Reed Electronics Group

MICROPROCESSOR REPORT

www.MPRonline.com

THE INSIDER'S GUIDE TO MICROPROCESSOR HARDWARE

ARC Shows SIMD Extensions

New Instructions With Macros and DMA Extend ARC 700 Processor

By Tom R. Halfhill {11/21/05-01}

Apple CEO Steve Jobs was denying he would introduce a video iPod until almost the moment he appeared on stage to demo the product. But everyone knew it was inevitable. Video is the next MP3, and any microprocessor competing for sockets in tomorrow's consumer gadgets must be able to handle audio and video as adroitly as Jobs handles the press.

Hence the rush to introduce new video processors and to modify existing CPU architectures for the growing demands of media processing. At the recent **Fall Processor Forum** in San Jose, both ARC International and Tensilica disclosed new audio/video extensions for their configurable processor cores. In the same session, German startup Videantis introduced a second-generation core for synthesizable video coprocessors. (See *MPR 11/7/05-01*, "Videantis Chases Digital Video.") And two weeks later, Royal Philips Electronics announced the first low-power mobile processor core in its 11-year-old TriMedia family—with new instructions for video.

Chief architect Nigel Topham presented ARC's new SIMD extensions at FPF. The extensions are an extra-cost option for the ARC 700 family of 32-bit processor cores, which are licensable, synthesizable, and user configurable. (See *MPR 6/21/04-01*, "ARC 700 Secrets Revealed.") Specifically, ARC offers the SIMD extensions for the ARC 710D, 725D, and 750D—three preconfigured cores in the ARC 700 family that allow designers to make additional configuration changes if needed. (See *MPR 3/14/05-02*, "ARC's Preconfigured Cores.") Although these are ARC's most powerful processors, they're suitable for mobile applications. When fabricated in a low-power 0.13-micron CMOS process, even the high-end ARC 750D consumes only about 0.13mW per megahertz, or about 69mW at its maximum worst-case clock frequency of 533MHz. That's less power than an ARM1136-series core.

When optimized for speed, the SIMD extensions will enlarge a 118,000-gate ARC 750D by about 150,000 gates (excluding memories), and by fewer gates when optimized for area. Either way, that isn't much silicon for the additional features. The extension package includes 104 new instructions for audio/video processing; configurable local memories, an independently pipelined SIMD unit, a machine-level macro instruction that initiates complex SIMD operations in parallel with regular instructions, and a smart DMA engine that can manage memory transfers for SIMD instructions in the background. In effect, the extensions turn the ARC 700 into a superscalar processor capable of sustaining two instructions per clock cycle—without significantly altering the existing uniscalar instruction pipeline, and without requiring the additional control structures of true superscalar pipelines.

For now, at least, ARC isn't licensing the SIMD extensions separately. Instead, ARC will license them as parts of larger extension packages released later this year. Those packages are the ARCvideo Subsystem and the ARCmedia Subsystem. Another licensable package in this collection is the ARCsound Subsystem, which ARC shipped last year. All these packages are part of an umbrella product called the ARC Multimedia Subsystem. Next year, ARC plans to introduce additional processor cores that will be compatible with these subsystems. ARC designed the extensions with both

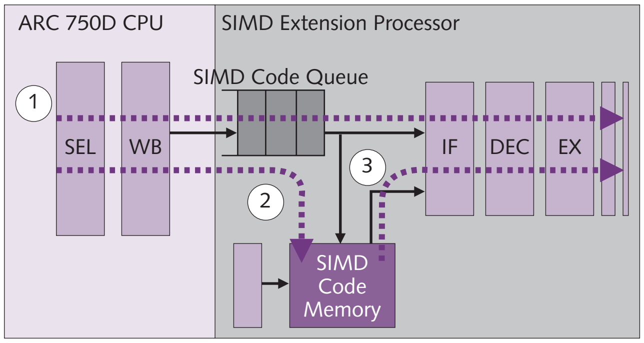


Figure 1. ARC's new SIMD instructions can execute in a closely coupled mode or a decoupled mode, depending on the program's requirements. In closely coupled mode, the ARC 700 processor issues SIMD instructions from the normal pipeline's write-back stage through a special queue to the SIMD unit. In decoupled mode, the SIMD unit fetches frequently executed sequences of SIMD instructions from local code memory dedicated for that purpose.

single- and multicore processing in mind, so we expect to see some new multicore-ready processors from ARC in 2006.

Decoupled SIMD Unit Enables Parallelism

The SIMD package extends the ARC 700 family's ARCompact instruction-set architecture (ISA) for data-level parallelism. ARCompact consists mostly of 32-bit RISC instructions, but it also has a subset of 16-bit instructions for greater code density. (See the sidebar, "ARCompact: An Elegant 16/32-Bit ISA," in *MPR 2/18/03-06*, "Soft Cores Gain Ground.") All the new SIMD instructions are 32 bits long. However, they can perform 128-bit vector operations on multiple datatypes common in audio/video processing: 4×32 bits, 8×16 bits, and 16×8 bits. When manipulating 8-bit values, the SIMD instructions can execute 16 operations per clock cycle.

