc\firmware
adb root
adb remount
adb shell mount -t vfat -o rw,remount /dev/block/mmcblk0p1 /firmware
adb shell mount -t ext4 -o rw,remount /dev/block/mmcblk0p12 /system
adb shell rm /firmware/image/cdsp.*
adb push . /firmware/image/
adb reboot
```


Note: The cDSP image folder location may be changed in an OEM environment. Check the folder location first.



VTCM

VTCM

- Hexagon V65 adds 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
  - Is only visible to cDSP loads and stores
    - Not connected to UBWC/DMA
  - 256 kB for SDM670/SDM710
- Use cases
  - New HVX scatter-gather instructions can only operate on VTCM memory
    - Typically, HVX-based memcpy is performed between L2 and VTCM before and after scatter-gather.
  - Scratch buffers for camera streaming use cases that exceed 256 KB L2 requirement
  - Scratch/temporary memory for any other algorithms that can benefit from local storage

VTCM – APIs

- Query VTCM size and minimum page size of target

```
void Resmgr_total_VTCM(int* page size, int* num_pages)
```

- Query available VTCM resource

```
void Resmgr_avail_VTCM(int* page_size, int* num_pages)
```

- Request VTCM resource

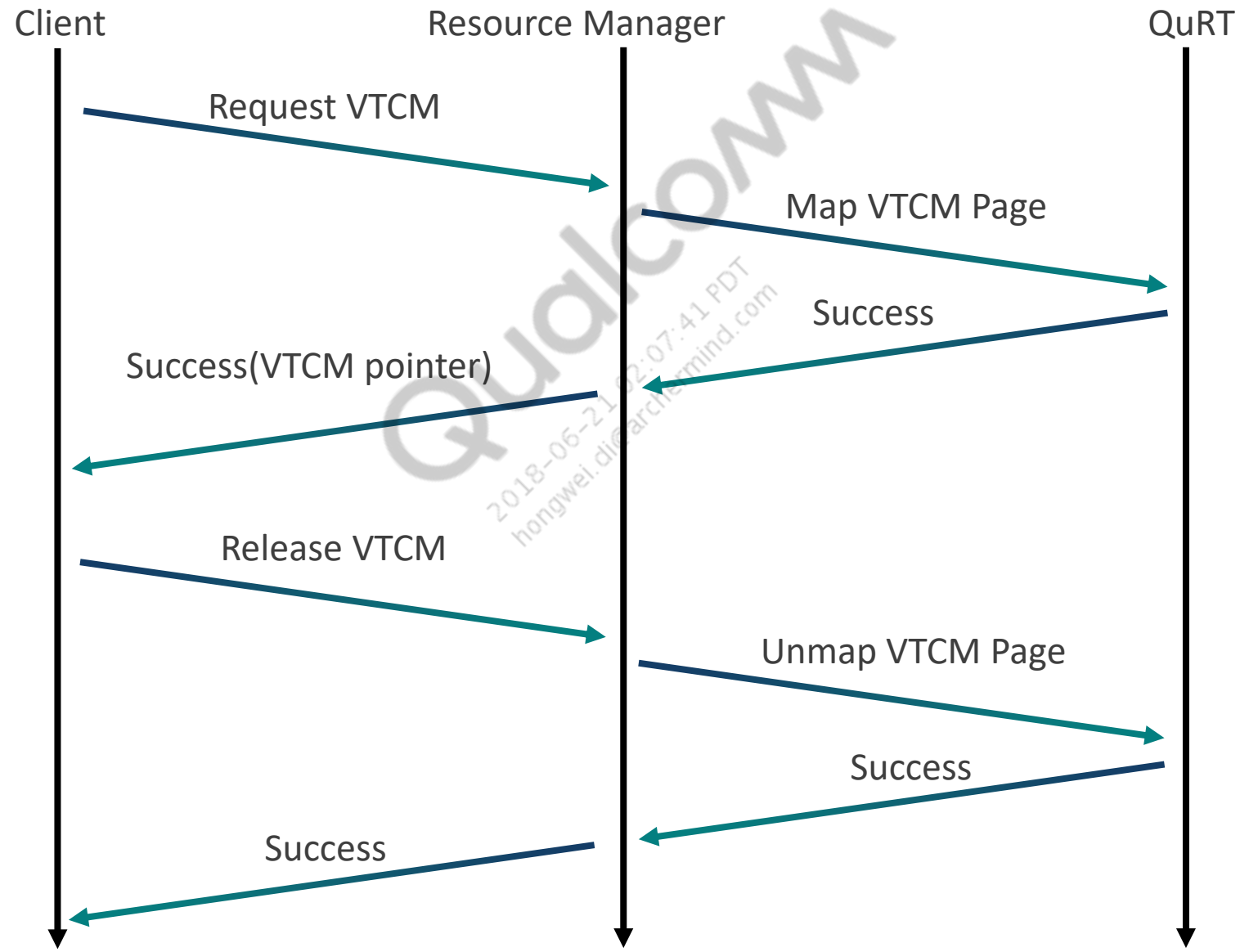
```
void* Resmgr_request_VTCM(int size, int single_page_required_flag)
```

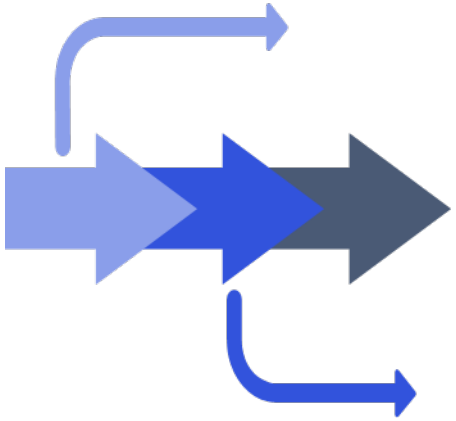
- Release VTCM resource

```
void Resmgr_release_VTCM(void* ptrVTCM)
```

- For more details, see SDK documentation, `C:\Qualcomm\Hexagon_SDK\<Hexagon SDK Version>docs/VTCM Manager.html`

VTCM – Call Flow



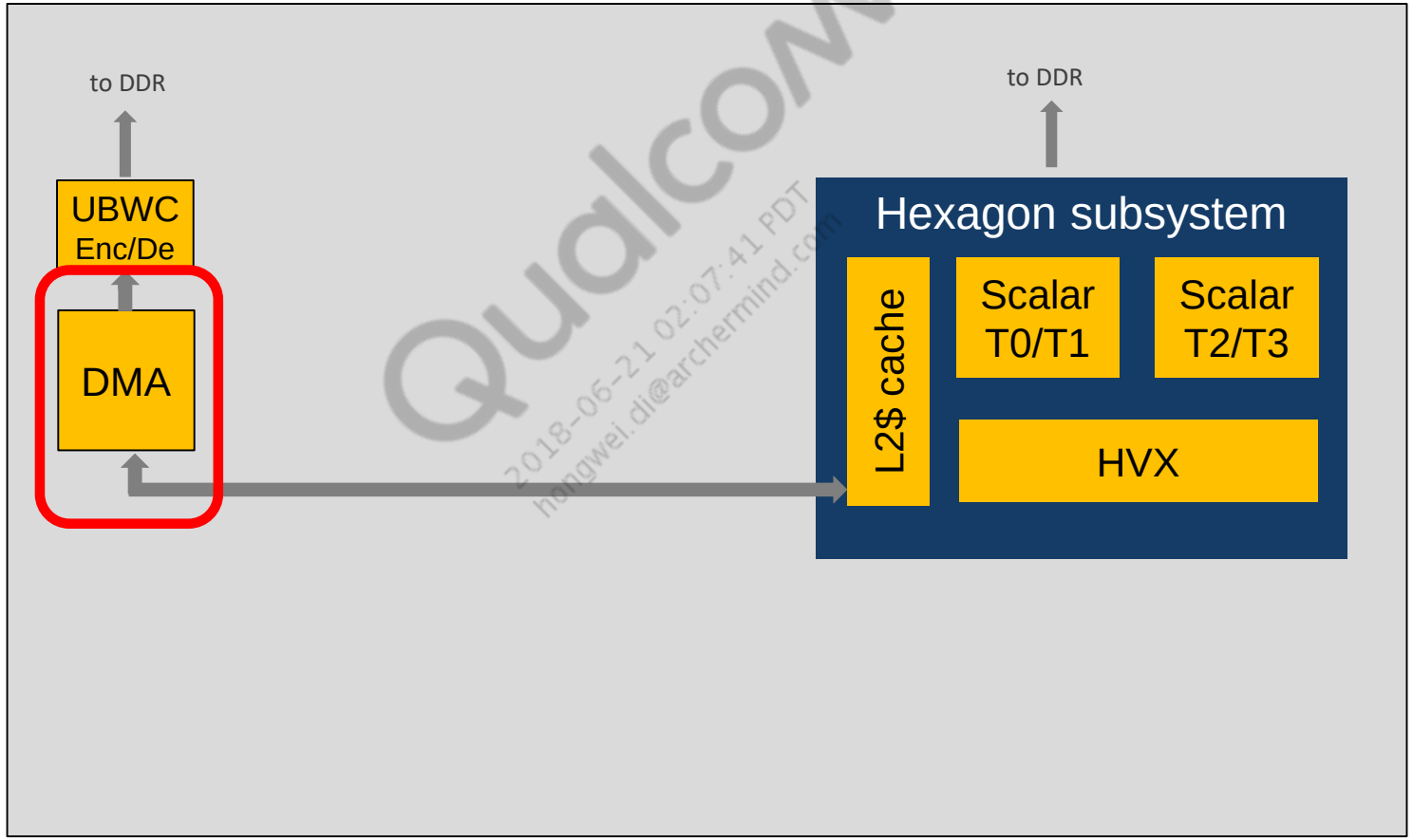


UBWC DMA

DMA Overview

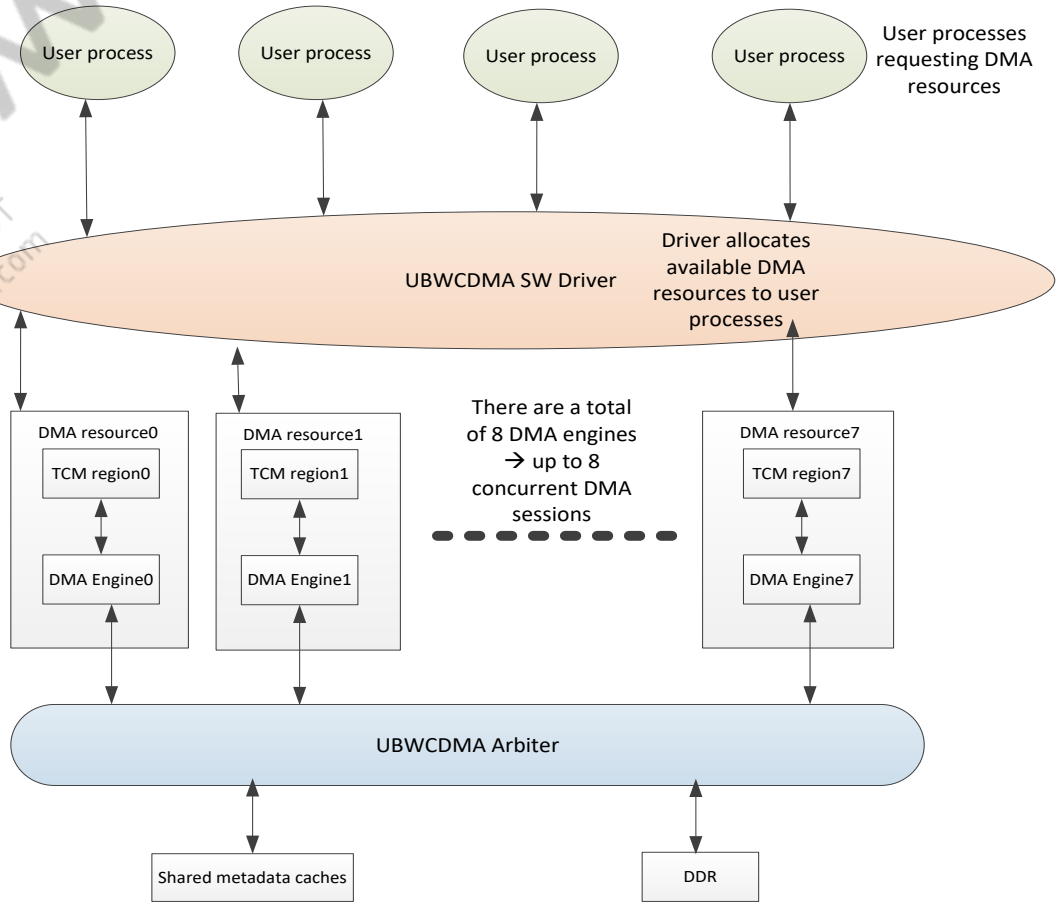
- Provides DMA block to access UBWC frames in DDR and present them to Hexagon DSP as linear frames
  - Allows Hexagon DSP to massage and alter linear frames, and then transfer them back to DDR in UBWC frames
  - Provides DMA transactions for linear frames in DDR
- Supports eight DMA engines
  - Total of eight DMA sessions can run concurrently
- Supports pixel formats
  - NV12, P010, TP10, and NV12-4R (no support for Linear TP10)
- Supports 16-bit pixel padding
  - Converts NV12/NV124R/TP10 into 16-bit values and store them in Hexagon DSP local memory
