Hexagon DSP: An Architecture Optimized for Mobile Multimedia and Communications

The Qualcomm Hexagon DSP is used for both modem processing and multimedia acceleration. By offloading multimedia tasks from the CPU to the DSP, significant power savings can be achieved. This article provides an overview of the Hexagon architecture. The processor is designed to deliver superior energy efficiency compared to mobile CPU alternatives and thereby help achieve long battery life for important mobile applications.

Lucian Codrescu
Willie Anderson
Suresh Venkumanhanti
Mao Zeng
Erich Plondke
Chris Koob
Ajay Ingle
Charles Tabony
Rick Maule
Qualcomm

To be competitive, a modern mobile product must provide a rich user experience and long battery life. Chips for these ecosystems integrate multiple subsystems, each customized for a particular application domain. By specializing a subsystem to a task, performance and power can be enhanced beyond what is possible with a homogenous CPU-based computing platform.

Figure 1 shows a block diagram of the Snapdragon 800. This chip contains dedicated subsystems for camera, display, video, audio/voice, sensors, graphics, cellular modem, Wi-Fi, and more. Each subsystem contains dedicated hardware, and many contain special-purpose processing engines and software customized to the task.

The Snapdragon 800 has two instances of the Hexagon digital-signal processor (DSP). The modem (mDSP) is dedicated and customized for modem processing, whereas the application DSP (aDSP) is used for multimedia acceleration. The modem processor is a closed subsystem and is programmed only within Qualcomm Technologies. The multimedia DSP, however, is licensed for programming by OEMs and third-party software vendors. This article provides an overview of the multimedia DSP and builds on the presentation from HotChips 25.[1]

Figure 2 shows the various Hexagon generations. Version 2 (V2) was the first production version and appeared in the initial Snapdragon mobile products in 2007. V3 featured an improved implementation with better power consumption. These early versions of Hexagon targeted voice and audio processing. Example functions include wideband vocoders, echo cancellation, audio postprocessing filters, MP3/AAC playback, speaker protection algorithms, and so on.

V4 and V5 expanded the application targets to include image processing for camera and video; computer vision tasks such as hand, gesture, and face recognition; and processing of sensor input (gyro, accelerometer, fingerprint, and so on). Unless otherwise noted, this article will focus on the latest Hexagon V5 core.

