A RISC ARCHITECTURE WITH UNCOMPROMISED DIGITAL SIGNAL PROCESSING AND MICROCONTROLLER OPERATION

Daniel Martin

Siemens Microelectronics, Inc.
10950 North Tantau Avenue
Cupertino, California US-95014

Robert E. Owen

Data/Time International
19348 Columbine Court
Saratoga, California US-95070-4039

ABSTRACT

Digital signal processors are paired with microcontrollers in many applications. Various attempts have been made to combine the two processor functions in one architecture, but there have remained two unresolved conflicts. These are different data and program memory hierarchy choices in speed and size, and different real-time control needs. This paper reviews the basic processing requirements for digital signal processing (DSP) and controllers and shows how a new 32-bit RISC architecture has resolved these conflicts and successfully integrated the two functions seamlessly into one processor core. This is confirmed with a detailed FIR filter example. Major innovations in this TriCore architecture are a novel memory organization used along with variable instruction word sizes and multiple issuing of instructions.

INTRODUCTION

Most systems that utilize digital signal processing (DSP) in their operation also have digital control processing requirements. Historically the signal processing has been done by a separate digital signal processor as a peripheral to a microcontroller with each being the more cost effective for its portion of the digital processing. They were natural partners with neither doing the other's task well. However, now all of the forces normal for the semiconductor industry are at work to encourage the integration of the DSP functionality with the microcontroller: lower power, smaller size and lower cost through smaller combined die size, less packaging costs and less testing costs.

Common Architectural Trends

In the course of serving their natural application markets, DSP and control processors have pursued some common techniques in their architectures. DSP functions are becoming more data dependent as nonlinear coding methods are used so that a full set of conditional bit operations are required in modern digital signal processors. Programs are larger with more control so high-level language support and context switching assists have been added. Thus larger memories of different types are needed and caches are starting to be used.

As controller's performance requirements have increased, their processors have moved more to Harvard (separate data and instruction memories) structures like the signal processor and to RISC from CISC. Higher and mixed precisions are more fully supported in basic hardware with SIMD (Single Instruction Multiple Data) parallel operations being used more frequently as with signal processors. Superscalar multiple issuing of instructions is more common.

In these ways control and signal processors have grown closer even without an attempt at integration, but there have also been deliberate attempts to integrate them [1]. DSP data arithmetic requirements can be added easily to a controller including even the multiply-accumulate function. Likewise, DSP data structures and program looping come at small cost in structures that are already Harvard organizations [2]. Initial blends have, however, been relatively inefficient integrations with the DSP capability being an add-on to an existing successful microcontroller [4]. The result is often more like a co-processor with a high duplication of functions and large complex instruction sets. As a result potential cost, power and ease of programming benefits are never fully realized.

Different Natures Of Signal and Control Processing

Digital signal processing is distinctly different from control processing and general purpose computing in four broad areas: data arithmetic, data structures, real-time control and program control.

DSP data structures are often vectors or even matrices where operations are, in effect, vector operations which require two operands. The vectors are often long so register files are not large enough and data caches are always a miss. Thus, large fast dual memories with elaborate dual address generators are the norm. Addresses are commonly data derived for table lookup or numerically complex like the bit-reverse of the fast Fourier transform (FFT).

Real-time control is critical in DSP because it keeps the data being processed and not backing up. This control is usually deterministic being related to the sample rate as opposed to random events and is typically data independent. Loss of real-time synchronization or data coherence is not tolerated, DSP control must be predictable.

Program control in DSP is oriented towards fast execution of tight loops of code: repeat instructions, zero-overhead looping and small instruction cache sizes with relatively primitive update strategies that can work. Nesting of loops may be important and stacks helpful but contexts seldom change, branching is rarely complex and although the benefits of high-level languages are sought, most programs are still developed in assembly language.

