

Tensilica White Paper

**Adding Video to SOCs:
The Diamond 388VDO Video Engine**

July, 2007

Executive Summary

Since its introduction in the early 1990s, video compression has grown increasingly important for the design of modern electronic products because it aids in the quest to deliver high-quality video using limited transmission bandwidth and storage capacity. The first widely successful video-compression standard was MPEG-2 used in DVD players. The MPEG-4 and H.264/AVC video-compression standards then followed some years later. With all video-compression standards, the goal is to deliver high-quality video with minimum bandwidth. As compression technology advances, codec (coder-decoder) developers can exploit increasing processor performance to achieve higher compression ratios while delivering better images.

Many consumer products incorporate video-recording and video-playback functions. SOC and ASSP design teams developing chips for such products need ways to quickly add proven, tested video functions to their designs. Tensilica's Diamond 388VDO Video Engine provides such a capability. Coupled with Tensilica's Diamond 330HiFi Audio Engine, IC designers can drop an AV solution into their design as a component.

Video Basics

Digital video encoding starts with a series of still images (frames) captured at a certain frame rate (usually 15, 25, or 30 frames/sec) by cameras use CCD or CMOS sensors to capture the images. These sensors image red, green, and blue (RGB) light but the RGB images they produce do not directly correspond to the way the human eye works. The human eye uses rods and cones to separately sense light intensity (luma) and color (chroma). The eye is much more sensitive to luma than chroma because it contains many more rods than cones. Consequently, most video-compression systems start by transforming RGB pictures into luma and chroma (YUV) images. To save bits in the video stream, compression schemes downsample the image's chroma portion. Thus most digital video compression schemes transform a series of YUV images into a compressed video stream while video decompression streams expand a compressed video stream into a series of YUV still images.

Because they start with a series of still images, video compression streams use many of the lossy compression techniques developed to compress still images. Lossy compression techniques identify and discard portions of an image that cannot be seen or are nearly invisible. The JPEG compression standard is the most widely used lossy, still-image compression scheme. Video streams work very well with lossy compression schemes because any image imperfections produced by the compression process appear fleetingly and are often imperceptible.

Digital video-compression algorithms slice pictures into small pixel blocks and then transform these blocks from the spatial domain into a series of coefficients in the frequency domain. The DCT (discrete cosine transform) is commonly used to transform from the spatial to the frequency domain. DCT was first widely used for JPEG still-image compression. Most video-compression schemes prior to the H.264/AVC digital-video

standard employ the DCT transform on 8x8-pixel blocks but H.264/AVC uses the more advanced, integer-based Hadamard transform on 16x16-pixel blocks.

Because the eye is more sensitive to lower-frequency image patterns, the pixel blocks' frequency-domain representations can be passed through a low-pass filter to remove higher-frequency patterns, which reduces the number of bits needed to represent that block. In addition, video-compression algorithms represent low-frequency coefficients with more precision (using more bits) while using less precision (fewer bits) to represent high-frequency coefficients.

Computing the frequency-dependent DCT coefficients takes two steps. First, the coefficients are quantized to discrete levels using perceptual weighting to limit the number of coefficient bits. The quantized coefficients are then generated using a lossless variable-length-coding (VLC) coding technique that assigns fewer bits to frequently occurring coefficient numbers, which again reduces the size of the video bitstream.

Putting Entropy to Work

VLC compression is called entropy coding. Huffman coding was the most common entropy-coding method used for video compression prior to the introduction of H.264/AVC compression. The H.264/AVC standard uses two entropy-coding methods called CAVLC (context-adaptive, variable-length coding) and CABAC (context-adaptive, binary arithmetic coding). CABAC coding uses a non-integer number of bits for VLC encoding, which improves compression by roughly 10% over CAVLC. However, CABAC requires more computation and therefore more processor bandwidth.