Published by the IEEE Computer Society

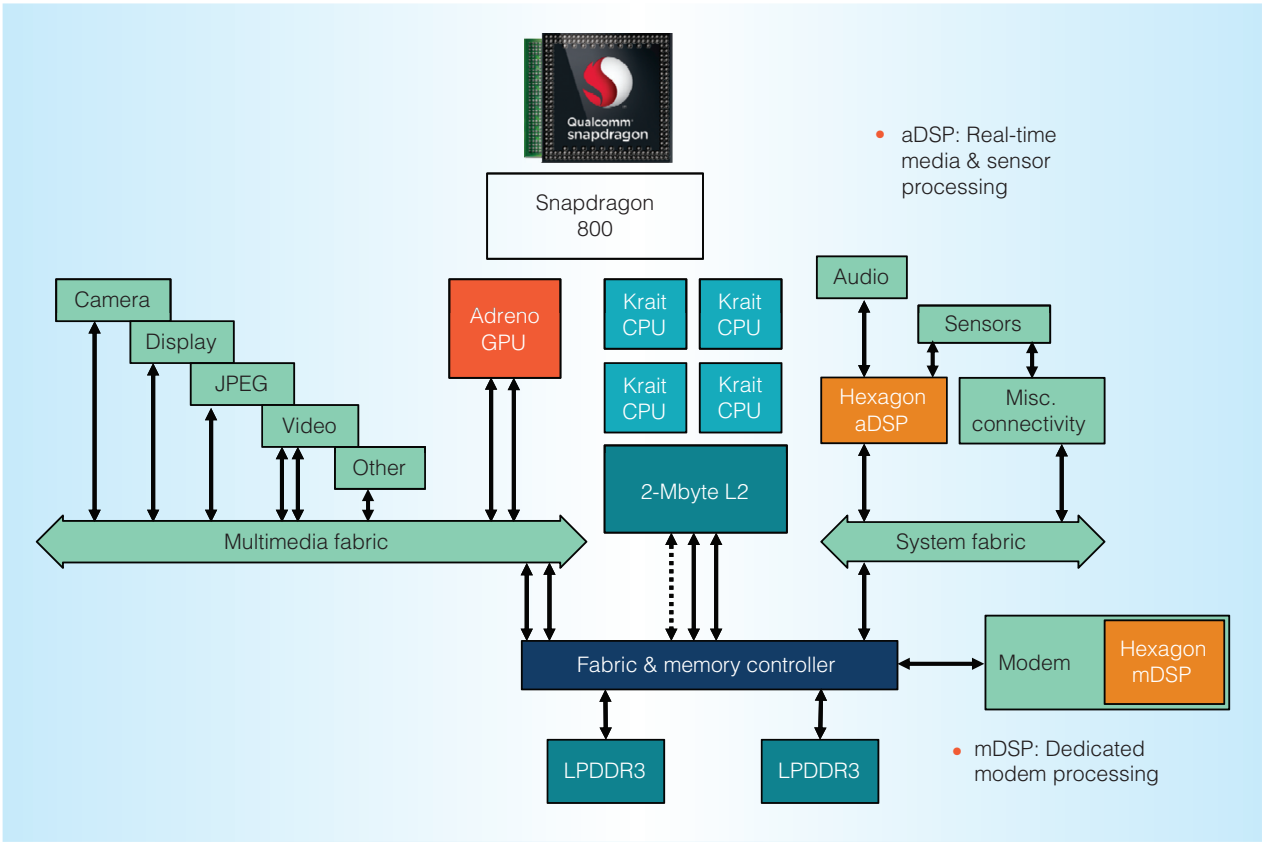

Figure 1. Snapdragon 800 block diagram. The chip contains dedicated subsystems for camera, display, video, audio/voice, sensors, graphics, cellular modem, and Wi-Fi.

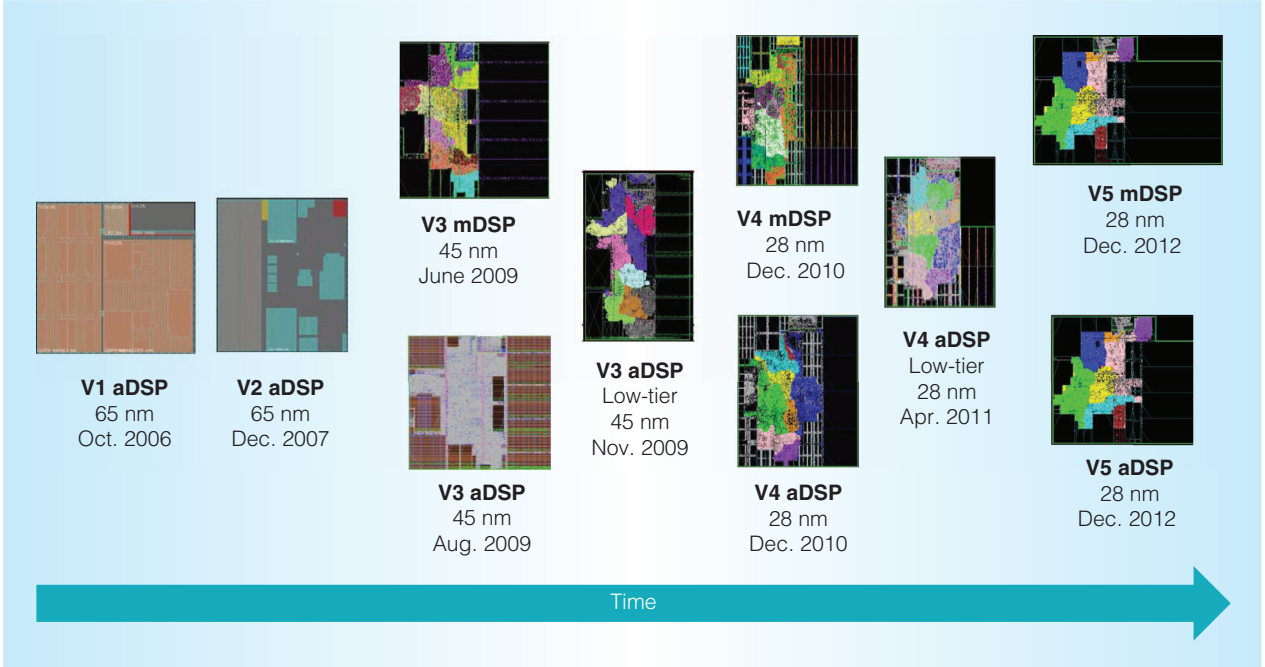

Figure 2. Hexagon digital-signal processor (DSP) evolution. The figure shows the evolution from Version 1 in October 2006 through Version 5 in December 2012.

