

Low-power MCUs and the MPEG-4 challenge

By Dr. Øyvind Strøm

Small form factors often take on big jobs, but designers can't just toss big processors into the fray – there's not enough power or cooling in some applications. New, more efficient processor architectures tailored for specific algorithm types are emerging, such as an MCU architecture described here that's designed for algorithms like MPEG-4 with low-power operation in mind.

The increasing use of complex algorithms in embedded systems is adding substantially to processing overhead. Fast Fourier Transforms (FFTs), inverse Discrete Cosine Transformation (iDCTs), and other compute-intensive algorithms requiring single-bit manipulation, matrix mapping, and byte and half-word arithmetic are becoming common in applications that were unimaginable a few years ago.

In many systems, the answer to the algorithm complexity dilemma is faster clock speeds or multicore processors. But for battery-operated end products such as PDAs, cell phones, point-of-sale devices, and portable media players, a more power-efficient approach is needed. New microcontroller (MCU) architectures can resolve this problem, supporting the computational intensity with a fraction of the power consumption.

MPEG-4 encoding and decoding

One example of an advanced, computationally intensive DSP algorithm, MPEG-4, is a video coding standard that allows universal, low bit-rate video data transfer by reusing most of the data from the first video frame and transferring only those bits that have changed from one frame to the next.

MPEG-4 video coding uses a block-based predictive differential video coding scheme. The main techniques for compression are division of the picture in 8 x 8 blocks or 16 x 16 macroblocks, motion-compensated prediction, transform coding with Discrete Cosine Transform (DCT), quantization, and run-length and Huffman coding for Variable Length Codes (VLCs).

Both spatial redundancy and irrelevancy are exploited with block-based DCT coding, quantization, and run-length and

Huffman coding. Only information from the picture itself is used, thus every frame can be decoded independently.

Additionally, MPEG-4 intermode encoding takes into account the temporal redundancy between the images in a video sequence. Macroblock-based motion estimation between two successive images is performed, allowing a motion-compensated prediction of the current picture. The predicted image is then subtracted from the original image and

the resulting difference picture is DCT coded, quantized, and VLC coded. The motion vectors describing the motion of the blocks in the picture are used later for decoding and are also encoded with VLC.

Both encoding and decoding are computationally intensive. During encoding, the Sum of Absolute Differences (SAD) must be calculated for all pixels in each frame. A qualitative measure of the distortion is assigned to each motion vector. The SAD for an N x N block located at position (x, y) with a given displacement (dx, dy) in the reference group of video objects is derived as shown in Equation 1.

Listing 1 shows a C-language representation of the algorithm.

$$SAD = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |pred(x+i, y+j) - ref(x+dx+i, y+dy+j)|$$

Equation 1

```

/* From sad.c of the open source xvid codec */
uint32_t sad = 0;
uint32_t j;
uint8_t const *ptr_cur = cur;
uint8_t const *ptr_ref = ref;

for (j = 0; j < 8; j++) {
    //Compute SAD for 4 bytes
    sad += ABS(ptr_cur[0] - ptr_ref[0]);
    sad += ABS(ptr_cur[1] - ptr_ref[1]);
    sad += ABS(ptr_cur[2] - ptr_ref[2]);
    sad += ABS(ptr_cur[3] - ptr_ref[3]);
    //Compute SAD for next 4 bytes
    sad += ABS(ptr_cur[4] - ptr_ref[4]);
    sad += ABS(ptr_cur[5] - ptr_ref[5]);
    sad += ABS(ptr_cur[6] - ptr_ref[6]);
    sad += ABS(ptr_cur[7] - ptr_ref[7]);
    ptr_cur += stride;
    ptr_ref += stride;
}

```

Listing 1

Every line in Listing 1 contains an addition, absolute value, and subtraction – three operations per line of code. Implementing this in a RISC architecture using standard arithmetical instructions would require 24 operations in addition to any nonarithmetic operations to align memory with load/store instructions. In a single-cycle architecture, that equates to a minimum of 24 cycles for the 8 x 8 SAD algorithm in addition to the other parts of the algorithm. This algorithm is run approximately 60-70 percent of the time during MPEG-4 encoding, putting a significant computational strain on most RISC architectures.