Still-image compression methods can deliver compression ratios of 10:1 while still producing good quality images. (Higher compression ratios deliver visibly inferior images.) Video consists of a stream of still images and video frames compressed using still-image compression techniques are called "I frames" (intra frames) or "I slices." Because each frame in the video image stream is related to the images that come before it and the images that follow, there is correlation and redundancy among successive time slices (except when there's a jump cut or sudden frame blackout).

Video-compression schemes exploit temporal inter-frame redundancy to achieve much higher compression ratios, approaching 200:1, while still delivering good quality video. In the simplest case of a video stream that represents an unchanging scene, a video-compression scheme merely needs to tell the video decoder to repeat the last image, which requires very few bits and results in very high compression. Most successive video frames differ by at least a few pixels (by many pixels if the video includes a lot of motion or there's a scene change), so video-compression schemes must use a variety of motion-dependent compression methods to accommodate different types of video.

One simple method involves subtracting one frame from a previous frame and then encoding the difference. For video streams where there's little movement, this approach can be very efficient. More advanced schemes employ a technique called motion

estimation, which breaks frames into 16x16-bit macroblocks and searches a previously encoded frame for the same macroblock. If there's a match, a motion-estimating video encoder encodes a motion vector that instructs the video decoder to reproduce a previously decoded macroblock in a new location within the frame. This scheme requires matching video decoders to save macroblocks from the previous frame for possible reuse.

A reference frame used for motion estimation need not be a previous frame. Video-compression techniques can encode frames out of temporal order (especially for recorded video that's not being encoded in real time), which means that intermediate image frames can be based both on previous and future frames. Frames based on previous frames alone are called "P" frames or slices and frames encoded using information from both previous and future frames are called "B" frames or slices. Motion estimation represents a very large search problem, which is computationally intensive and consumes billions of operations per second.

Figure 1 shows simplified block diagrams for a video-compression encoder and decoder based on the techniques just discussed. Video encoding is more complex than video decoding and a video encoder includes most of the blocks found in a corresponding decoder, which the encoder uses for motion estimation.