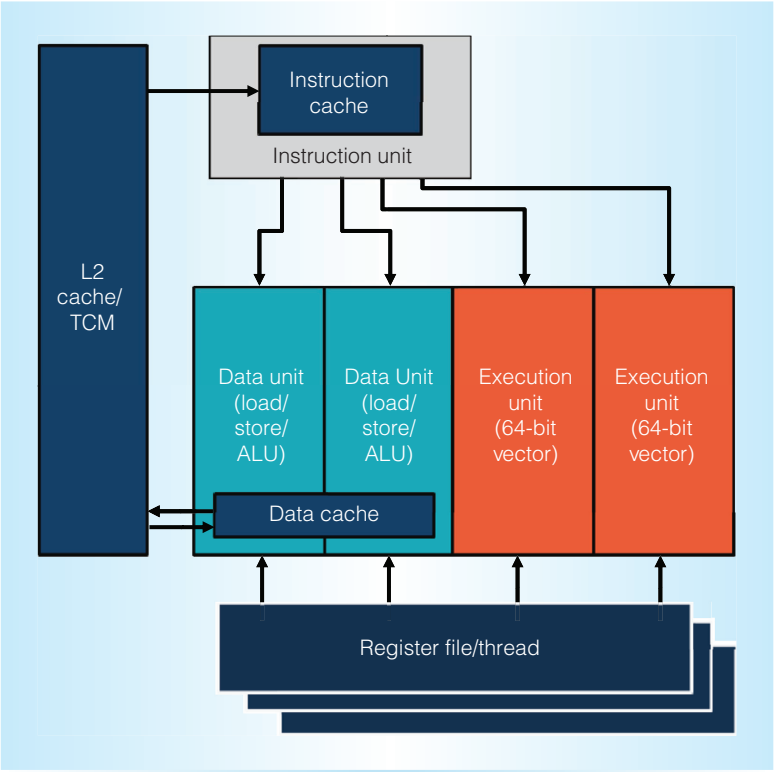

Figure 3. Hexagon block diagram. The architecture features a four-wide very long instruction word (VLIW) with dual load/store and dual single-instruction, multiple-data (SIMD) execution units and supports hardware multithreading.

Hexagon is a multithreaded very long instruction word (VLIW) DSP. The design philosophy is to maximize work per cycle for performance, but target the microarchitecture to modest clock speeds and low power.

Instruction-set architecture overview

The architecture's foundation is a statically scheduled four-way VLIW. The VLIW approach puts the burden of instruction parallelism on the compiler and thereby avoids costly and power-hungry dynamic-scheduling hardware.[2] The VLIW approach is popular among commercial DSPs. Figure 3 shows a block diagram of Hexagon.

Registers and memory

The Hexagon processor features a unified byte-addressable memory. This memory has a single 32-bit virtual address space that holds both instructions and data. It operates in little-endian mode. A full-featured memory management unit (MMU) translates virtual to physical addresses.

All user-level registers are replicated per thread. There are two sets of user registers: general registers and control registers. The general registers include thirty-two 32-bit registers that can be accessed either as single registers or as aligned 64-bit register pairs. The general registers contain all pointer, scalar, vector, and accumulator data. The control registers include special-purpose registers such as the program counter, status register, and loop registers.

Data-processing instructions

There are two identical 64-bit single-instruction, multiple-data (SIMD) execution units. Each unit supports all multiply, shift, arithmetic logic unit (ALU), and bit manipulation instructions. Supported data types include

- 8-, 16-, 32-, and 64-bit integer;
- 16- and 32-bit fractional with optional rounding and saturation;
- 16-bit complex; and
- single-precision IEEE-compatible floating point.

Each unit is capable of supporting:

- four 16 × 16 multiplies;
- two 32 × 16 multiplies; or
- one 32 × 32 multiply, one complex multiply, or one floating-point fused multiply-add (FMA).

Many of the instructions are complex and application specific. Complex instructions targeted to a particular application domain can provide high performance and energy efficiency. For example, Figure 4 depicts a complex multiply instruction used in a 16-bit fixed-point fast-Fourier transform (FFT). Without such an instruction, it would take four multiplies, four shifts, four adds, and two saturates to perform the operation. It should be clear that packing all the work in a single instruction executed in a single pipelined execution unit provides large efficiency gains.

The Hexagon instruction set architecture (ISA) contains numerous special-purpose instructions designed to accelerate key multimedia kernels. Multimedia algorithms with special instruction support include

- variable-length encode/decode, such as context-adaptive binary-arithmetic-coding processing in H.264 video;
- features from accelerated segment test (FAST) corner detection image processing;
- FFT algorithms;
- sliding-window filters;
- linear-feedback shift;
- table lookup from an arbitrary bit field index;
- elliptic curve cryptography; and
- cyclic redundancy check (CRC) calculation.

Load/store instructions

Dual load/store units access signed or unsigned 8-, 16-, 32-, and 64-bit values in memory. There is a rich variety of addressing modes, including

- absolute 32-bit,
- base plus scaled immediate and base plus scaled register,
- auto-incrementing by register and immediate,
- circular addressing, and
- bit reversed.

To increase the number of instruction combinations allowed in packets, the load/store units also support 32-bit ALU instructions.

Conditional execution and program flow

The Hexagon ISA includes conditional execution. Conditional execution is useful to remove branches through if-conversion and is helpful for a VLIW processor. Compare instructions target one of four predicate registers. These predicate registers can then be used to conditionally execute certain instructions. Not all instructions can be conditional—only the most common load/store and ALU instructions.

A unique feature of Hexagon conditional execution is that the processor can generate and use a predicate in the same VLIW instruction packet. This reduces packet count and creates denser packets, both of which improve performance and reduce energy consumption. Consider the following C statement and the corresponding assembly code that is generated from it by the compiler. The ".new" suffix implies the source predicate is generated in the same packet. In this example, the dot-new construct enables the work to be done in one instruction packet instead of two.