Similarly, the decoding of MPEG-4 data streams includes a very compute-intensive 2-D 8 x 8 iDCT algorithm shown in Equation 2.

While combining conventional MCUs and DSPs could provide the throughput for MPEG-4 algorithms, portable devices can't supply enough electrical power to do so. A better solution would be a new MCU architecture with an instruction set that supports single-cycle execution of frequently used operations and combinations thereof (FFT, SAD, iDCT, and others). This architecture would also have the command and control functionality of a microcontroller and could employ advanced computational throughput to maintain minimal power consumption.

Architectural enhancements

Developers can take the following architectural steps to improve CPU computational throughput for these algorithms:

- **Reduce the amount of load/store cycles.** More than 30 percent of the instructions executed in an RISC architecture are load or store instructions, each of which takes one or more cycles. Reducing the number of load/store cycles can have a substantial effect on processor throughput.
- **Streamline repetitive operations.** Some algorithms such as those for multimedia contain operations that are repeated thousands of times on data streams. For example, the 8 x 8 SAD algorithm contains 24 operations that must be executed on every pixel of an image during MPEG-4 encoding. Performing these operations on multiple data simultaneously (Single Instruction Multiple Data or SIMD)

$$f(x, y) = \frac{2}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(u)C(v)F(u, v) \cos \frac{(2x+1)u\pi}{2N} \cos \frac{(2y+1)v\pi}{2N}$$

Equation 2

causes a linear reduction in cycles required to process the data stream.

- **Maximize pipeline resource utilization.** Some arithmetic operations take a single cycle while others take several cycles. For example, a division operation can take 32 cycles to execute. If the processor must wait for a multicycle operation to complete before issuing a new instruction, other resources in the pipeline will be under-utilized. Allowing unused pipeline resources to be used for nondependent calculations (out of order execution) increases utilization of these resources and increases throughput per clock cycle.

can consume an enormous number of cycles. Implementing logic that predicts the branch and folds it into the instruction can reduce the branch penalty to zero cycles after the first loop iteration.

- **Improve code density.** Since memory is relatively inexpensive, few people worry about code density. However, with processors that rely on an instruction cache for fast performance, code density can have a direct effect on performance. If the code is smaller, more instructions can be stored in the cache, resulting in fewer cache misses and fewer cycle-intensive fetches from external memory. Reducing the traffic on the main system bus can also significantly reduce power consumption.

**“The processor core
can execute 30 frames
per second Quarter
VGA MPEG-4 decoding
with a clock frequency
of 100 MHz ...”**

- **Use operators efficiently.** Providing a variety of instructions (such as multiplies) that fully exploit computational resources for target algorithms can increase throughput.
- **Minimize branch latency.** Most multimedia and cryptography algorithms consist of outer and inner loops. Branches for tight inner loops can consume three to five cycles each. In DSP algorithms some inner loops are executed tens of thousands of times a second, and their branches

High performance, low power

Atmel has developed a high-performance 32-bit RISC processor core with an instruction set architecture that increases the computational throughput per cycle while also delivering ultra-low power consumption. The processor core can execute 30 frames per second Quarter VGA MPEG-4 decoding with a clock frequency of 100 MHz by implementing several of the aforementioned enhancements. Figure 1 shows a block diagram of the AVR32 processor core.

Reduced load/store cycles. The architecture has load/store instructions supporting byte (8-bit), half-word (16-bit), word (32-bit), and double-word (64-bit) widths. The instructions are combined with various pointer arithmetic to efficiently access tables, data structures, and random data.

Multiple pipelines supporting out-of-order execution. The architecture has three pipelines (load/store, multiplier, and ALU as depicted in Figure 2) that allow arithmetic operations on nondependent data to be executed out of order. For example, if a 32-cycle divide operation enters the pipeline, instructions that follow it in the code may be executed in the ALU and/or load/store pipes during those 32 cycles. Rather than halting the code until the division is complete, the