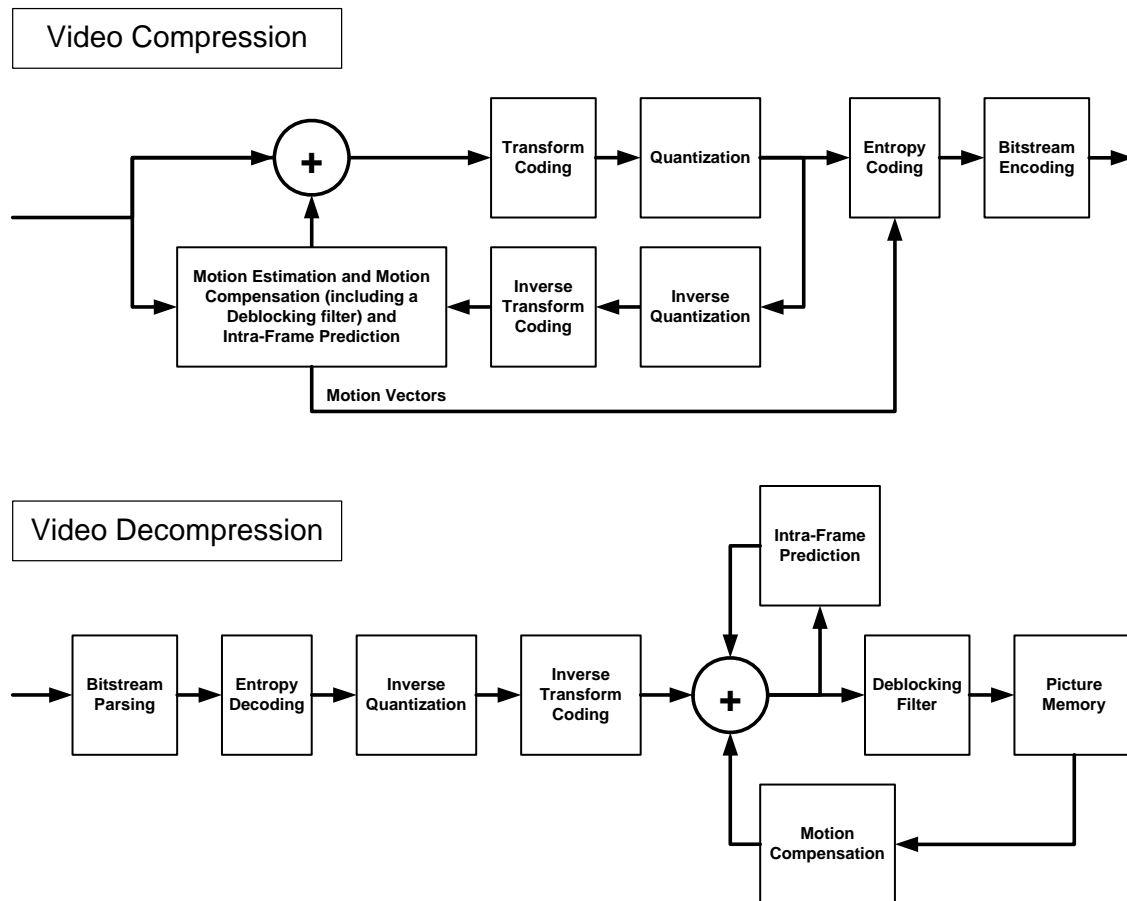


Figure 1: Simplified block diagrams for a video-compression encoder and decoder.

The two block diagrams in Figure 1 clearly show the sequential nature of the steps used for video compression and decompression. All of these steps can be performed by a microprocessor or microprocessor core, for example. However, general-purpose processors must operate at very high clock speeds to achieve the required computational bandwidth for video compression and decompression. Multi-GHz processors draw substantial amounts of power and are inappropriate for all portable and almost all other video applications. Multi-GHz processor cores for SOC designs don't yet exist. So there are tremendous benefits to finding more efficient video-codec hardware.

The block diagrams in Figure 1 also clearly show that the encoding and decoding algorithmic sequences contain substantial inherent parallelism. Frames are passed down an assembly line where separate operations are performed sequentially. A separate processor can perform the function of each box in the Figure 1 block diagrams, which would greatly reduce the required clock rate. The lower clock rate would substantially reduce operating power to levels compatible with fanless, line-powered products and portable, battery-powered devices.

A Hardware Video Codec

Figure 2 shows a block diagram of Tensilica's Diamond 388VDO Video Engine, which uses separate microprocessor cores (a Stream Processor and a Pixel Processor) and a DMA controller to exploit the parallelism inherent in video-compression and video-decompression algorithms. The Stream and Pixel Processors inside the Diamond 388VDO core split the video-compression tasks while the DMA controller moves uncompressed and compressed images into and out of the core and between the two processors. Each processor inside the Diamond 388VDO Video Engine has its own local instruction and data RAMs.