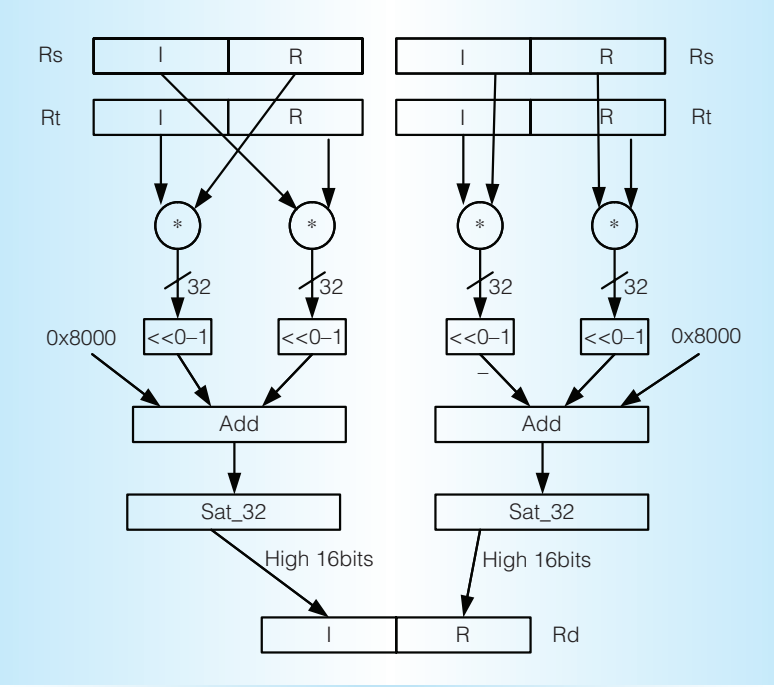

Figure 4. Complex multiply instruction. Such an instruction executed in a single pipelined execution unit provides large efficiency gains for applications that use complex arithmetic.

The C statement is as follows:

```
if (R2 == 4)
 R3 = *R4;
else
 R5 = 5;
```

Assembly code with braces delineate packet boundaries:

```
{
P0 = cmp.eq(R2,#4)
if (P0.new) R3 = memw(R4)
if (!P0.new) R5 = #5
}
```

Similar to many DSP processors, Hexagon includes a zero-overhead hardware counted looping mechanism with support for two levels of nesting. An instruction is used to initialize the loop count and the start address. Bits encoded in the last packet of the loop delineate the end of the loop. This architecture allows execution of loops with

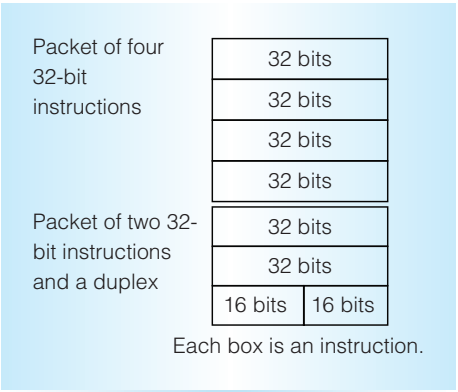

Figure 5. Visualization of a duplex. A duplex is a 32-bit subpacket containing two 16-bit instructions.

no branch mispredicts or stalls, and no hardware devoted to loop branch prediction.

Compound and memop instructions

Compound instructions combine two or more dependent operations in a single instruction. These instructions improve code size and save power by reducing register file and forwarding power. Hexagon includes many such instructions, including shift-add, shift-or, add-add, compare-branch, shift-xor, and many flavors of the classic DSP multiply-add.

Another class of instruction performs simple operations directly on memory, including add, subtract, logical–or, and logical–and. Without these memory operations (memops), three instructions would be necessary to perform the same task: one to load the value, one to perform the arithmetic or logical operation on the value, and one to store the result. Memops improve code size and reduce power because intermediate register access is not needed.

VLIW instruction grouping

VLIW instruction packets are variable sized and contain one to four instructions. If a packet contains more than one instruction, the instructions execute in parallel. The instruction combinations allowed in a packet are limited to the instruction types that can be executed in parallel in the four execution units. The processor uses parallel execution semantics. All registers are read, then all instructions are executed, then all registers are written.

Duplex instructions

Low code size is advantageous for an embedded processor. Hexagon instructions are fixed size and 32 bits in length. To improve code size, the Duplex feature enables some use of 16-bit instructions by creating a 32-bit subpacket containing two 16-bit instructions. These subpackets are called *duplexes*. Figure 5 shows a visualization of a duplex.

Because duplexes are always 32 bits, packet sizes continue to be multiples of 32 bits. This leads to a simpler and lower-power implementation as compared to instruction sets with a mixed 16-/32-bit instruction set. Additionally, duplexes must always end a packet, and are always dispatched to the same two execution units, which further simplifies the implementation. The instructions allowed in duplexes, called *subinstructions*, are the most common subset of normal Hexagon instructions, with reduced ranges of registers and immediate operands.

Multithreading and microarchitecture

The Hexagon processor is multithreaded. The number of threads varies by implementation. Early implementations included six hardware threads, but more recent cores include three hardware threads. There are many trade-offs in choosing the number of threads. Additional threads provide more latency tolerance and enable power-saving opportunities in the microarchitecture by serializing work rather than speculating work. On the other hand, additional threads increase cache pressure and increase the software programming burden. Our experience is that three or four threads are a sweet spot in the design space.

Hexagon is designed to look like a multicore architecture with communication through shared memory. Figure 6 shows how the processor appears to the programmer. Software threads are mapped to hardware threads by the operating system.