To accommodate the new datatypes, the SIMD extensions add several new registers to the ARCompact ISA. All are independent of the existing general-purpose registers. ARC has allocated enough address bits for up to 64 new vector registers, each 128 bits wide, but this implementation has only 24 vector registers (vr00–vr23), which ARC deems sufficient for the target applications. In addition, the SIMD extensions add eight 16-bit-wide registers (i0–i7) and a 320-bit-wide vector accumulator. The 16-bit registers are for scalar datatypes and memory addresses, and they actually occupy the same physical space as the first vector register (vr00). The vector accumulator can hold eight values, each 32 bits or 40 bits long. Depending on the type of instruction, the vector accumulator may hold a third operand and/or the result of a SIMD operation.

ARC's SIMD extensions may not seem much different from those in many other CPU architectures. However, ARC has a few twists. One is the way the SIMD instructions execute in the context of the ARC 700's existing scalar pipeline. For obvious reasons, ARC didn't want to significantly redesign the pipeline of a clean-slate microarchitecture introduced only last year. (See *MPR 3/8/04-01*, "ARC 700 Aims Higher.") Nor would executing SIMD instructions in the same pipeline as other instructions be the best way to achieve parallelism. So ARC provides two execution modes for SIMD instructions: closely coupled and decoupled.

In closely coupled mode, programs can freely mix SIMD instructions with other ARCompact instructions in the same instruction stream, which is closely managed by the processor. During the write-back stage of the scalar pipeline, the processor diverts SIMD instructions into a special code queue, which can be up to 256 slots deep. When the SIMD unit signals it's ready to accept an instruction, the processor dispatches the next SIMD instruction in order from the queue. This closely coupled mode is useful for SIMD code that doesn't recur too often. Figure 1 shows a diagram of the SIMD execution modes.

Decoupled execution mode is better for frequently repeated sequences of SIMD instructions—the types of sequences often encountered in software codecs performing repetitive operations on streams of audio/video data. In decoupled mode, the program stores a SIMD instruction sequence in a configurable-size SIMD code memory and logs the address in a general-purpose register. A new instruction called VRUN initiates execution at that address by handing off the operation to the SIMD unit. In effect, the SIMD code sequence is a macro, and the region of SIMD code memory storing the macro behaves like a locked instruction cache. Decoupled execution mode allows the SIMD unit to independently fetch and execute code sequences without burdening the ARC 700's regular pipeline, which may continue executing scalar instructions in parallel with the SIMD unit. Figure 2 shows snippets of code written differently for the closely coupled and decoupled SIMD execution modes.

SIMD Unit Is Almost a Coprocessor

The SIMD unit's ability to run code from local memory independently of the ARC 700's main pipeline shows that ARC has created the near equivalent of a vector coprocessor. In decoupled mode, the SIMD unit can execute instructions in parallel with the main pipeline and sometimes out of program order. Yet it doesn't require the extra overhead of dual-issue superscalar and out-of-order control structures, which would inflate the size of the processor and consume more power. With target audio/video applications, the SIMD unit can run autonomously much of the time, scarcely interfering with normal scalar execution.

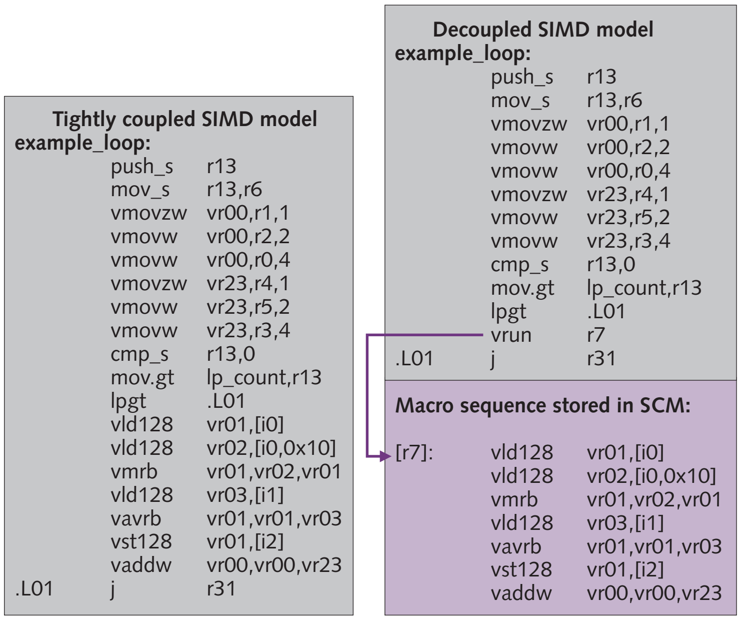
Decoupling the SIMD unit from the main pipeline may appear to make the processor less integrated than it should be. Note that SIMD instructions don't enter the SIMD queue or SIMD code memory until leaving the final write-back stage of the ARC 700's seven-stage main pipeline. A better-integrated processor might divert SIMD instructions into their separate pipeline much sooner, perhaps at the decode stage (stage 3 in the ARC 700). Instead, SIMD instructions must traverse the entire length of the main pipeline before starting their detour through the SIMD queue or SIMD code memory and only then begin their journey through the SIMD pipeline, as Figure 3 shows.