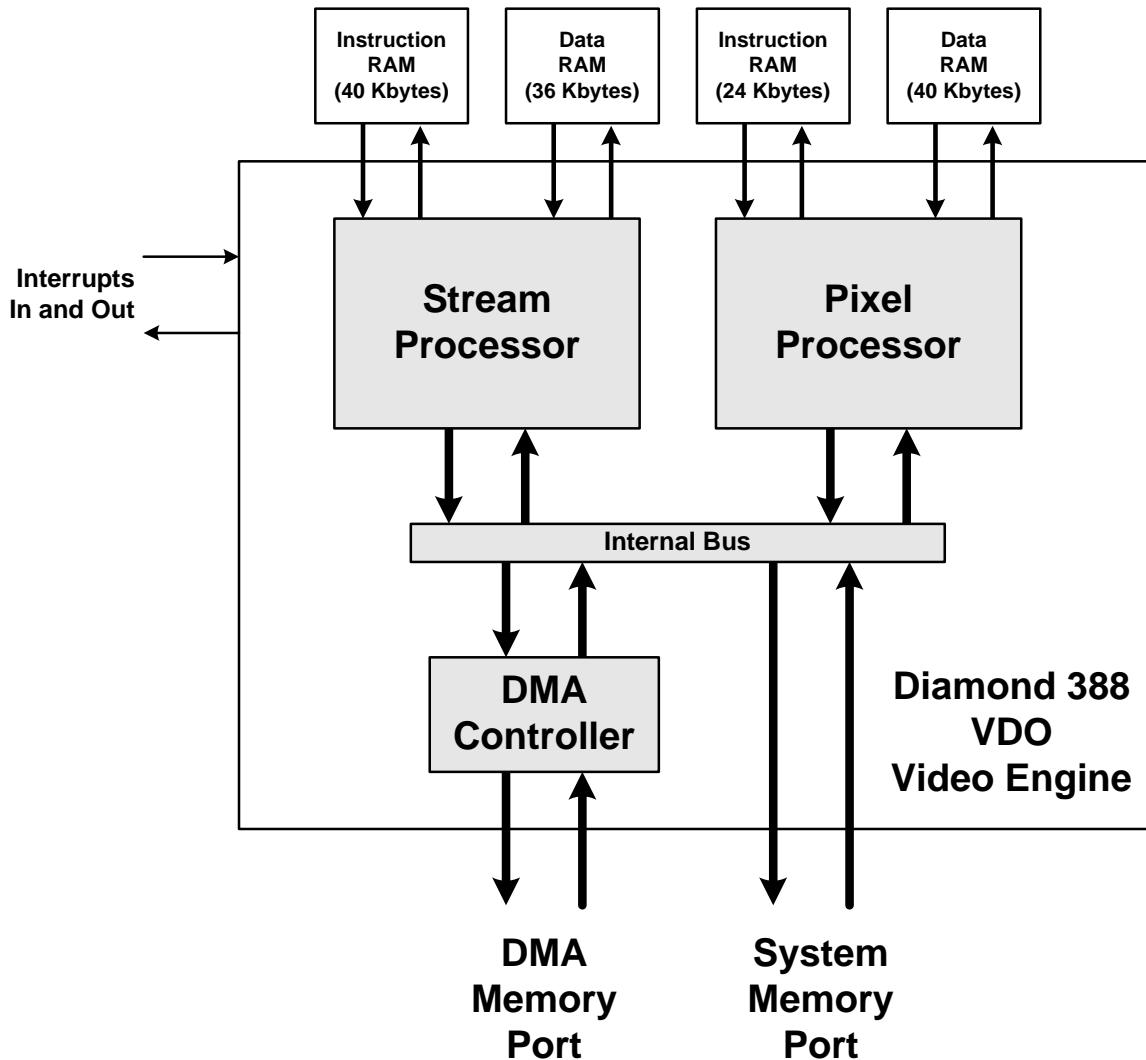


Figure 2: Block diagram of Tensilica's Diamond 388VDO Video Engine.

The Diamond Video Engine's Stream and Pixel Processors are based on Tensilica's configurable Xtensa processor architecture. The Stream Processor has been augmented with additional instructions to perform bitstream parsing and entropy coding. Some of these new instructions are based on Tensilica's FLIX (flexible-length instruction extensions) and employ a VLIW instruction format with two independent operations per instruction. The Diamond Video Engine's Pixel Processor has been augmented with SIMD (single-instruction, multiple data) instructions that perform operations on multiple pixels simultaneously.

The instructions added to both processors allow the Diamond Video Engine to encode MPEG-4 Advanced Simple Profile bitstreams and to decode H.264/AVC Main Profile, MPEG-4 Advanced Simple Profile, MPEG-2 Main Profile, and VC-1/WMV 9 Main Profile video bitstreams at standard-definition (SD or D1) display resolution and 30

frames/sec while running at clock rates below 200 MHz. Low clock rates generally mean lower power requirements and the 200-MHz clock-rate target was selected so that the Diamond Video Engine could be implemented in a generic, low-cost, 130nm IC-fabrication process.