In the physical implementation, however, there is only one processor, which the three hardware threads share. Hexagon V1 through V4 implemented a simple round-robin interleaved multithreading (IMT) approach.[3] On every clock tick, a different thread is given a

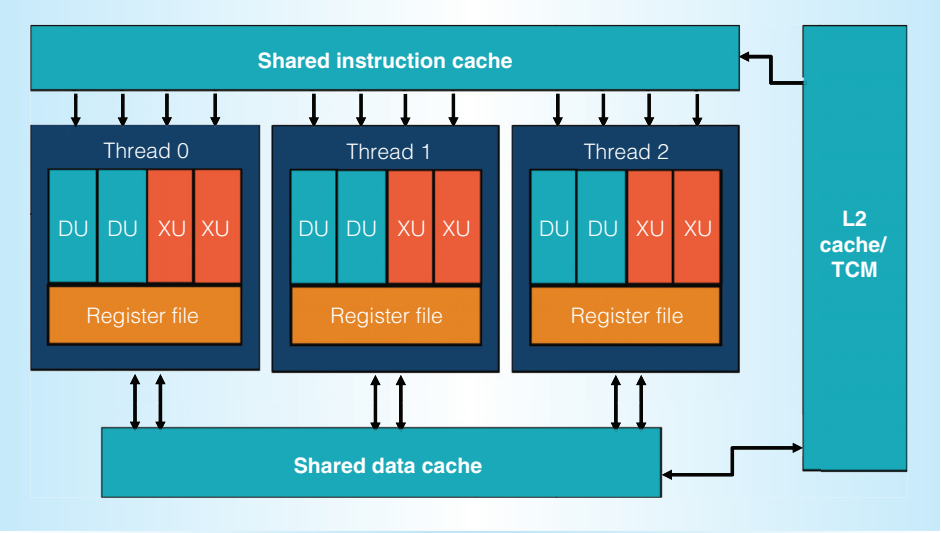

Figure 6. The programmer's view of multithreading. To the programmer, it appears as three VLIW cores with shared caches. Software threads are mapped to hardware threads by the Hexagon operating system.

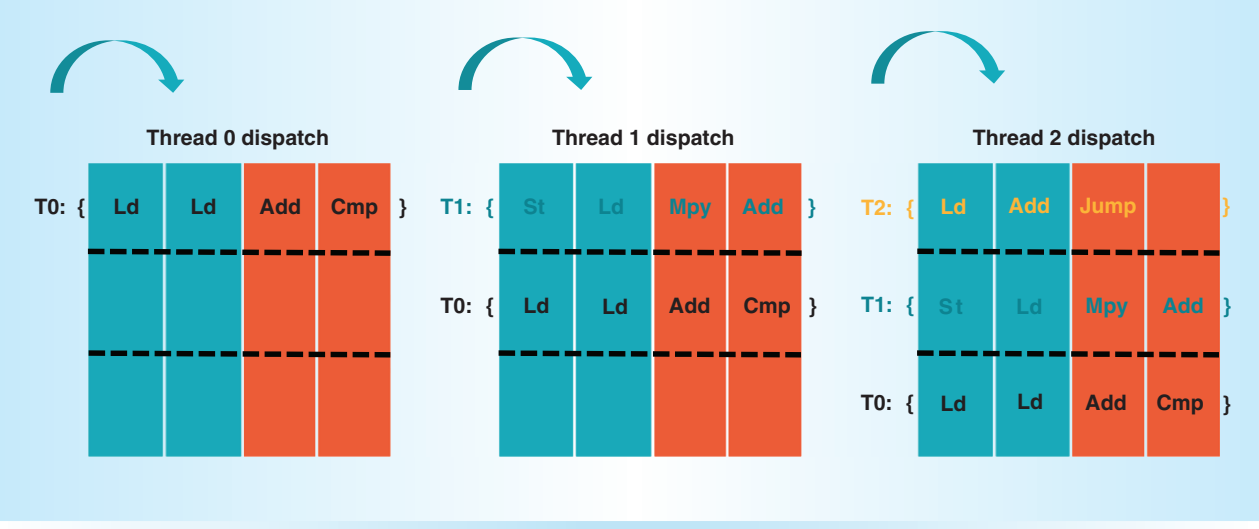

Figure 7. Interleaved multithreading. The figure shows a three-stage execution pipeline with three threads taking turns dispatching packets.

turn at each pipe stage. Figure 7 shows a three-stage execution pipeline and with three threads taking turns dispatching packets.

With the number of threads matched to the execution pipe depth, all of a thread's instructions from a VLIW packet are complete before the next VLIW packet starts. Because there is no observable latency, the compiler is not concerned with instruction latency and scheduling for latency. This yields higher VLIW packet density. When instructions are no longer needed to hide latency, they can be used instead to fill the packets.