Once the vector instructions reach the SIMD unit, they find themselves in a new world. The SIMD unit has its own eight-stage pipeline with 128-bit datapaths—much wider than the conventional 32-bit datapaths of the ARC 700's main pipeline. (A user-configuration option can widen the SIMD accumulator's datapath to 160 bits, allowing rapid transfers of four 40-bit values, instead of the four 32-bit values permitted by the default 128-bit datapath.) As described above, the SIMD unit also has its own register files for vector and scalar data. In addition, the new vector load/store instructions must always access dedicated local memories for SIMD instructions and data, not main memory directly or the ARC 700's instruction and data caches.

In other words, the SIMD unit is virtually a coprocessor bolted onto the back door of the ARC 700. It's not just another function unit among equals. The ARC 700's main pipeline is essentially a dark tunnel to the SIMD unit. In contrast, ARC's XY Advanced DSP Extensions—a long-available option for the ARC 700—hook into the main pipeline at stage 2 (the alignment stage) and add a separate memory-access pipeline in parallel with the main pipeline.

However, there are good reasons why ARC attached the SIMD unit to the tail of the main pipeline. First, it decouples performance as well as execution. The SIMD unit can run at the same clock frequency as the rest of the ARC 700 or at a different frequency, depending on the application's performance and power requirements. And the ARC 700 can run at its maximum clock frequency with or without the SIMD extensions. This isn't true of the XY Advanced DSP Extensions, which tend to slow the processor.

Another reason for arranging the pipelines end-to-end is that the ARC 700 is the first ARC processor with a precise exception model. To support sophisticated virtual-memory operating systems, the processor must be able to resume executing load/store instructions after recovering from page faults. The processor cannot change its architectural state until instructions finish executing and are committed in the final write-back stage. Appending the SIMD pipeline after writeback


Tightly coupled SIMD model

```
example_loop:
        push_s   r13
        mov_s    r13,r6
        vmovzw   vr00,r1,1
        vmovw    vr00,r2,2
        vmovw    vr00,r0,4
        vmovzw   vr23,r4,1
        vmovw    vr23,r5,2
        vmovw    vr23,r3,4
        cmp_s    r13,0
        mov.gt   lp_count,r13
        lpgt     .L01
        vld128   vr01,[i0]
        vld128   vr02,[i0,0x10]
        vmrb     vr01,vr02,vr01
        vld128   vr03,[i1]
        vavrb    vr01,vr01,vr03
        vst128   vr01,[i2]
        vaddw    vr00,vr00,vr23
.L01    j        r31
```

Decoupled SIMD model

```
example_loop:
        push_s   r13
        mov_s    r13,r6
        vmovzw   vr00,r1,1
        vmovw    vr00,r2,2
        vmovw    vr00,r0,4
        vmovzw   vr23,r4,1
        vmovw    vr23,r5,2
        vmovw    vr23,r3,4
        cmp_s    r13,0
        mov.gt   lp_count,r13
        lpgt     .L01
        vrun     r7
.L01    j        r31
```

Macro sequence stored in SCM:

```
[r7]:   vld128   vr01,[i0]
        vld128   vr02,[i0,0x10]
        vmrb     vr01,vr02,vr01
        vld128   vr03,[i1]
        vavrb    vr01,vr01,vr03
        vst128   vr01,[i2]
        vaddw    vr00,vr00,vr23
```

Figure 2. These two examples show how programmers can write the same code for closely coupled SIMD execution or decoupled SIMD execution. At left, the closely coupled code executes a loop of mixed scalar and SIMD instructions. At right, the decoupled code uses the new VRUN instruction—a nonblocking operation—to run a macro of frequently executed vector code previously stored in SIMD code memory. The SIMD unit can autonomously execute the macro while regular code executes in the scalar pipeline.