- Supports Low Power Polling (LPP), interrupt, spin-loop polling

cDSP with DMA – High-Level Block diagram

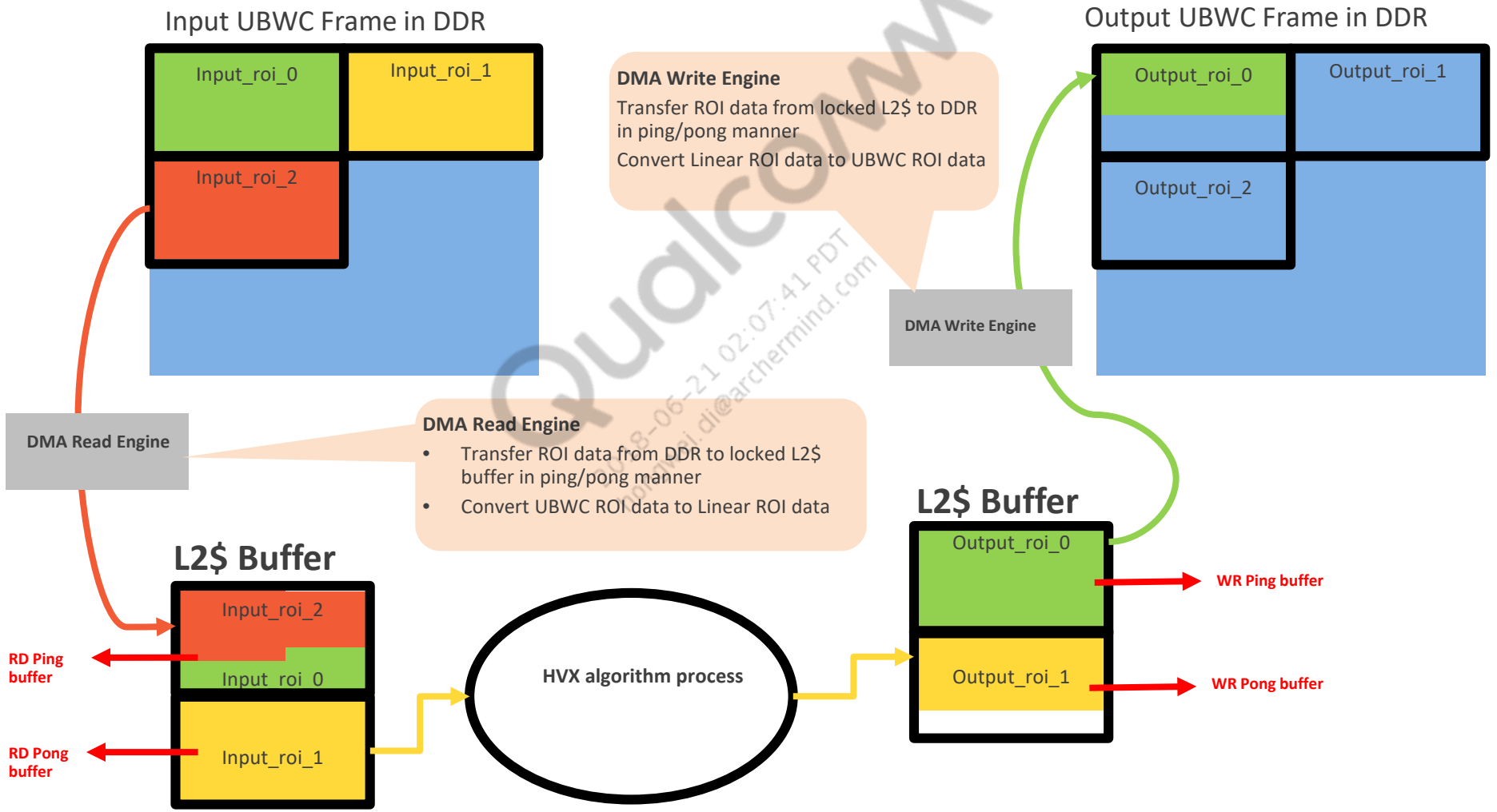


DMA Resources and User Process Allocations

- Typical user processes request DMA resources
- N number of user processes can access DMA engines as long as DMA engine is available
- One DMA engine per one DMA resource
- Each DMA resource has its own Hexagon DSP local memory region (TCM)
- Metadata cache is shared between DMA engines



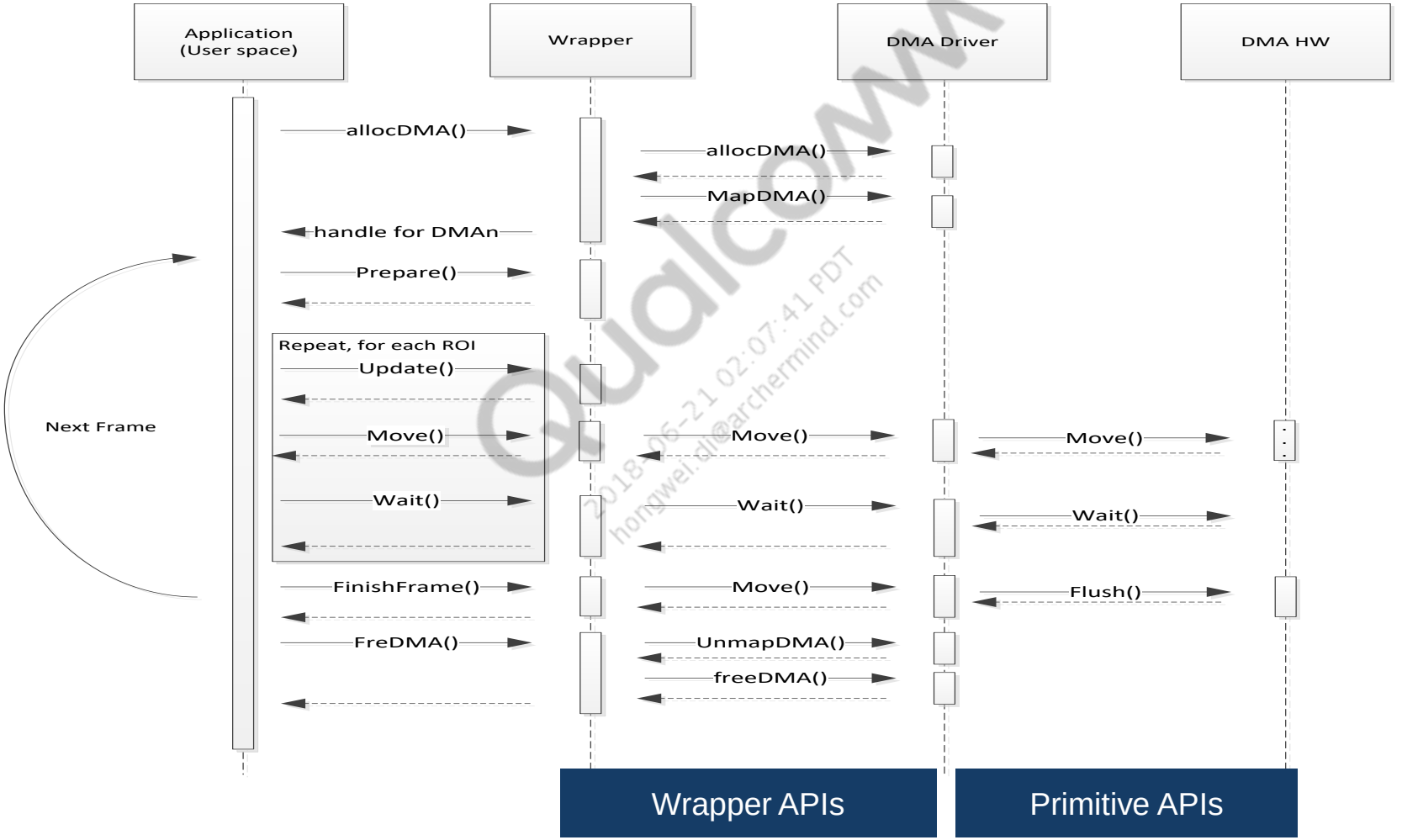
Typical UBWC Data Flow via DMA



DMA Programming Modes

- DMA firmware driver provides two programming modes:
  - High-level Wrapper mode
    - Helps speed up end-user developing cycle without developer knowing much about DMA block
    - DMA firmware driver provides high-level wrapper APIs to interface with DMA block
  - Primitive mode
    - For more advanced cases, can create custom high-level wrapper APIs to suit your needs
    - DMA firmware driver provides access to low-level APIs called Primitive APIs
    - Currently, this mode is not supported