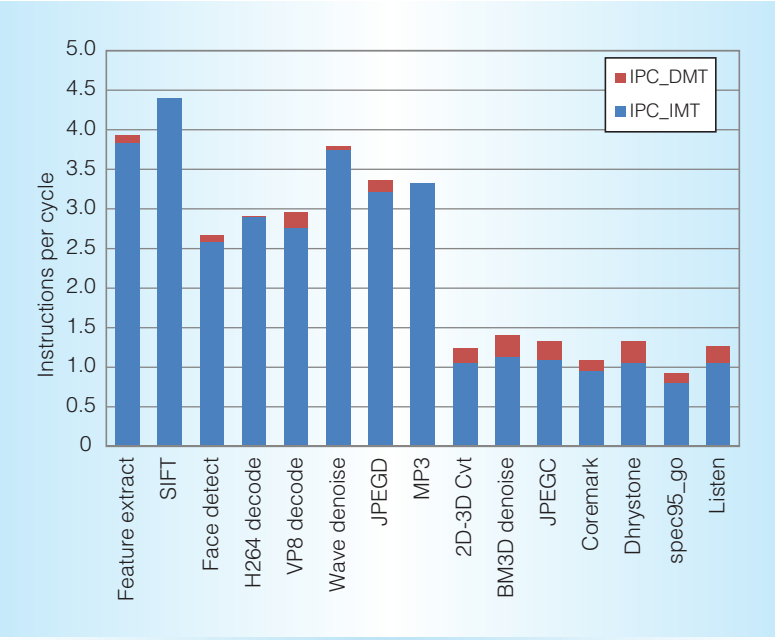

Figure 8. Performance on multimedia applications. The figure shows the instructions per cycle (IPCs) for various multimedia benchmarks, which are sorted into multithreaded applications (left) and single-threaded applications (right).

The simple IMT model and simple in-order pipeline yield a small, low-power processor that is critical to meeting the aggressive area and power targets.

The obvious problem with IMT is that when threads are idle or stalled, their slice of the processor goes unused. Starting with Hexagon V5, a more dynamic approach to thread scheduling has been implemented. Often, packets contain only simple instructions and can be completed in fewer than three cycles. With Hexagon V5, the processor will opportunistically execute packets faster if threads are idle or stalled and simple packets are available. The design philosophy is not to shoot for the best single-thread performance, but rather to provide some performance boost when it is easy to do so and without compromising energy efficiency.

Figure 8 shows the instructions per cycle (IPCs) for various multimedia benchmarks. The benchmarks are sorted into multithreaded applications on the left, and single-threaded on the right. The boost from the V5 dynamic multithreading (DMT) is shown as the additional (striped) bar on top of the baseline (solid) IMT bar.

In the Snapdragon 800 implementation, the DSP runs up to 800 MHz. The instruction cache is 16 Kbytes, the data cache is 32 Kbytes, and the level-2 (L2) cache is 256 Kbytes. Connection to main memory is provided over a 64-bit system bus that runs at 240 MHz.

System programming model

Communication between the DSP and CPU is done through a traditional shared-memory-plus-interrupt mechanism. Both the DSP and CPU can access the full physical address space and share the external memory. Access to memory is cache based, and there is no explicit data mover. The DSP includes an extensive prefetching capability to help hide cache latency. The CPU and DSP are not cache coherent with each other, so coherency must be maintained in software with explicit cache maintenance operations.

A software remote procedure call (RPC) interface lets a CPU application offload work to the DSP. When an RPC is made, any data associated with the call is flushed to main memory from the CPU caches and mapped into the DSP virtual address space. The DSP is then interrupted to process the RPC call, after which any results are flushed from the DSP caches back to main memory, and a completion interrupt is sent to the CPU.

The overheads of software-managed coherency preclude offloading very small tasks to the DSP. Large kernels that run continuously or process large data (full image frames) are typically needed to amortize the overhead.

Power

Battery life is extremely important in mobile computing. Offloading applications from the CPU to a specialized low-power processing engine such as Hexagon is critical to achieving the power goals. In addition to the ISA and microarchitecture for efficient multimedia processing, Hexagon is implemented with aggressive low-power design techniques, including hierarchical clock gating with a custom clock tree, voltage scaling with split-grid memories, pulse latches

instead of flip-flops, and full-custom caches and register files designed for low power. A more complete description of the low-power techniques used by Hexagon is available elsewhere.[4]

An important power benchmark for mobile phones is MP3 playback. Figure 9 compares current measured at the battery for various competitive smartphones. Battery current includes all components of system power, such as the CPU, DSP, memory, and I/O. The data is presented as (battery current in mA for the device divided by battery current in mA for the Hexagon-based chip). A 2× delta on this chart represents twice the hours of music playback, which is a key marketing and user-experience metric.

Google recently announced support for DSP offload of audio playback in the Android 4.4 (KitKat). Quoting from the "Audio Tunneling to DSP" section on Google's Android developer webpage:[5]

> For high-performance, lower-power audio playback, Android 4.4 adds platform support for audio tunneling to a digital signal processor (DSP) in the device chipset. With tunneling, audio decoding and output effects are off-loaded to the DSP, waking the application processor less often and using less battery.
>
> Audio tunneling can dramatically improve battery life for use-cases such as listening to music over a headset with the screen off. For example, with audio tunneling, Nexus 5 offers a total off-network audio playback time of up to 60 hours, an increase of over 50% over non-tunneled audio.

The Nexus 5 device uses Snapdragon 800, and the audio offload is done to the Hexagon V5 aDSP.