Control processors use more conventional arithmetic with often the only enhancements being for bit operations and character string processing. Data structures are small so RISC register files can work well. Precisions and data types vary and compact data storage is important but the normal byte boundaries are adequate. Control processing tends to be intolerant of even a single data inaccuracy so memory protection and fail-safe service routines may be important.

Real-time control is a controllers primary task: they respond to more nearly random, non-deterministic inputs, their operation is highly data dependent and there may be many discrete contexts due to multiple interrupts, traps and operating modes. Control timing is usually less critical than DSP in absolute time tolerance, but the priority and hierarchy of control is much more complex. Program control must accommodate larger programs due to wider use of high-level languages in controller applications and for the more complex tasks. Large slower memories make caches more necessary and they must be larger than for DSP. Compact instruction sets help to reduce program memory size and cost.

Previously Unresolved Conflicts

Current attempts at integrating DSP capability into controllers have resulted in certain compromises, but many of the additional requirements of DSP have been met by just adding them in. However, there are two requirements for DSP that have conflicted with controller requirements. The conflicts have not been resolved and the controller requirements have prevailed. The conflicts are in the areas of the memory organization, speeds and sizes and in the real-time control requirements.

Most DSP continuously needs fast data memory that is larger than

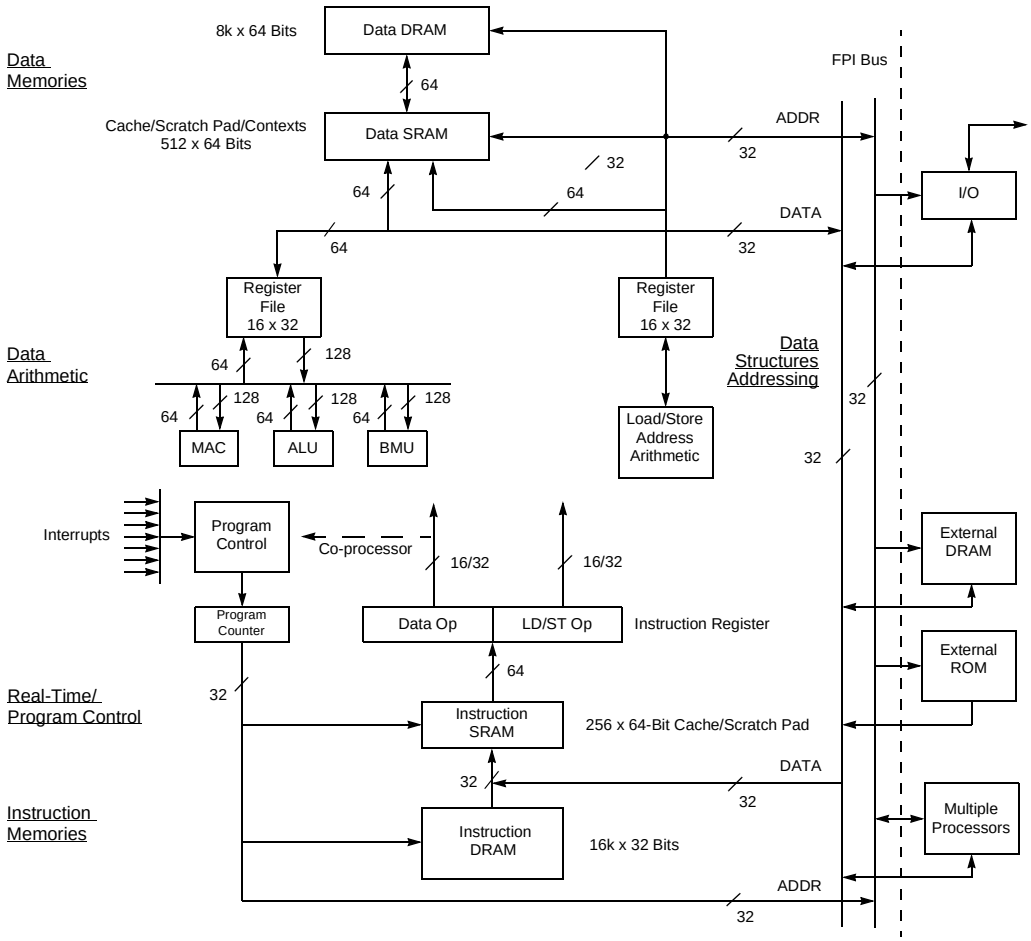

Figure 1. The First Implementation Of The TriCore Architecture

a register file. Also instruction memory must be fast, continuously available and wide enough to support the parallel processing required for DSP. These both conflict with the low-cost requirement for controllers where the need for both memories to be large can be traded-off for lower cost by using slower, on-the-average-available DRAMs. The result is that existing microcontrollers with DSP capability operate substantially below their expected performance because of the overheads of memory swapping.

However, the largest single factor that has kept signal processing being done on separate processors is the need to maintain the necessary temporal integrity for real-time DSP. Unless a conventional microcontroller can prove by design that it can deterministically be responsive with the processing power needed for the DSP portion of the task, it will always have to work far below optimum usage just to assure that it is statistically safe.