DMA Programming Call Flow



High-Level Wrapper API Overview

- Per session
  - hDmaWrapper_AllocDmaSpecifyWaitType:
    - Allocates DMA engine with specific wait type – LLP, interrupt, spin-loop
    - Returns handle to be used for most other wrapper APIs
  - hDmaWrapper_AllocDma:
    - Allocates DMA engine with default wait type
    - Returns handle to be used for most other wrapper APIs
  - nDmaWrapper_FreeDma:
    - Uses DMA handle to free DMA engine
- Per frame
  - nDmaWrapper_Prepare:
    - Input – Frame details, ROI details, number of frames, and DMA handle
    - Prepares internal structures so DMA can ping-pong transfers
  - nDmaWrapper_FinishFrame:
    - Used to flush DMA

High-Level Wrapper API Overview (cont.)

- Per ROI
  - nDmaWrapper_Move:
    - Kicks off DMA to perform transfer transactions
  - nDmaWrapper_Wait:
    - Waits for outstanding transfer transactions
  - nDmaWrapper_Update:
    - Updates ROI and L2 cache address

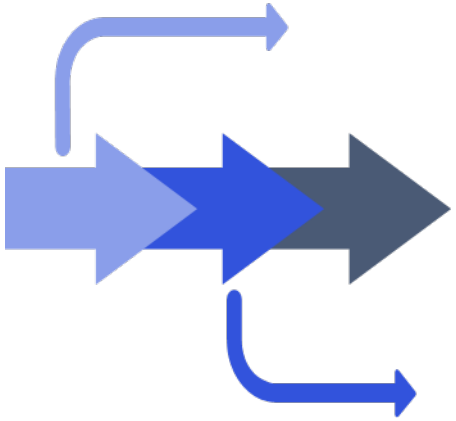
DMA Driver Use Case Example

- Get Hexagon SDK UBWC DMA Add-on release from CreatePoint, and extract it in following folder

  C:\Qualcomm\Hexagon_SDK\<Hexagon SDK Version>
- Documentation is available at:

  C:\Qualcomm\Hexagon_SDK\<Hexagon SDK Version>\docs\ubwcdma\ubwcdma.html
- Start with memcopy example:

  C:\Qualcomm\Hexagon_SDK\<Hexagon SDK Version>\examples\ubwcdma\dma_memcpy
- Copies DDR frame to Hexagon DSP local memory, and then copies it back to DDR



Profiling

Profiling – Source Code Functions

- Use the GetTime() function (defined in SDK example code) on application processor side when profiling end-to-end timing
- #define PROFILING_ON (defined in SDK example code for cDSP side skel implementation) provides processor cycles, execution time, and observed clock rate for cDSP

Profiling – Simulator

1. Run simulator with –profile and --timing options, to generate profiling data file and timing analysis data files
2. Generate profiling data

```
HEXAGON_Tools_DIR\Examples\HVX\integrate>make SRC=I sim
HEXAGON_Tools_DIR\Examples\HVX\integrate\test\build>hexagon-sim –profile
--timing --packet_analyzehist.json--integrate.exe 1920 1080 ..\..\..