Figure 3 shows the task allocation within the Diamond Video Engine while decoding H.264/AVC video bitstreams. The Stream Processor performs bitstream parsing (separating the Network Abstraction Layer, the Picture Layer, and the Slice Layer) and entropy decoding. The Pixel Processor performs inverse quantization, inverse transform coding, intra-frame prediction, motion compensation, and image deblocking. The Stream Processor assists the Pixel Processor with motion compensation.

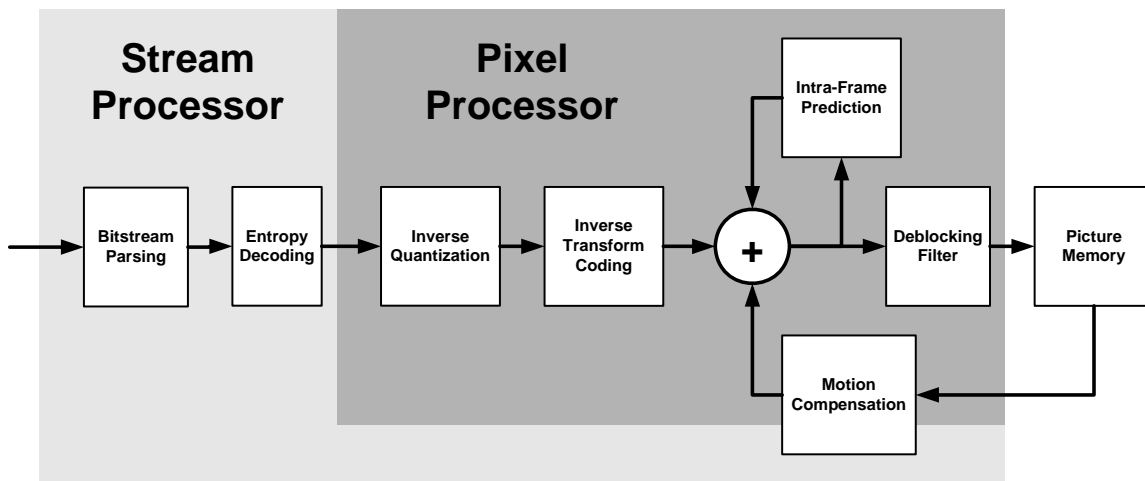


Figure 3: Task allocation within the Diamond 388VDO Video Engine for H.264/AVC decoding.

Note that it would have been possible to run all of these decoding tasks on one processor, but at a much higher clock rate that would have required a more expensive process technology. Given the number of available transistors in nanometer IC lithographies and the need to minimize power dissipation in portable, battery-powered video products, the division of labor among the two processors and the DMA controller within the Diamond Video Engine accomplishes the goal of minimizing power dissipation by keeping clock rates low even when decoding standard-definition video.

Frames In/Frames Out

Although video encoding and decoding are complex processes, the Diamond Video Engine simplifies digital video for an SOC design team by acting as a black-box hardware video codec, as shown in Figure 4. To encode video, the system's host processor first configures the Diamond Video Engine with a command and then passes unencoded video frames to the engine, which encodes the frames and passes back

encoded images (called “video decodable entities” or VDEs) to the host processor. For video decoding, the host processor first configures the Diamond Video Engine appropriately with a command and then passes VDEs to the Diamond Video Engine, which decodes the images and passes decoded frames back to the host processor.