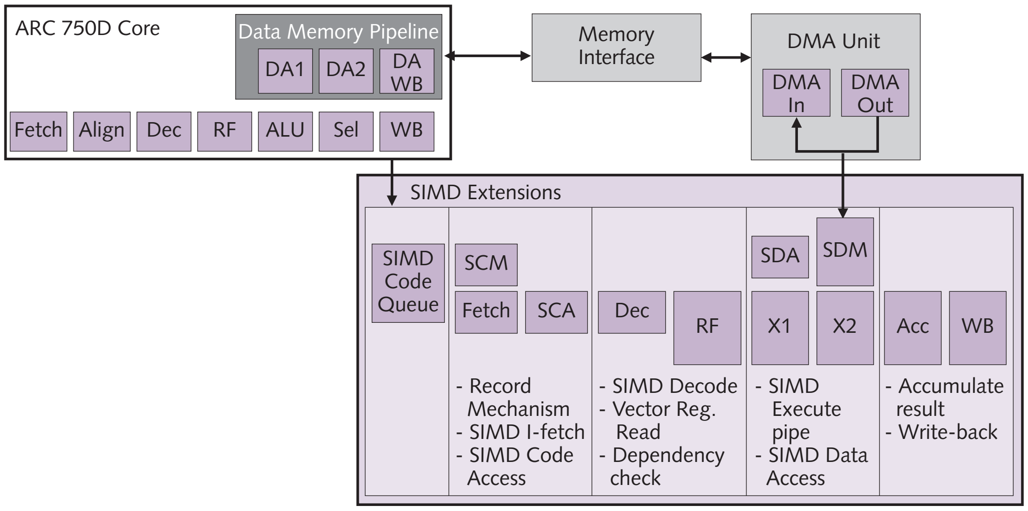


Figure 3. ARC's SIMD unit branches from the ARC 700's main pipeline after the final write-back stage, then appends a new eight-stage pipeline for vector instructions. The SIMD unit always fetches vector instructions from the SIMD code queue or SIMD code memory, and it always fetches vector data from SIMD data memory. The DMA unit handles all memory transfers between SIMD memory and main memory.

Instruction	Description	Notes
Data-Movement Instructions		
vld32	32-bit vector load	
vld64	64-bit vector load	
vld64w	64-bit vector load, 16-bit elements	Sign extend
vld32wl	32-bit vector load, 16-bit elements	Upper 4 elements
vld32wh	32-bit vector load, 16-bit elements	Lower 4 elements
vld128	128-bit vector load, SIMD addr reg	
vld128r	128-bit vector load, scalar addr reg	
vst16	16-bit store from vector register	
vst32	32-bit store from vector register	
vst64	64-bit store from vector register	
vst128	128-bit store from vector register	SIMD addr reg
vst128r	128-bit store from vector register	Scalar addr reg
vmvw	Vector move (conditional)	
vmvzw	Vector move (conditional)	Zero unwanted
vmovw	Copy GPR to vector elements	
mmovzw	Copy GPR to vector elements	Zero unwanted
vmvaw	Vector move (conditional)	Accumulate
vmovaw	Copy GPR to vector elements	Accumulate
Logic Instructions		
vand	Vector logical AND	
vor	Vector logical OR	
vxor	Vector logical XOR	
vbic	Vector bit-clear	
vandaw	Vector logical AND	Accumulate
vxoraw	Vector logical XOR	Accumulate
vbicaw	Vector bit-clear	Accumulate
Arithmetic Instructions		
vaddw	Vector add	
vsubw	Vector subtract	
vsummw	Vector masked summation	
vaddsuw	Vector add/sub	Butterfly
vavb	Vector byte average	
vavrb	Vector byte average with rounding	
vmulw	Vector multiply	
vmulfw	Vector multiply fractional	
vmaxw	Vector maximum	
vminw	Vector minimum	
vabsw	Vector absolute value	
vdifw	Vector absolute difference	
vsignw	Vector sign	
vbaddw	Vector-scalar add	
vbsubw	Vector-scalar subtract	
vbrsubw	Vector-scalar reverse-subtract	
vbmulw	Vector-scalar multiply	
vbmulfw	Vector-scalar multiply fractional	
vbmaxw	Vector-scalar maximum	
vbminw	Vector-scalar minimum	
vaddaw	Vector add	Accumulate
vsubaw	Vector subtract	Accumulate
vmulaw	Vector multiply	Accumulate
vmulfaw	Vector multiply fractional	Accumulate
vbmulaw	Vector-scalar multiply	Accumulate
vmaxaw	Vector maximum	Accumulate
vminaw	Vector minimum	Accumulate
vabsaw	Vector absolute value	Accumulate
vdifaw	Vector absolute difference	Accumulate
Compare Instructions		
veqw	Vector compare for equality	Boolean result vectors for conditional vector instructions
vnew	Vector compare for inequality	
vlew	Vector compare less-than or equal	
vltw	Vector compare less-than	
Permute / Align / Scale Instructions		
vexch1	Vector exchange, single elements	
vexch2	Vector exchange, paired elements	
vexch4	Vector exchange, quad elements	
vmr1w	Vector merge, shift right 1 element	
vmr2w	Vector merge, shift right 2 elements	
vmr3w	Vector merge, shift right 3 elements	
vmr4w	Vector merge, shift right 4 elements	
vmr5w	Vector merge, shift right 5 elements	
vmr6w	Vector merge, shift right 6 elements	
vmr7w	Vector merge, shift right 7 elements	
vsr8	Byte shift right by register distance	
vmrb	Vector merge and byte shift right	
vmr1aw	Vector merge, shift right 1 element	Accumulate
vmr2aw	Vector merge, shift right 2 elements	Accumulate
vmr3aw	Vector merge, shift right 3 elements	Accumulate
vmr4aw	Vector merge, shift right 4 elements	Accumulate
vmr5aw	Vector merge, shift right 5 elements	Accumulate
vmr6aw	Vector merge, shift right 6 elements	Accumulate
vmr7aw	Vector merge, shift right 7 elements	Accumulate
vsr8aw	Vector byte shift, accumulate elements	
vasrw	Vector arithmetic right shift	
vasrrw	Rounding arithmetic right shift	Simple rounding
vasrsrw	Rounding arithmetic right shift	Symmetric
Pack / Unpack Instructions		
vasrpwb	Pack with unsigned saturation	
vasrrpwb	Pack with unsigned saturation	Simple rounding
vupbw	Unpack bytes unsigned	
vupsbw	Unpack bytes signed	
vupbaw	Unpack bytes unsigned	Accumulate
vupsbaw	Unpack bytes signed	Accumulate
Miscellaneous & Special Instructions		
vnop	Vector no-op	
vint	Interrupt scalar processor	
vd6tapf	Dual 6-tap filter for H.264	
vh264ft	H.264 filter test	
vh264f	H.264 filter operation	Uses filter test
vvc1ft	VC-1 filter test	
vvc1f	VC-1 filter test operation	Uses filter test
Macros & DMA Instructions		
vrec	Start recording SIMD macro	
vendrec	End macro recording	
vrun	Run macro sequence	
vrecrun	Record and run macro sequence	
vdirun	Run DMA input operation	
vdorun	Run DMA output operaion	
vdiwr	Write DMA input channel register	
vdowr	Write DMA output channel register	
vdird	Read DMA input channel register	
vdord	Read DMA output channel register	