THE TRICORE ARCHITECTURE

Figure 1 illustrates the first implementation of the TriCore architecture that meets the processing requirements without compromise for both DSP and control while giving all of the benefits of integration [3]. Shown to the left of the FPI bus in the figure are the actual bus sizes and execution model of this initial core. Memory sizes and types shown are for the standard product but these are selectable along with a wide choice of peripherals in the normal configurable product.

The three arithmetic units (multiply-accumulate, arithmetic-logic and bit-manipulation) operating out of the 16 x 32-bit multiport register file do meet all of the combined data arithmetic requirements. A 64-bit data SRAM can do two full 32-bit word load/store operations with the register file simultaneous with data operations requiring three reads and two writes. The data SRAM is used for data caching, as a data scratch pad or for four-words-per-cycle context switches for both data and address register files. A larger internal 8k x 64-bit data DRAM with two word transfers and the single-word transfer external memory complete the data memory hierarchy. All share the 32-bit address space of the address generator that meets the combined complex data structure and memory protection needs.

Most data and address processing can be accomplished with 16-bit instructions sequentially, but 32-bit instructions can be used when processing, data constants or address ranges required them. Instructions can be dual issued when the highest concurrent processing is required because of the 64-bit path to the 256 x 64-bit instruction cache and scratch pad SRAM. Main internal instruction memory is a 16k x 32-bit DRAM that can be augmented with external memories on the FPI bus. All share the program control 32-bit instruction address space for large programs. This control has a full set of zero-overhead, nesting and memory protection functions. Program space overlaps the data address space for external memories.

A large prioritized vectored interrupt structure and hardware and software traps complement the fast context switching and three permission levels for easily maintained and rapid task control. Two contexts can, in effect, be resident within the split register files. Additions contexts are stacked within the data SRAM and aligned with the wide memory bus for fast saving and restoring.