\testvectors\football1920x1080.bin output.bin
HEXAGON_Tools_DIR\Examples\HVX\integrate\test\build>python ..\..\
..\..\Profiling\proftool\scripts\proftool.py --stathisthist.json
integrate.exe out.html
HEXAGON_Tools_DIR\Examples\HVX\integrate\test\build>hexagon-gprof
integrate.exe gmon.*.t_0 > gmon.*.log
```

- For information on gmon.*.log, see *Hexagon gprof Profiler User Guide* (80-N2040-29)
- For information on out.html, see *Hexagon Profiler User Guide* (80-N2040-10)
- These documents are released as part of Hexagon SDK and are at *C:\Qualcomm\Hexagon_SDK\3.3\tools\HEXAGON_Tools\8.1.03\Documents*

Profiling – Target

- sysMonApp, sysMon_DSP_Profiler_V2.apk, and SysmonParser
  - Available as part of cDSP build, which is available on ChipCode portal
  - Available at *Hexagon_SDK*\<*Hexagon SDK Version*>*tools**utils**sysmon*
- sysMonApp
  - Available at cdsp_proc\performance\sysmonapp\sysMonApp
  - Android native application executable in adb shell
  - Monitors and dumps key cDSP metrics in binary file
  - Detailed documentation is available in cDSP build folder cdsp_proc\performance\sysmonapp\readme.txt
- sysMon_DSP_Profiler_V2.apk
  - Android Java apk started from device is tested
- SysmonParser
  - cdsp_proc\performance\tools\SysmonParser
  - Parses binary file generated by sysmonappinto Microsoft Excel spreadsheet

Profiling – sysMonApp Usage

1. Copy sysMonApp from the cDSP build to the target /data
   folder:CDSP..._HY11\cdsp_proc\performance\sysmonapp
2. Copy the skel file (libsysmon_skel.so) from the following path to */system/vendor/lib/rfsa/adspfolder* on the *target:CDSP..._HY11\cdsp_proc\build\dynamic_modules_nonavs\845.cdsp.prod*
3. Run the clock setting:

```
C:\>adb shell /data/sysMonApp getstate --q6 cdsp
 Domain Configured Compute DSP
 DSP Core clock:144.00 MHz
 SNOC Vote:0.00 MHz
 BIMC Vote:2.50 MHz
 GuestOS: Total Heap:1024.00KB
 Available Heap:451.21KB
 Max.Free Bin:422.98KB
```


Note: Use the --q6 tag for selecting the correct DSP domain (aDSP, cDSP, sDSP).

Profiling – sysMonApp Usage (cont.)

4. Start profiling, press any key.

```
>adb shell /data/sysMonApp profiler --defaultsetenable1 --q6 cdsp
```

Parameters:

```
Q6 Processor: cdsp
Sampling Interval in ms: 1
Total samples: 0
samplesInSet: 16
Default Mode: 1
dcvs enable: 0
Domain Configured Compute DSP
Q6 architecture detected as v65...
opening outputfile @/sdcard/sysmon_cdsp.bin
Enabling DSP SysMon using FastRPC
Allocating output buffer
```

Profiling – sysMonApp Usage (cont.)

5. To stop profiling, press any key.
   The following message is displayed:

```
>> Starting thread to Query DSP SysMon for samples
>> Waiting for a keyboard input...
>> Sending kill to Query thread...

>> Waiting for the Query thread to join...
<< Received TERMINATE query signal

**************************EXITING!**************************
```

6. The output bin file is placed in */sdcard: /sdcard/sysmon_cdsp.bin*

Profiling – SysmonParser Usage

1. Get the dump file from the target:

   ```
   >adb pull /sdcard/sysmon_cdsp.bin c:/temp
   ```

   For detailed information, see: *adbshell /data/sysMonAppprofiler*
2. Go to the SysmonParser folder in the cDSP build: CDSP..._HY11\cdsp_proc\performance\tools
3. Run the SysmonParser: `>SysmonParser.exe sysmon.bin sysmon_dump user`

Profiling – SysmonParser Usage (cont.)