Table 1. ARC's SIMD extensions add 104 new instructions to the ARCompact ISA. Most of these instructions operate directly on the new 128-bit vector registers; others use the new 16-bit scalar registers. The SIMD instruction set is designed to accelerate popular video algorithms and codecs at high frame rates and resolutions up to D1 (720 × 480 pixels NTSC, 720 × 576 pixels PAL and SECAM).

guarantees that all instructions reaching the SIMD code queue will commit. This simplifies the design of both pipelines.

A third reason for decoupling the pipelines is that either one can stall without necessarily affecting the other. Unless there are mutual instruction dependencies, one pipeline can keep running while the other recovers. This is possible even if an instruction destined for the SIMD pipeline needs operands from the processor's core registers, because the instruction carries those operands into the SIMD code queue or SIMD code memory. ARC says it would decouple the SIMD unit from the ARC 700 this way, even if it were starting the whole design from scratch.

We can think of yet another advantage of designing the SIMD unit as a virtual coprocessor: it's more portable to other implementations. We suspect it will appear in future ARC processor cores and operate in much the same way it does now.

New Instructions Optimized for A/V

With 104 new instructions, the SIMD extensions represent a major addition to the ARCompact ISA. The instruction set adheres to a classic RISC load/store architecture, separating arithmetic and logical operations from memory operations. Most instructions perform fixed-point or fractional vector operations on multiple elements of 8-, 16-, or 32-bit data. The load/store instructions can read or write 128 bits of data from or to SIMD data memory in a single clock cycle. The DMA controller handles all transfers between local data memory and external memory.

There's a great deal of flexibility in this instruction set. Some load instructions can read unaligned data, though it's often unnecessary, because the DMA controller aligns most data automatically. Some load instructions can unpack data by extending the elements, which prepares the data for later processing at higher levels of precision. For instance, the new vld64w instruction loads an eight-byte vector array into a 128-bit vector register by extending each byte-size element into a 16-bit value. Similar instructions can "unpack" a single scalar value by replicating it as multiple elements in a vector register, a transformation known as scalar-to-vector broadcasting. By applying bit masks, many of these load/store instructions can select or deselect individual elements of a vector array before moving the data. Deselected elements may be cleared or left unchanged.

Arithmetic instructions are equally powerful. One instruction (vaddsuw) can perform 16 operations with 16-bit precision to calculate a two-point vectorized butterfly, which is useful for discrete cosine transforms (DCT) and inverse DCTs (iDCT). To support these kinds of vector-multiplication instructions, the SIMD unit has eight 16- × 16-bit multipliers capable of handling 16-bit integer or fractional data. The 16-bit fractional datatype isn't an IEEE 754 floating-point datatype; it uses the 1q15 format, with one sign bit and 15 mantissa (significand) bits. The multipliers can automatically round results from fractional operations.

ARC created many of these new instructions with specific video algorithms in mind. For example, byte-averaging instructions are valuable for interpolating half- and quarter-pixel values when running the motion-compensation routines commonly found in video codecs. Two of these instructions (vavb and vavrb) can perform 16 byte-averaging calculations per clock cycle, allowing the SIMD unit to interpolate a 16- × 16-pixel macroblock in only 16 cycles. Table 1 lists all 104 new instructions with brief descriptions.

In another example of clever architecture definition, ARC created three variants of a vector-exchange instruction (vexch1, vexch2, vexch3) specifically for transposing blocks of data arranged in two-dimensional arrays, as often found in video streams. This instruction can rapidly transpose a 4 × 4 or 8 × 8 matrix using only two vector registers. Transposing a 4 × 4 matrix requires only four iterations of this instruction; transposing four 4 × 4 matrices requires only eight iterations; and transposing an 8 × 8 matrix requires only twelve iterations. Performing these kinds of transformations without special instructions or registers would require dozens, perhaps hundreds, of conventional RISC instructions.