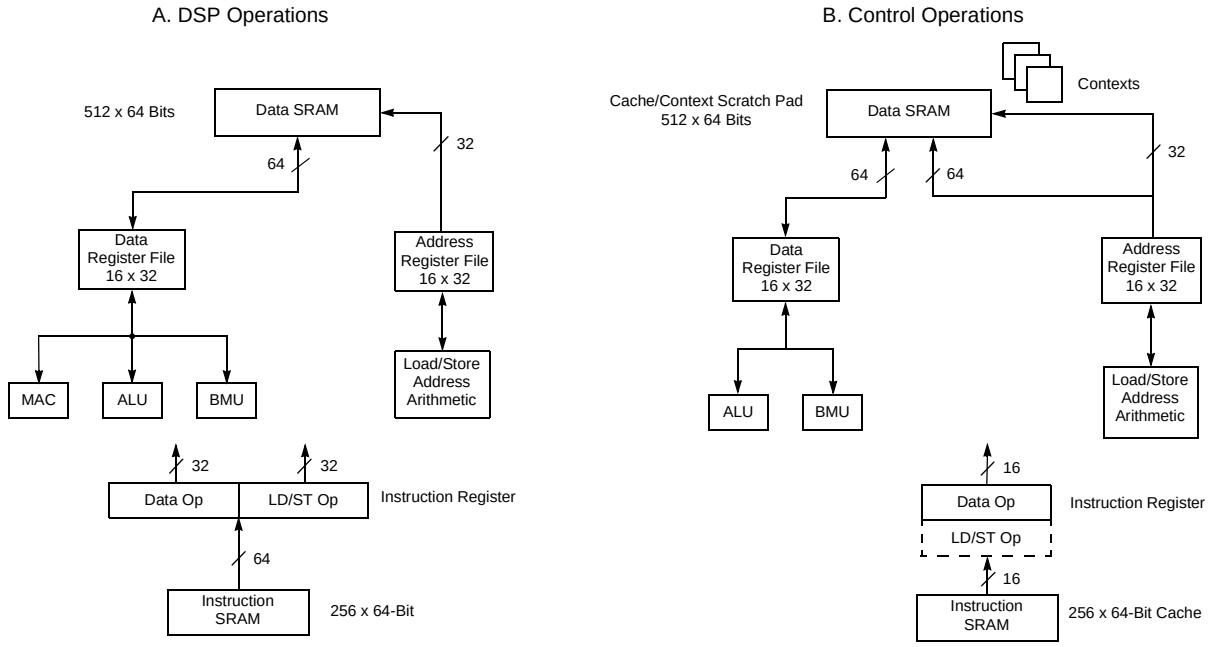

Figure 2. The Architecture In Effect For DSP Operations (A.) And Control Operations (B.)

All of these architectural features provide the RISC-like structure to support efficient C and C++ language compilers without compromise to the processing speed possible for DSP applications. Figures 2A and 2B illustrate the same core architecture redrawn to emphasize its operation for DSP and control operations. For DSP in 2A, 32-bit data operation instructions keep all three arithmetic units busy operating on the register file data that is simultaneously updated from complex data structures in data SRAM maintained by the 32-bit load/store operation instructions. The instruction SRAM holds 256 of these 64-bit instruction pairs which is adequate for the tight inner loops, while the data SRAM holds 2048 data words at the common DSP precision of 16-bits.

For control in Figure 2B, sequential 16-bit data and load/store instructions are sufficient for microcontroller performance with the bit-manipulation and arithmetic-logic units. The data and instruction SRAMs are more likely to function as caches for the larger associated DRAMs needed for large control programs. The data SRAM acts as a context scratch pad for rapid saving of both data and address register file contents. Note that these separately drawn architectures exist simultaneously without any mode switching, being determined only by the instruction flow.

Conflicts Resolved

The specific implementation in Figure 1, coupled with the configurability of the broader TriCore product line illustrate how the seemingly conflicting memory hierarchy requirements were resolved. First in the data memory hierarchy. The large directly-addressable data register file meets compiler needs and provides the small instruction benefit for both types of processing. Yet the wide, fast and transparent load/stores with the moderate sized scratch pad SRAM gives the larger fast memory with more complex addressing needed for DSP. This comes with an expansion of the instruction width with dual issuing only when specified by the program. And finally the slower, lower cost but denser DRAM meets the large data memory requirements of controllers and yet remains fast, due to wide transfers, and flexible for DSP use. Even external data memory maintains the flexible addressing along with the directly addressable I/O. All the data memory hierarchy benefits too from the data packing of bytes and half-words within the 32-bit full word and the SIMD operations of the arithmetic units.

Next is the instruction memory hierarchy. Here a wide, moderate size instruction SRAM meets the speed needs of DSP without high cost, while the larger program needs of controllers are met with a less costly internal DRAM and external memories. Instruction memory efficiency remains high without speed penalties because of the choice of 16- or 32-bit instructions, dual issuing and SIMD operations.