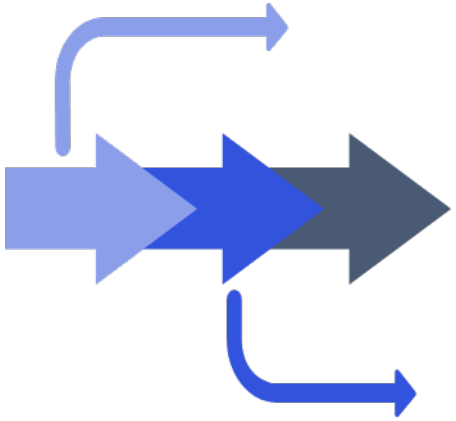
- SysmonParser generates output as shown in the following example
  - HVX MPPS shows number for overall HVX module
  - Effective Hexagon frequency is minimum cDSP clock for running this process

Overall summary														
Overall test duration – 14.63														
Processor Metrics					**Bus metrics**					**L1 cache metrics**				
Metric	**Unit**	**Avg**	**Max**	**Min**	**Metric**	**Unit**	**Avg**	**Max**	**Min**	**Metric**	**Unit**	**Avg**	**Max**	**Min**
MPPS	MPackets/ sec	1.192	359.27	0.05	AXI 64 byte Cached Rd BW	MBPS	4.117	1486.34	0.06	IU STALL CYCLES	MHz	2.354	184.86	0.36
1THREAD ACTIVE	MHz	5.164	430.19	0	AXI 64 byte Cached Wr BW	MBPS	1.05	639.74	0.01	DU CACHE MISS PVIEW FREQ	MHz	1.614	255.11	0.19
pCPP	Cycles/ Packets	6.3	43.07	1.87	HVX Metrics					L2 Cache Metrics				
NPA CORE Clk	MHz	402.86	729.6	124.8	HVX THREADS MPPS	MPkts/ sec	0.227	284.68	0	L2 FETCH MISS	MBPS	0	0	0
NPA HSSNOC Clk	MHz	345.15	750	0.626										
NPA BIMC Clk	MHz	690.3	1500	1.25										
MEASURED snoc_clk	MHz	219.2	240	200										
MEASURED bimc_clk	MHz	520.3	902.4	200										
ADSPPM CORE Clk	MHz	426.13	825	80										

Profiling – SysmonParser Usage (cont.)

- Further documentation for interpreting data and usage on previous slide is available at:
  - *C:\Qualcomm\Hexagon_SDK\<Hexagon SDK Version>\docs\Hexagon_Profiler_v2.html*
  - *C:\Qualcomm\Hexagon_SDK\<Hexagon SDK Version>\docs\Hexagon Profiler.html*

Overall summary													
Overall test duration – 14.63													
Processor metrics					**Bus metrics**					**L1 cache metrics**				
Metric	**Unit**	**Avg**	**Max**	**Min**										
ADSPPM SNOC Clk	MHz	750	750	750										
DCVS CORE Clk	MHz	0	0	0										
DCVS SNOC Clk	MHz	0	0	0										
DCVS AHB Clk	MHz	27.58	60	0										
Eff Q6 Freq	MHz	7.503	731.33	0.98										
Total Avail Heap(GuestOS)	MB	0.62	0.64	0.61										
Largest Avail Heap Block(GuestOS)	MB	0.56	0.58	0.55										
Total Available Heap (AudioPD)	MB	8.33	8.33	8.33										
Largest Avail Heap Block(AudioPD)	MB	8.26	8.26	8.26										



Debugging

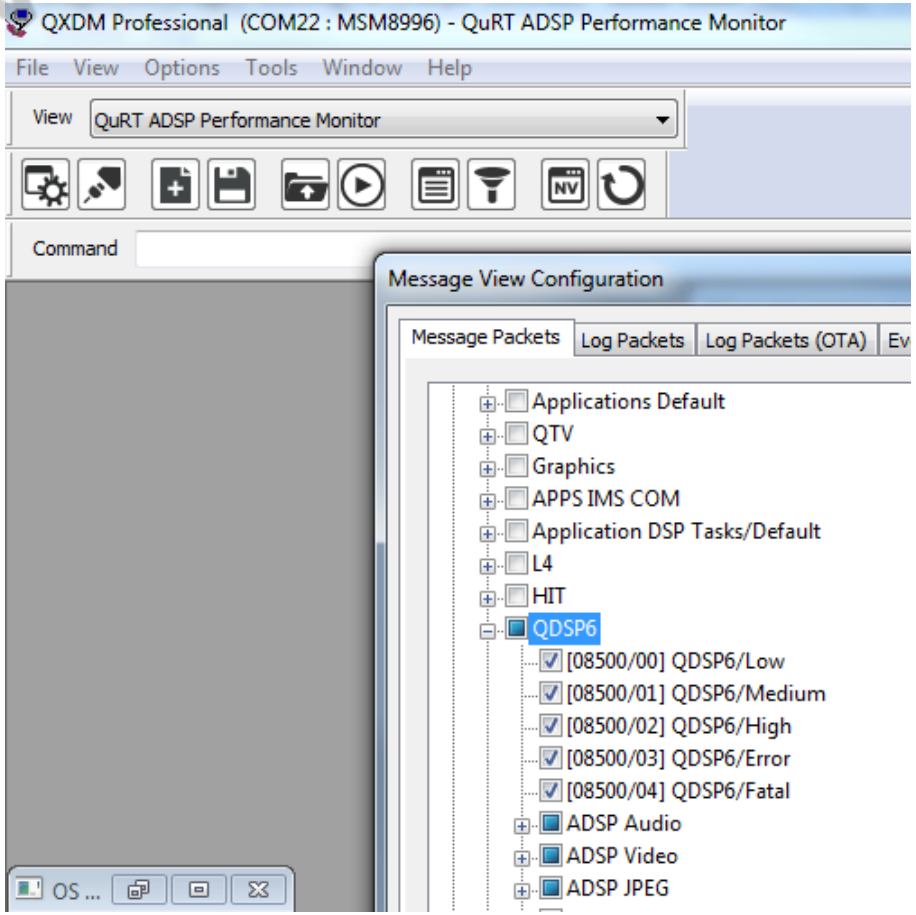
Debugging – Tools

- QXDM Professional – For debug messages, available on CreatePoint
- Mini-dm – Available as part of Hexagon SDK installation
  - *C:\Qualcomm\Hexagon_SDK\<Hexagon SDK Version>\tools\mini-dm*
- LLDB – Available as part of Hexagon toolchain installation
  - *C:\Qualcomm\Hexagon_SDK\<Hexagon SDK Version>\tools\HEXAGON_Tools\<HEXAGON_Tools Version>\Tools\bin\hexagon-lldb.exe*

Debugging – Tools (cont.)