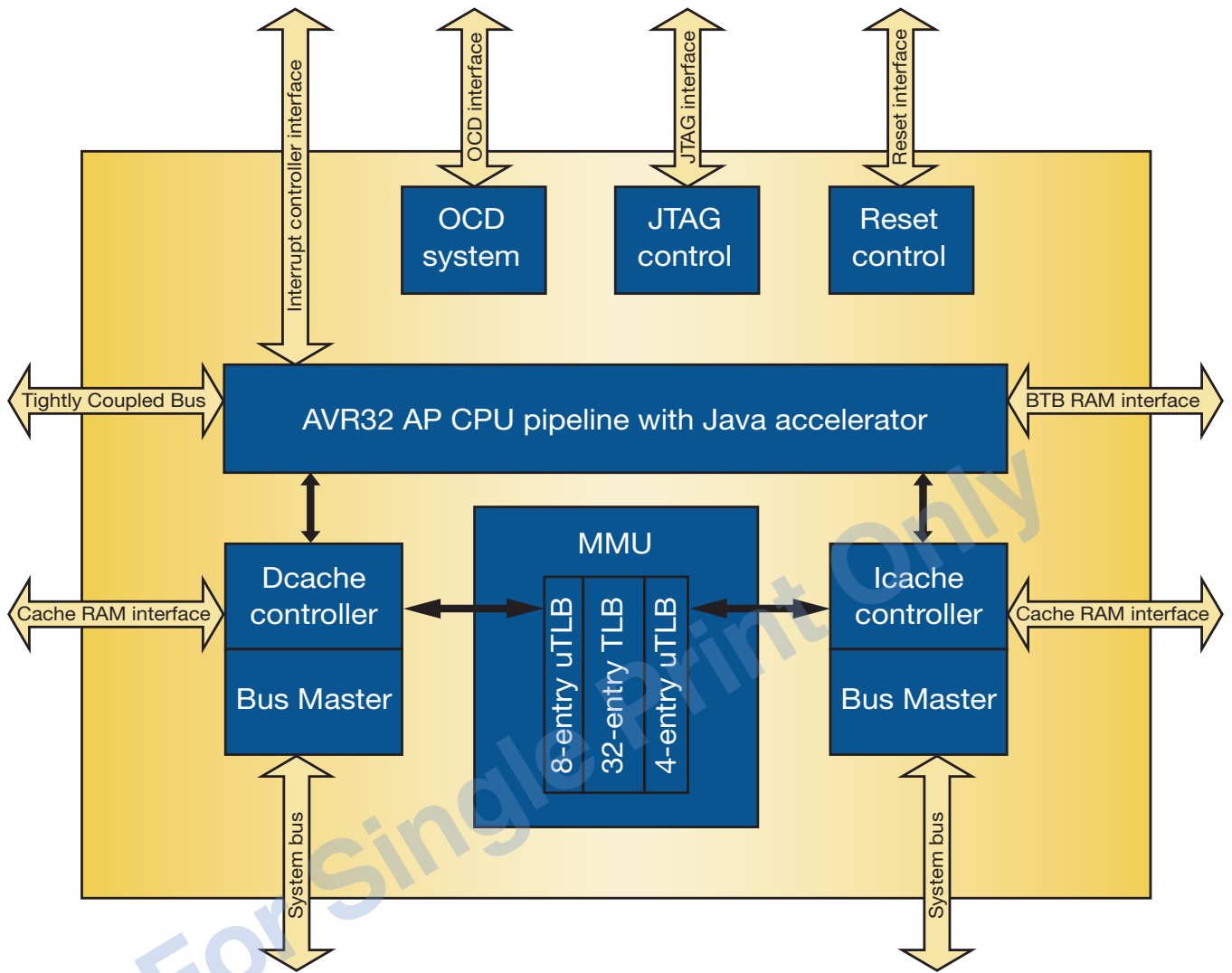


Figure 1

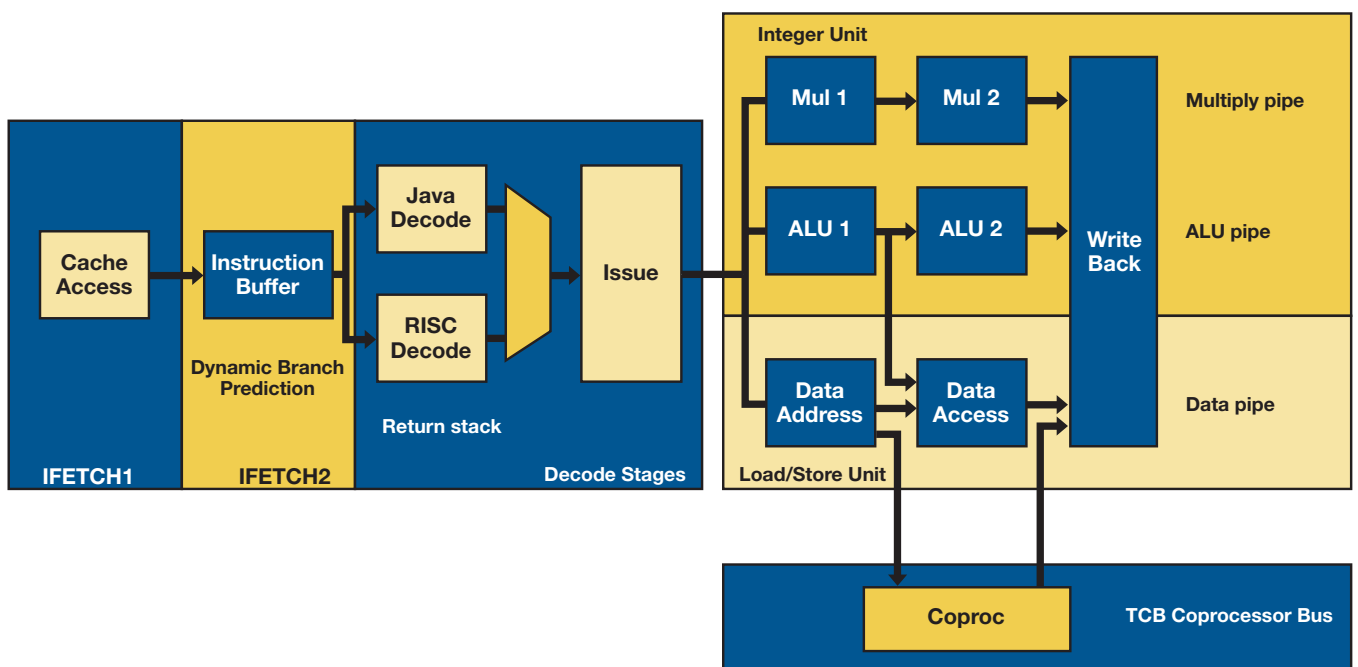


Figure 2

architecture allows instructions to execute using available resources. Hazard detection logic detects and holds dependent instructions at the beginning of the pipeline until the dependent operation is complete. The architecture also implements a full forwarding scheme where all instructions forward their results as soon as they are finished. Pipelines have branch prediction logic that can accurately predict all change-of-flow instructions, and branches are *folded* with the target instruction resulting in a zero-cycle branch penalty.

Powerful instructions. Single Instruction Multiple Data (SIMD) in the architecture can quadruple the throughput of algorithms requiring repetitive operation on a data stream. For example, an 8-bit Sum of Absolute Differences (SAD) is calculated by loading four 8-bit pixels from memory in a single load operation, executing a packed subtraction of unsigned bytes with saturation, adding together the high and low pairs of packed bytes, unpacking them into packed half-words, and adding

those together to get the SAD value. Also, improvements in the instruction set enhance code density and cacheability of code, reducing cache misses.

Instruction set supporting advanced operating systems. The architecture specifically supports the use of Linux with cycle-saving instructions, an advanced MMU, and security modes. These include an *application call* instruction that calls subroutines from a jump table with an 8-bit index and a *system call* instruction that issues a call to the operating system routine.

Keeping the power down

With these and other enhancements, the resulting architecture achieves low power usage – as low as 250 mW when active at 100 MHz. Low power, combined with the ability to process advanced MPEG-4 and other complex algorithms, can help designers take on tough tasks in small form factors using microcontrollers based on this type of architecture. ➤



Dr. Øyvind Strøm is the lead designer of Atmel's AVR32 design team and an expert in computer architectures and low-power design. He holds an MSc (1995) in Electrical Engineering from Delft University of Technology, The Netherlands. He received his PhD (2000) in Electrical Engineering from the Norwegian University of Technology and Science with the thesis, "Micro-processor for executing byte compiled Java code."

To learn more, contact Øyvind at:

Atmel Corporation
Vestre Rosten 79
7075 Tiller
Norway
+47-72-89-75-15
ostroem@atmel.com
www.atmel.com