Both memory hierarchies can be further refined for a particular class of applications by altering the memory sizes and types. This is because of the large unified address spaces and the broad choice of memory technologies available from Siemens. Dense, high-speed non-volatile ROMs can be added anywhere as can flash EPROMs.

There is no inherent conflict between the DSP need for fast deterministic responses and the controller need for complex processing responses to a large variety of random events of varying importance. There is a choice of features in this architecture to insure speed of response, speed of resolution and availability of resources for DSP. They overlap in their effect on those three important real-time factors.

Vectored interrupts reduce response and resolution times and even dual issuing reduces response latency. Interrupt resolution times are more predictable and reduced by the large number of priority levels and the existence of non-maskable interrupts. The two sizes of context switching available, using the fast save/restore with the wide data SRAM, also assure fast interrupt and trap response, resolution and availability of resources in the correct context. Three levels of permission assure processing resources are used for time-critical program execution. Software set-able traps can provide rapid dynamic control of the system's real-time response. The processor priority level itself is dynamically adjustable. Conditional execution of code can be used to insure that execution times are consistent.

COMPREHENSIVE EXAMPLE

Successful integration of DSP capability in the architecture can be confirmed by looking at a typical programmed DSP example, the

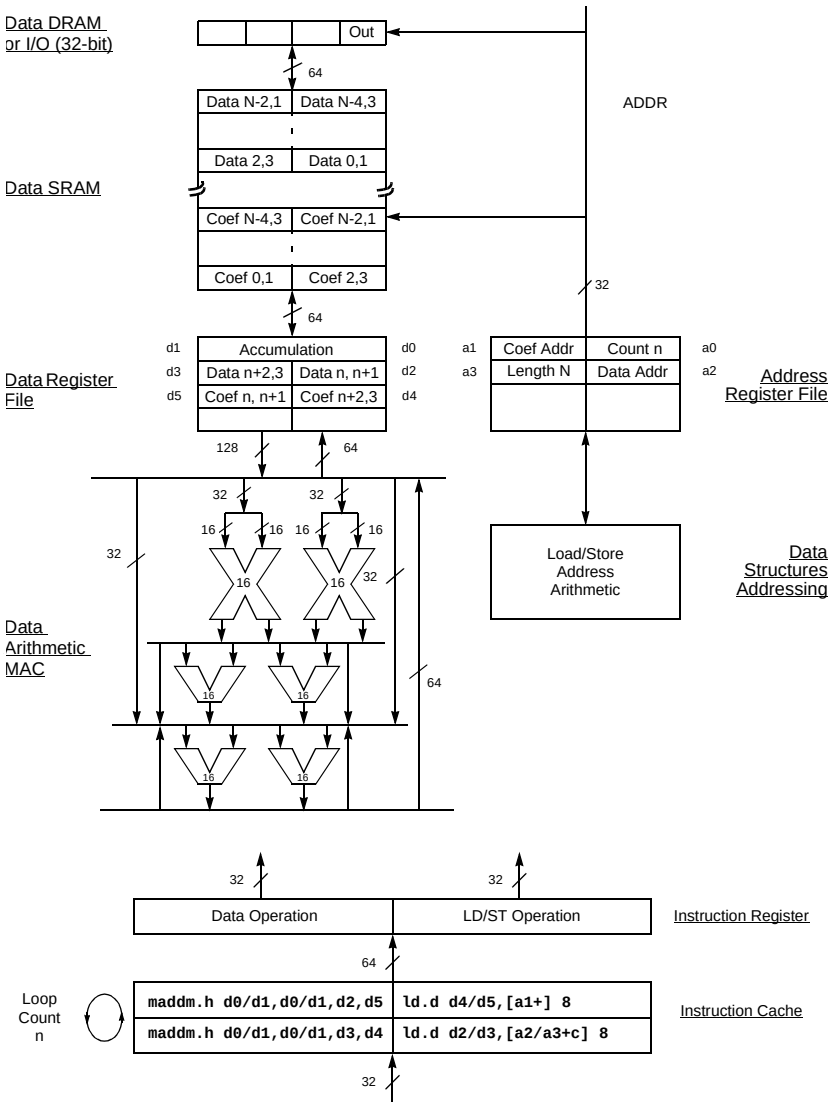

Figure 3. Data Memory And Data And Address Register File Organization For A 16 x 16-bit FIR Filter With 64-bit Accumulation