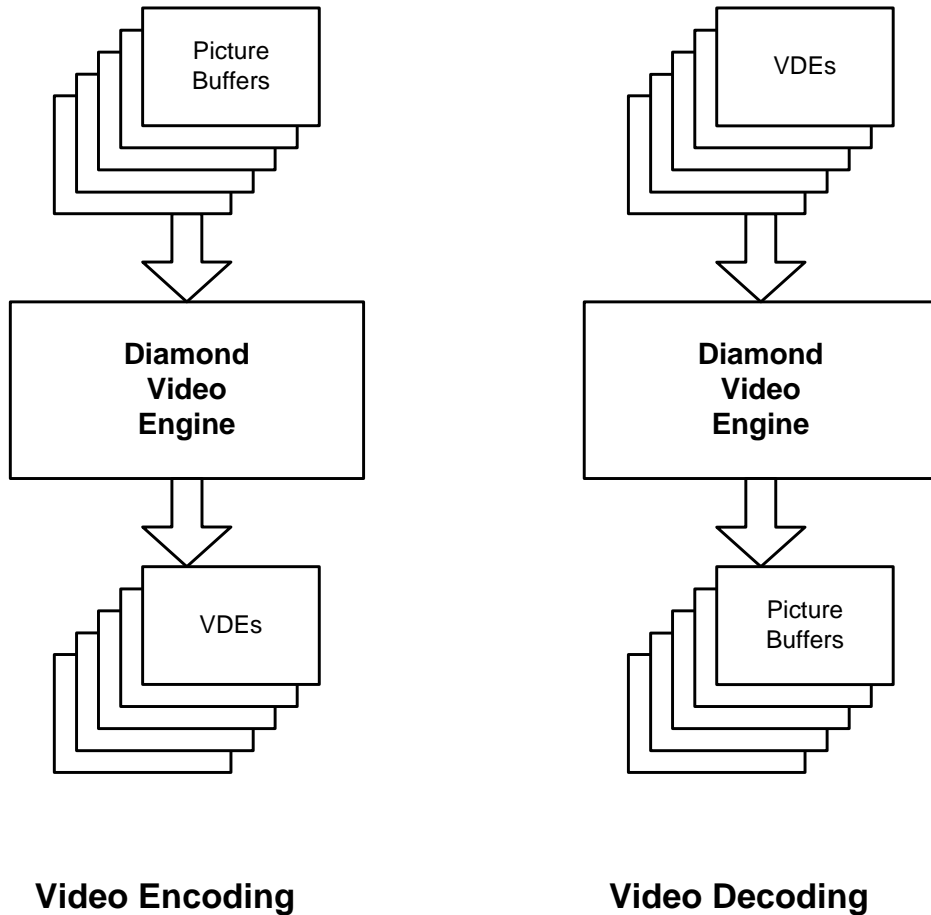


Figure 4: Passing encoded and unencoded video to and from the Diamond 388VDO Video Engine.

The host processor uses a predefined set of API calls to operate the Diamond Video Engine, as shown in Figure 5. The Diamond Video Engine’s internal operations are visible to the host processor, which is consistent with the Diamond Video Engine’s role as an SOC component. Figure 5 shows the host processor running a video application called the “System Controller Application,” which uses two queues in main system memory to send command messages and data to the Diamond Video Engine and two more queues to receive status messages and data from the Diamond Video Engine. Firmware-driven interrupts between the host processor to the Diamond Video Engine initiate the queue-based message transactions.