H.264 and VC-1 codecs are gaining popularity, so ARC has defined some new instructions to accelerate their underlying operations. Two examples are vh264ft (which applies a horizontal deblocking filter test) and vh264f (which applies the filter values to block boundary pixels). ARC says these instructions exploit more pixel-level parallelism and improve performance 15–25 times over an unmodified ARC 700. In addition to all these examples of powerful vector instructions, ARC has defined new scalar extensions to accelerate entropy decoding of compressed bitstreams. Table 2 summarizes the performance of ARC's extensions.

DMA Controller Optimized for Multimedia

The DMA controller that manages all transfers between main memory and SIMD data memory differs in some respects from similar controllers in other embedded processors. It's designed specifically for the requirements of multimedia datatypes, which are often arranged in small blocks or arrays. For instance, H.264 video frames appear in memory as two-dimensional arrays subdivided into macroblocks of pixels.

To reduce the manual labor normally required for maintaining memory-address pointers, the DMA controller accepts simple commands for defining the boundaries of a frame and for shifting the frame's X and Y locations. Other commands can define blocks of memory that exist within a frame or extend beyond the frame's boundaries. The DMA controller can automatically fill or clip regions of blocks that don't coincide with the frame. These kinds of operations are common when manipulating the reference frames in H.264 and VC-1 video streams, and they normally create lots of work for programmers. To save even more time, the DMA controller explicitly supports block transfers for both interlaced and progressive-scan video.

By offloading data transfers onto the smart DMA controller, the SIMD subsystem seems even more like an independent coprocessor. It can continuously stream audio or video from main memory into SIMD data memory, perform vector operations on the data, and stream results back into main memory—all in parallel, with little attention from the rest of the ARC 700. Routing the audio/video data through SIMD data memory avoids polluting the ARC 700's data cache, and the vector instructions bypass the normal instruction cache, too. The DMA controller even handles some messy details of main-memory management, such as automatically compensating for DRAM page boundaries when manipulating blocks of data.

At times, the ARC 700—nominally a uniscalar processor—achieves three-way parallelism with these extensions. It can simultaneously execute control code in its main pipeline, execute SIMD code in its vector pipeline, and transfer data to and from memory in the background.

Doing another favor for programmers, ARC provides code libraries for popular audio, video, and image codecs: H.264 decoding (baseline and main profiles); VC-1 decoding (baseline and main profiles); MPEG-4 decoding (simple profile to advanced simple profile, layer 4); MPEG-2 decoding (main profile at main level); JPEG encoding; Motion JPEG (M-JPEG) encoding; AAC encoding; and MP3 encoding (up to 12 channels). Because the ARC 700 has an MMU, it can run these codecs on sophisticated operating systems like Linux or as standalone programs in deeply embedded applications.

Computation	SIMD Clocks	Block	Clocks / Pixel-Op	SIMD Speedup
MPEG-1, MPEG2, MPEG-4				
Forward DCT 8x8	113	8x8	1.77	9.4x
Inverse DCT 8x8	129	8x8	2.02	10.4x
Matrix Transpose 8x8	12	8x8	0.19	12.5x
Half-Pel Interpolate 16x16	83	16x16	1.3	9.8x
MPEG-4				
Vertical Low-Pass Filter	453	16x16	7.08	12.9x
Horizontal Low-Pass Filter	516	16x16	8.06	9.8x
H.264				
Integer Transform	82	4x4	5.13	10.3x
Inter-Prediction				
Vertical Filtering	71	8x8	1.11	25.2x
Horizontal Filtering	116	8x8	1.81	15.4x
Quarter-Pel Interpolate	46	8x8	0.72	8.3x
Intra-Prediction (Planar)	240	16x16	0.94	7.7x
Deblocking Filter				
Horizontal Filtering	2	6	0.33	70x
Vertical Filtering	54	24	2.25	10.4x
VC-1				
Integer Transform	143	8x8	2.23	10.7x
Bilinear Block Filter	93	8x8	1.45	11x
Bicubic 2D Filter	216	8x8	3.38	11x

Table 2. ARC measured the performance of its new SIMD extensions by synthesizing an ARC 750D processor core in the Virtex-4 FPGA of an ARCangel-4 development system. Cycle counts include load/store operations from local memory, as well as SIMD computations. ARC's SIMD extensions not only speed up common media operations but also save power by allowing the processor to run at a lower clock frequency for a given level of performance.

ARC Extensions Face Growing Competition

There's a video stampede. In the past year, ARC, ARM, Silicon Hive, and Tensilica have all announced or introduced video extensions for their licensable processor cores. (See *MPR 6/20/05-01*, "Busy Bees at Silicon Hive.") Startups such as Elixent and Videantis are elbowing into the crowd with their configurable processor cores. (See *MPR 6/27/05-02*, "Elixent Improves D-Fabrix.")