FIR filter. A FIR (finite-impulse-response) filter is a continuing computation over time of the form:

where N is the number of multiply-accumulates or filter taps on the uniformly sampled data points with the index *t*. This computation is mapped onto the new architecture in Figure 3, where the details of the MAC arithmetic unit are also shown. This is for the common case of where input and output data and coefficients are all 16 bits with 32-bit products being accumulated in a 64-bit register.

If the input data and coefficients are ordered in the data register file as shown in Figure 3, then four 16 x 16-bit multiply-adds can be done in two instruction cycles with two **maddm.h** instructions. Simultaneously the 64-bits of input data and coefficients can be updated with loads from the data memory. If N is even and less than 256, then the input data and coefficients can all reside in the data SRAM in the first implementation with the ordering as shown. Processing time without I/O is 7 + N/2 instruction cycles. With the target 100-MHz instruction rate and allowing for I/O, the real-time bandwidth can be 5 MHz for a 20-tap non-symmetrical FIR filter.

This example illustrates specifically how the high performance is a direct result of features in the new architecture. First the partitioning of the multiplier-accumulator into a dual MAC for 16-bits doubles the rate for this, the most common, DSP precision. The MACs operate with conventional signed fractional data formats that saturate properly without an unnecessary loss of precision to cover the -1 times -1 possibility. The double-precision 64-bit accumulation takes no additional cycles because of the wide data buses and dual register addressing. The dual issuing allows the simultaneous loading of the register file with the ongoing MAC operations for the small inner-loop instructions that are cache resident, yet only a 32-bit main instruction memory is needed. The instruction loop counting and comparison takes no additional cycles.

The double-word data path between the data memory and the register file allows the use of smaller, less costly single-port data memories, including DRAM. The memories are addressed by alternating counters that are self incrementing by variable amounts and invisibly maintain a circular buffer. The addressing supports compact use of data memory for a smaller than full word precision. The Input/Output operation is flexible with a variety of format and precision options either in on-chip memories or external 32-bit I/O, all of which appear in the same direct address space.

As can be seen from the techniques used, this high performance is broadly useful for DSP and not just for this special case or set of parameters. Performance decreases only gradually with using the larger on-chip data DRAM rather than SRAM. Even use of external RAM can be a penalty of a factor of only two. With byte addressing and the full range of shifting and sign-extension capabilities it is easy to use different precisions for saving memory space. Higher precisions impose small penalties generally because of the wide data bus structure. For example, in the FIR filter if the precision is expanded to 32 x 32 bits with 64-bit accumulation the data organization remains the same as Figure 3 where a new *n* equals the old *n, n+1*. The speed is reduced by a factor of only two because it takes four **maddm** instructions to accomplish the four MAC operations.

REFERENCES

[1] D. Bursky, “Merged Embedded Controller And DSP Designs Simplify Systems”, *Electronic Design*, Vol. 45, No. 21, pp. 69-80, 1 October 1997.

[2] K. Nadehara, M. Hayashida, I. Kuroda, “A Low-Power, 32-bit RISC Processor With Signal Processing Capability”, *VLSI Signal Processing, VIII*, pp. 51-60, IEEE, 1995.

[3] “*TriCore Architecture Manual*”, Siemens Microelectronics, Inc., 1997. Order Number: M32T011.

[4] D. Walsh, “Piccolo - The ARM Architecture for Signal Processing: An Innovative Architecture for Unified DSP and Microcontroller Processing”, *Proc. ICSPAT96*, Vol. 1, pp. 658-663, 1996.