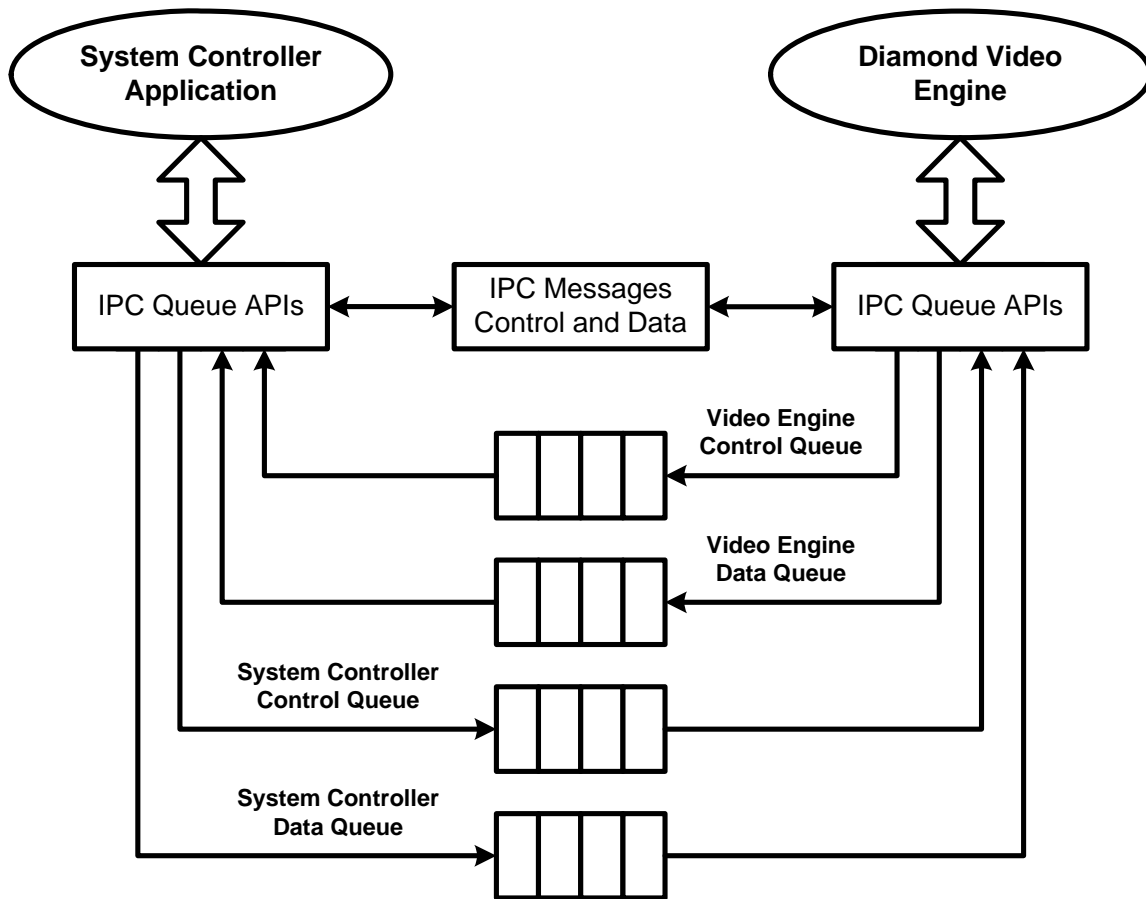


Figure 5: The Diamond 388VDO Video Engine's queue-based, message-passing API.

A simple design for a single-chip video system based on the Diamond 388VDO Video Engine appears in Figure 6. The Diamond Video Engine, like the other system components, attaches to the main system bus in this SOC design. A host System Control processor directs the SOC's operation and runs code from on-chip and off-chip memory while a Tensilica Diamond 330HiFi Audio Engine core provides digital-audio decoding. The Diamond Audio Engine, which is currently in volume production inside of mobile-phone SOCs, runs a wide and growing range of ready-to-run digital-audio codecs. (In some designs, the Diamond 330HiFi Audio Engine can also serve as the host processor.)

Both the Diamond Video Engine and the Diamond Audio Engine illustrate how processors and ready-to-run firmware can form the core of a complex, high-performance, low-power block for complex chip designs. Using the block-oriented approach to design illustrated in Figure 6, development teams can rapidly assemble extremely complicated SOCs from complex, proven IP cores and then program these SOCs with application code to produce unique products for the market.