On November 1, MIPS Technologies announced that Sarnoff Corp. will provide synthesizable accelerator cores and software codecs for H.264 and MPEG-4 video on MIPS processors. On November 8, Philips announced the TM3270, the first low-power TriMedia processor core, which has new video extensions. And on November 16, Imagination Technologies introduced the PowerVR MSVDX video-decoder core, which supports the latest video codecs at high-definition resolutions up to 1080i/1080p and 2048 × 1024 pixels. The options for licensing digital-video intellectual property (IP) keep multiplying.

The two gorillas in the room are MIPS and ARM. MIPS processors are very popular in set-top boxes and other home-video appliances, but they are less common in mobile-video products, a fast-growing market. At Spring Processor Forum, MIPS introduced the four-member 24KE family of 32-bit processor cores, which add DSP extensions to existing 24K cores. Although the 24KE DSP extensions are adequate for low-end digital video, they are mainly intended for digital audio and speech coding, so they aren't as video-specific as the extensions from ARC, Tensilica, Philips, and some other competitors. (See *MPR 5/31/05-01*, "The MIPS 24KE Family.")

That's why Sarnoff's deal with MIPS is timely. Sarnoff's licensable Silicon IP cores combine programmability with hard-wired acceleration and are fully synthesizable. Depending on the levels of performance and functionality required, the cores range in size from tens of thousands to hundreds of thousands of gates, similar to competing solutions. Sarnoff's cores won't be as tightly integrated with the processor as are the ISA extensions from ARC, ARM, Philips, Tensilica, and others, but they should provide competitive performance.

ARM is traditionally the lower-power alternative to MIPS. Until recently, ARM processors lacked enough horsepower for the most demanding multimedia applications. In the past 18 months, ARM has taken three big steps toward high performance: OptimoDE, Neon, and Cortex-A8. OptimoDE is a highly customizable coprocessor core with impressive DSP and media-processing capabilities. (See *MPR 6/7/04-01*, "ARM's

Configurable OptimoDE.") The Neon Advanced SIMD Extensions include new vector and DSP instructions plus registers for the latest ARMv7 architecture. Cortex-A8 (code-named Tiger) is ARM's first superscalar processor core and the first processor to incorporate Neon. (See *MPR 10/25/05-02*, "Cortex-A8: High Speed, Low Power," which includes our analysis of Neon.)

Cortex-A8 and Neon dramatically alter the competitive landscape. They are a bold challenge to MIPS, which has been shipping high-performance superscalar processor cores for years but isn't the first choice for designs needing low power consumption. If ARM's estimates are close to accurate, Cortex-A8 will compete strongly with the best MIPS processors while holding down power consumption to levels more appropriate for mobile applications. Neither ARC nor Tensilica has a baseline processor as powerful as Cortex-A8, although EEMBC benchmarks show that well-crafted extensions can boost throughput more efficiently than superscalar pipelines and high clock speeds do.

ARC vs. ARM and Tensilica

Neon is a major upgrade to the ARM architecture, improving performance two to four times over the existing ARMv6 SIMD extensions. In some ways, Neon resembles ARC's SIMD extensions. Both architectures append the SIMD unit to the processor's main pipeline and allow decoupled SIMD execution. Both SIMD units have their own vector register files and instruction queues. Both have deep pipelines: ten stages for Neon, eight stages for ARC.

However, there are vital differences. Neon is less specifically targeted at video and is more suitable for other embedded applications. Although Neon's vector datapaths are only 64 bits wide, not 128 bits wide like ARC's, Neon can manipulate operands as large as its datapaths, whereas ARC's largest operands are limited to 32 bits. For media processing, the inability to handle 64-bit operands isn't a disadvantage for ARC. Likewise, Neon supports 32-bit single-precision floating-point operations, whereas ARC's extensions are limited to 16-bit 1q15 fractional data. Again, that's not a problem in audio/video processing. For developers needing true floating-point math, ARC offers other extensions. (See *MPR 5/23/05-02*, "Float Without Bloat.")

ARC's SIMD unit appears to be capable of operating more autonomously than ARM's. It has its own code and data memories in addition to an instruction queue, plus the ability to run macro routines without bothering the processor. Neon has instruction and data queues, plus a store buffer, but Neon also uses the processor's L1 and L2 data caches, which could conflict with other operations and cause thrashing. Audio/video datastreams have relatively little temporal locality, because the processor uses the data only once. Streaming the data through conventional caches may replace other data the processor needs to retain, unless programmers carefully lock and manage the caches. In contrast, ARC's SIMD unit bypasses the caches in favor of dedicated code and data memories. In addition, ARC's dedicated DMA controller can natively transfer blocks of media data to those memories in parallel with other operations.

Early performance estimates from ARC and ARM give ARC a significant advantage in power consumption. In a next-generation 65nm fabrication process, Cortex-A8 will consume 300mW at 600MHz, including Neon and caches. In a generic 90nm process, an ARC 750D with SIMD extensions and memories requires about 80mW at the maximum worst-case clock frequency of 533MHz. Both processors are capable of executing two instructions per cycle with target applications. Unfortunately, the throughput performance estimates from ARC and ARM aren't directly comparable, so we can't judge the relative value of their SIMD extensions in those terms. *MPR* encourages both companies to publish benchmark scores using EEMBC's Digital Entertainment suite. (See *MPR 2/22/05-01*, "EEMBC Expands Benchmarks.")