Figure 10 shows another example of offloading a computer-vision-object detection algorithm from the CPU to the Hexagon DSP. Initially, the algorithm is run on the CPU. The algorithm is fully optimized with Neon SIMD instructions. After offloading to the DSP, the same algorithm is called via an RPC. Because the algorithm is no longer running on the CPU, the CPU load is reduced. In terms of speed, the algorithm is marginally faster on the DSP. This includes all RPC overheads. But, significantly, the total system power as measured at the battery has been reduced by 32 percent.

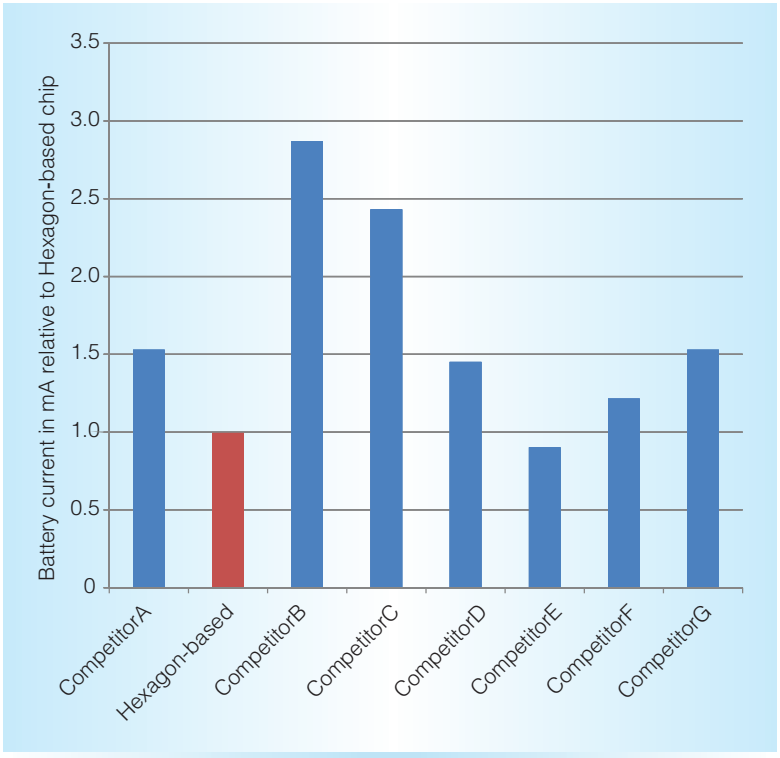

Figure 9. MP3 playback battery current for various smartphones. Lower bars represent more hours of music playback, which is a critical customer metric.

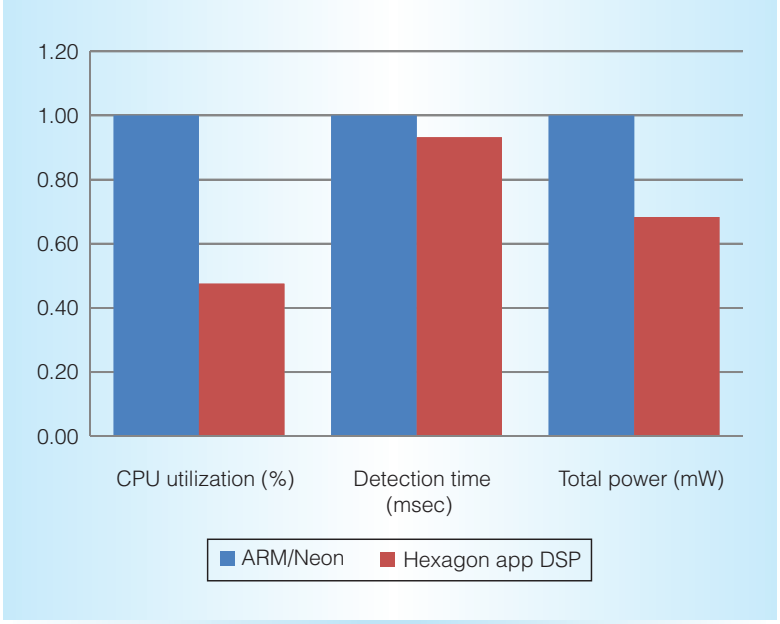

Figure 10. An example of feature detection offload. The algorithm is offloaded from the CPU to DSP, which results in comparable performance but much reduced battery current.