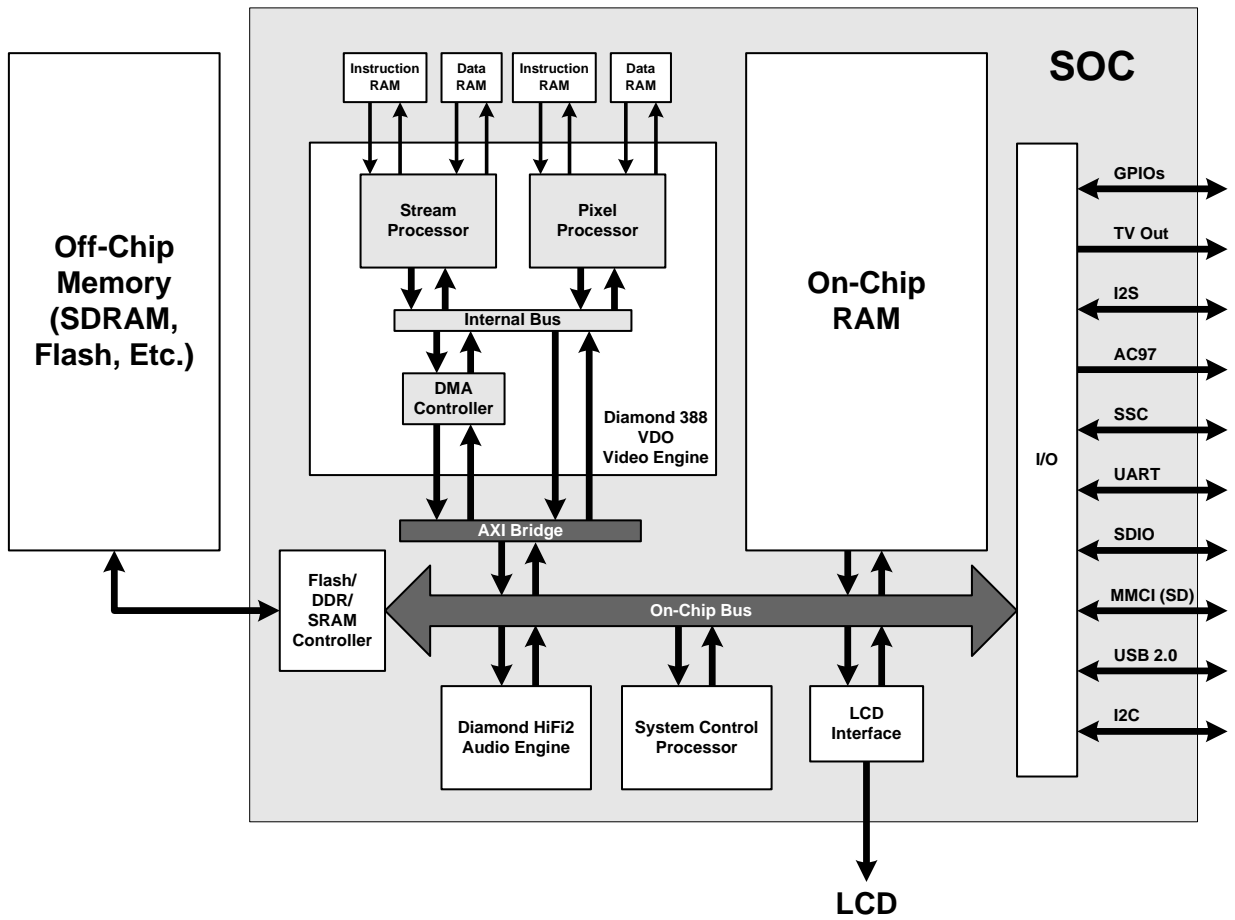


Figure 6: An Audio/Video SOC design incorporating Tensilica's Diamond 388VDO Video Engine.

An Open Invitation

If your design team is responsible for incorporating video and audio features into your next SOC design, contact Tensilica for assistance. For more information on the unique abilities and features of the Diamond 388VDO Video Engine and Diamond 330HiFi Audio Engine, see www.tensilica.com, email sales@tensilica.com, or contact Tensilica directly at:

Tensilica, Inc.
3255-6 Scott Blvd.
Santa Clara, CA 95054
Phone: (408) 986-8000
FAX: (408) 986-8919