Tensilica's presentation at FPF was a technology preview, not a product announcement. Nevertheless, Tensilica was generous with details. Aiming for high performance, Tensilica previewed a heterogeneous dual-core audio/video processor based on differently configured Xtensa LX cores. (See *MPR 5/31/04-01*, "Tensilica Tackles Bottlenecks.") One core is a stream processor; the other is a pixel processor. Both make extensive use of custom instructions—more than 200—and share a five-channel DMA controller. Total size is nearly 300,000 gates. Even with two cores, Tensilica's video processor isn't much larger than an ARC 750D with SIMD extensions, shown in Figure 4. *MPR* will analyze Tensilica's video processor in more depth soon.

Price & Availability

Currently, ARC doesn't license the new SIMD extensions separately. Instead, the extensions are part of the ARCmedia and ARCvideo Subsystems, which are larger extension packages for the ARC 710D, 725D, and 750D preconfigured processor cores. Another package is the ARCsound Subsystem. In turn, all these packages are part of the ARC Multimedia Subsystem. ARC plans to ship the ARCmedia and ARCvideo Subsystems later this year.

The new SIMD extensions are fully synthesizable Verilog models and include software libraries for popular multimedia codecs: H.264 decoding (baseline and main profiles), VC-1 decoding (baseline and main profiles), MPEG-4 decoding (simple profile to advanced simple profile, layer 4), MPEG-2 decoding (main profile at main level, MP@ML), JPEG encoding, M-JPEG encoding, AAC encoding, and MP3 encoding.

Licensing fees and terms are negotiable and not publicly disclosed. For more information about ARC's subsystems, visit *www.arc.com/subsystems/*.

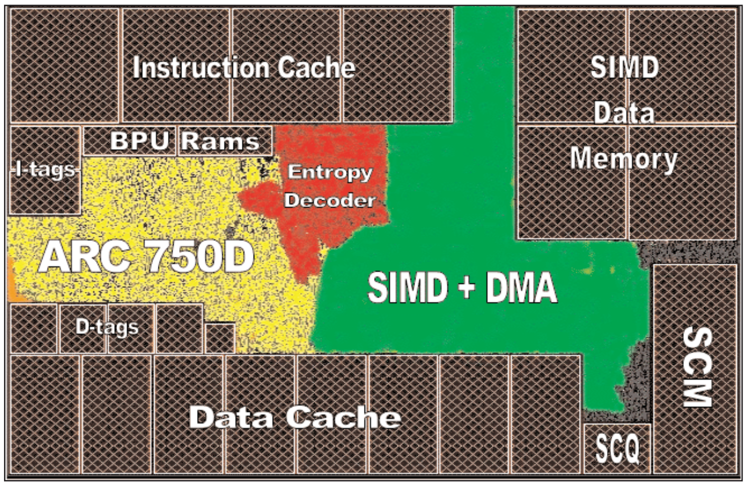


Figure 4. This synthesized floorplan shows an ARC 750D processor core enhanced with the ARCmedia Subsystem, which includes the new SIMD extensions. This configuration has 32KB instruction and data caches, 10KB SIMD code memory, 32KB SIMD data memory, the media-aware DMA controller, and entropy-decoding extensions for H.264, VC-1, MPEG-2, and MPEG-4. Excluding memories and other components of the ARCmedia Subsystem, an ARC 750D core with SIMD extensions requires about 268,000 gates. *MPR* estimates that the full ARCmedia configuration shown here is appreciably larger, perhaps 400,000 gates. However, as is often the case, the memory requires more silicon than the logic does. ARC says this floorplan occupies 9mm^2 in TSMC's 0.13-micron LVLK-OD process when synthesized with Virage physical-IP libraries. Worst-case clock frequency is an impressive 500MHz.

With HDTV and digital broadcasting finally catching on, and the U.S. government moving closer to mandating the obsolescence of broadcast analog TV, the consumer-electronics industry is on the verge of a tremendous sales boom. Plunging prices of large flat-panel displays are another incentive for consumers to replace their old TVs. ARC would love to gain ground against MIPS in home video, but ARC's new extensions are more suitable for mobile video than for higher-resolution HDTV—unless, perhaps, ARC combines the extensions with a more powerful processor or a multicore processor. Meanwhile, ARM's Cortex-A8 is promising higher performance, too. Luckily, the home video market is big enough for more than one winner.

In mobile video, there are two important applications: cellphones and everything else. ARM has a tight grip on the phones. Cortex-A8 and Neon are necessary to hold that grip, because next-generation do-it-all wireless handsets need more processing power than ARM's older cores can realistically deliver. ARC's processors and media extensions can meet the performance requirements—even beating ARM in some respects, such as power consumption—but breaking ARM's hold will be difficult. ARC stands a better chance competing for design wins in mobile video products not currently dominated by ARM. Ideally, ARC will find a market niche it can dominate in the same way ARM has muscled its way into cellphones and Apple's iPod. ◇