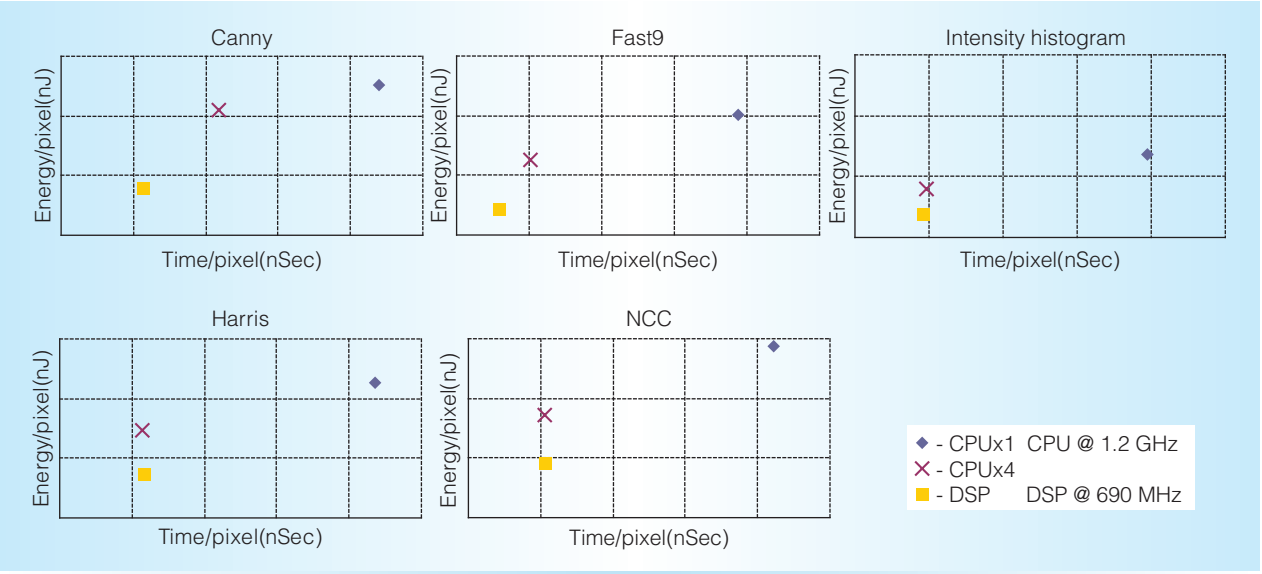

Figure 11. Energy versus latency for multimedia benchmarks. Each chart features a different algorithm used in a computer vision application.

Figure 11 provides additional examples of performance and power, comparing the Hexagon V5 DSP to the CPU used in the Snapdragon 200 chip. Each chart features a different algorithm used in a computer vision application. For the CPU, all code is fully optimized using Neon SIMD instructions. Both single-CPU and quad-CPU data is shown. The *x*-axis shows latency in units of time/pixel, and the *y*-axis shows energy/pixel. The origin is (0,0) in all charts. Power is measured at the battery and includes all system power. When compared to the quad CPU in Snapdragon 200, the Hexagon V5 DSP provides similar or better performance and lower power in all cases.

This article provides an overview of the Hexagon DSP architecture. The demand for low-power signal processing in mobile applications continues unabated. Camera and video applications require sophisticated signal processing at ultra-high definition resolution. At the same time, "always-on" voice activation features are pushing power requirements to new lows. Future versions of Hexagon DSP will be enhanced and specialized to tackle these upcoming challenges.

For readers who would like to explore further, the Hexagon Software Developer's Kit (https://developer.qualcomm.com/mobile-development/maximize-hardware/multimedia-optimization-hexagon-sdk/multimedia-optimization-h-2) provides everything needed to program the DSP, including full documentation, software tools, a cycle-approximate simulator, and example code. MICRO

References

1. L. Codrescu et al., "Qualcomm Hexagon DSP: An Architecture Optimized for Mobile Multimedia and Communications," Hot Chips 25, 2013.
2. J. Fisher, "VLIW Architectures: An Inevitable Standard for the Future?" *J. Supercomputer*, vol. 7, no. 2, 1990, pp. 29-36.
3. B. Smith, "Architecture and Applications of the HEP Multiprocessor Computer System," *SPIE Real Time Signal Processing IV*, 1981, pp. 241-248.
4. M. Saint-Laurent et al., "A 28 nm DSP Powered by an On-Chip LDO for High-Performance and Energy-Efficient Mobile Applications," to be published in *Proc. IEEE Int'l Solid-State Circuits Conf.*, 2014.

5. "Android KitKat: New Media Capabilities," Android.com, Jan. 2014, http://developer.android.com/about/versions/kitkat.html#44-media.

Lucian Codrescu is a senior director at Qualcomm, where he leads the Hexagon Architecture team. Codrescu has a PhD in computer engineering from the Georgia Institute of Technology.

Willie Anderson is a vice president at Qualcomm, where he runs the DSP engineering organization. Anderson has a BS in physics from the University of Texas at Austin.

Suresh Venkumanhanti is a principal engineer at Qualcomm, where he is responsible for the DSP core microarchitecture. Venkumanhanti has an MS in electrical engineering from the University of Florida.

Mao Zeng is a senior staff engineer at Qualcomm and a principal designer of the DSP instruction set architecture. Zeng has a PhD in electrical engineering from the University of Victoria.

Erich Plondke is a senior staff engineer at Qualcomm, where he leads the system architecture team. Plondke has an MS in electrical engineering from the Georgia Institute of Technology.

Chris Koob is a principal engineer at Qualcomm, where he is responsible for the memory system microarchitecture. Koob has an MS in electrical engineering from Virginia Tech.

Ajay Ingle is a principal engineer at Qualcomm, where he is responsible for DSP architecture support of modem applications. Ingle has an MS in electrical engineering from Texas A&M University.

Charles Tabony is a senior engineer at Qualcomm, where he contributes to the instruction set design and performance verification. Tabony has a BS in computer engineering from the University of Texas.

Rick Maule is a senior director at Qualcomm, where he is responsible for DSP product management. Maule has an MS in electrical engineering from the University of Arkansas.

Direct questions and comments about this article to Lucian Codrescu at lucian.codrescu@gmail.com.