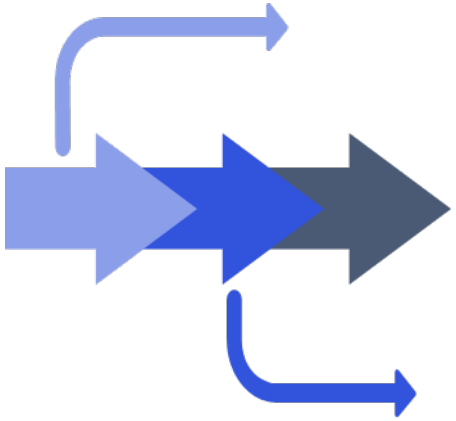
- QXDM Professional
  - F3 message shows DSP log message
  - In QXDM Pro, enable DSP log mask:
    1. Select Options → Message View → Hexagon DSP
    2. Open the F3 message window
- Min-DM
  - Min-DM shows same log as QXDM Pro
  - Recommended if you are not familiar with QXDM Pro
  - In device manager or QPST, check comport number of USB diagnostic port
  - Close QXDM Pro and QPST because Min-DM shares same 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 – Crash and Stability

- QPST – Crash dump collection
- aDSP Crashman – Ramdump parsing
- TRACE32 – Dump loading, Scripts
- For more information, see *Qualcomm Hexagon aDSP and cDSP Stability Debugging Guide* (80-P8754-61)



References

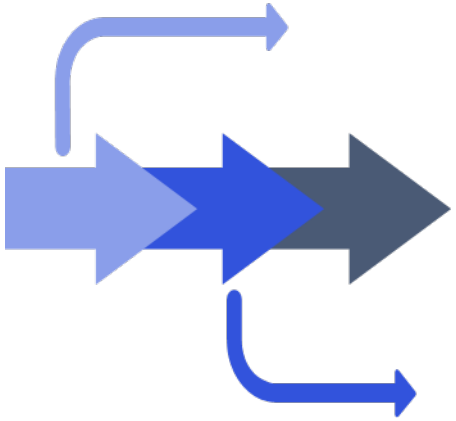
References

Documents	
Title	**Number**
Qualcomm Technologies, Inc.	
Qualcomm Hexagon Access HVX Streaming Customization for Camera for MSM8996/MSM8998/SDM660/SDM670	80-NF772-36*
Qualcomm HVX Quick Start	80-NV396-69
Qualcomm FastCV Integration	80-NE989-7
Qualcomm Hexagon aDSP and cDSP Stability Debugging Guide	80-N4597-2
FastRPC Training	80-N7039-1
Hexagon Programming: Vector Extensions (HVX)	80-VB419-97
Hexagon Profiler User Guide	80-N2040-10
Hexagon gprof Profiler User Guide	80-N2040-29
Snapdragon Computer Vision Overview	80-NL315-9
Hexagon V65 HVX Programmer Reference Manual	80-N2040-41
Hexagon V65 Programmer Reference Manual	80-N2040-39
Hexagon v60 HVX Instruction Quick Reference	80-N2040-35
Resources	
Hexagon SDK, Hexagon Toolchain	Hexagon_SDK\3.3\docs at https://developer.qualcomm.com/

***** Updated document will be available by ES

References (cont.)

Acronyms	
Acronym or term	**Definition**
aDSP	Application digital signal processor
cDSP	Compute digital signal processor
DCVS	Dynamic clock and voltage scaling
GPU	Graphics processing unit
HLOS	High-level operating system
HVX	Hexagon vector extension
LPI	Low-power island
Mops	Million operations
QDSS	Qualcomm debug sub system
SSR	Sub system restart
TCB	Task control block
TCM	Tightly coupled memory
UBWC	Universal bandwidth compression
VFE	Video front-end of camera subsystem
VLIW	Very long instruction word
VTCM	Vector TCM



Questions?

https://createpoint.qti.qualcomm.com