VERY LONG INSTRUCTION WORD ARCHITECTURES FOR DIGITAL SIGNAL PROCESSING

Jonathon D. Mellott[1,2] *Fred Taylor*[1]

[1]University of Florida, Department of Electrical and Computer Engineering, Gainesville, FL, 32611, USA
[2]The Athena Group, Inc., 3424 NW 31st Street, Gainesville, FL, 32605, USA

ABSTRACT

Due to advancements in semiconductor processing technology, unprecedented levels of system integration are now possible in digital signal processing systems. MIMD/multicomputer architectures used for parallel digital signal processing applications are not always efficient, and are difficult to program. Very long instruction word processors are uniquely suited to digital signal processing applications, able to exploit opportunities for fine and coarse grained parallelism efficiently without the overhead of MIMD/multicomputer approaches. A flexible, high-level language programming environment has been developed in support of this processor paradigm.

1. INTRODUCTION

Since the 1970s the semiconductor industry has experienced geometric growth in the number of transistors that can be placed on a chip [1]. With time, designers of digital signal processing devices have been able to take advantage of the geometric growth with respect to the number of transistors that could be placed on a chip to produce successive generations of processors that offered greater performance due to the increased number of circuit elements available. For example, consider the TMS320 DSP family. Using the sixteen bit, fixed-point C10 generation as a baseline, the C20 generation augmented the C10 architecture with a fast multiplier. The C30 generation used a thirty-two bit floating point architecture. The C40 generation added DMA processors for multicomputer interconnect to the C30 core. As the number of available circuit elements per chip increases, increasingly more functionality can be added. As the number of functional units that can be placed on a single chip processor increases, the question of how to actually use those resources efficiently becomes very difficult to answer.

In this paper, the application of very long instruction word (VLIW) architectural techniques for high performance digital signal processors that are not highly application specific will be examined. Existing solutions based on general purpose digital signal processors have concentrated on MIMD parallel solutions (e.g., Texas Instruments TMS320C40 and TMS320C80 products); these solutions have not proven to be entirely satisfactory due to system integration and software development obstacles. While, for the development of DSP algorithms, a block diagram approach is attractive, the application of that methodology to a multi-DSP microprocessor solution may lead to regrettable software and hardware inefficiencies, see Figure 1. In particular, interprocessor communications can consume significant hardware resources, while barrier synchronization can consume significant processor time resources. Also, software development cycles are extended when programming in this type of environment when compared to a conventional uniprocessor development environment. While this latter element is mitigated to some extent by new tools that have been developed to address the difficulties inherent in this type of programming, the former problems remain.

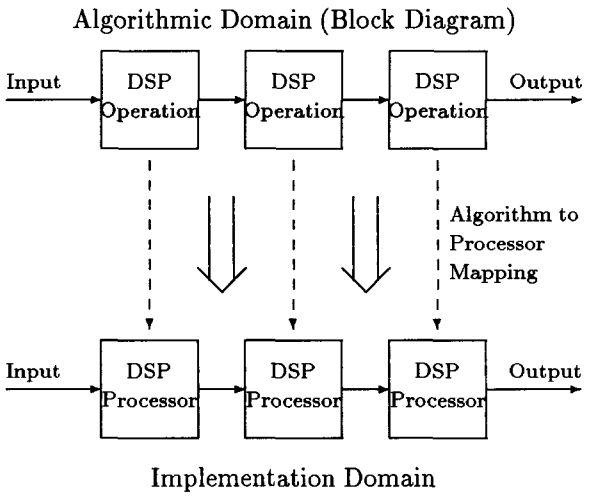

Figure 1. Conventional Mapping of DSP Block Diagram to Multiprocessor Implementation

Digital signal processing applications are especially well suited for VLIW architectures, given their regular algorithmic structure and dataflow properties, and that the nature of digital signal processing implementations sidesteps the most troublesome software life-cycle compatibility issues that currently hinder the widespread application of VLIW techniques in the general purpose computer market. VLIW techniques can be used to exploit opportunities for instruction-level parallelism just as superscalar and superpipelining techniques are also used to exploit opportunities for instruction level parallelism [2]. VLIW instruction scheduling techniques can also be adapted to allow opportunities for block-level parallelism to be exploited. A significant advantage of VLIW architecture over the competing superscalar architecture is that the hardware resources

that are expended in superscalar architectures to support multiple instruction issue are eliminated in VLIW and are therefore available for additional functional units or other architectural resources [3].

2. ARCHITECTURAL FEATURES FOR VLIW DSP

The demands placed upon a DSP microprocessor are somewhat different from those placed upon a general purpose processor. These differences impact the choices that are made in the design of the architectures for each problem. Some of the most significant differences are:

- **Processor Core** In a general purpose machine, the processor core is called upon to perform a variety of arithmetic and logic operations on a variety of data types ranging from characters, to integers, to floating-point numbers. In a DSP machine, the processor core only has to perform multiply-accumulate operations efficiently to be efficient for most DSP applications.
- **Memory Addressing** In DSP applications most processing is performed on arrays of data. The types of processing to be performed usually demand special addressing features such as circular or bit-reversed addressing of arrays. Due to the dominance of these operations in DSP, justification for inclusion of these resources at the hardware level is easily obtained, whereas in general purpose architectures these features usually cannot be justified.
- **Data Locality** Data locality differs greatly between general purpose applications and DSP applications. Since most DSP operations are upon arrays of data, there is usually little benefit to a large register file as would be found in a general purpose architecture. Likewise, since DSP applications tend to operate on large arrays of data, they violate the "cache assumption" that justifies the inclusion of data cache memories in general purpose architectures. Instead, since access to arrays of data in DSP applications is predictable, inclusion of on-chip data memories is more beneficial than data cache.
- **Programming** The issues in programming a DSP are significantly different from those in general purpose computing. Since most DSP applications are single-tasking, hard real-time in nature, instruction execution timing must be predictable in order to guarantee task completion. Since most DSP applications are embedded, the use of assembly language programming or additional programmer effort to optimize code can usually be justified if it resulting in more efficient use of hardware resources which can be reflected in manufacturing savings.

In order to meet the needs of high performance DSP applications, the developed VLIW architecture for DSP includes the following architectural resources:

- integer (fixed-point) multiplier-accumulator and ALU units,
- memory addressing units capable of supporting circular (modular) and bit-reversed addressing,
- local data memories,
- DMA for programmed management of on-chip and external memories, and
- correlation/convolution/filter accelerator engines.

A block diagram of a VLIW DSP architecture is shown in Figure 2.

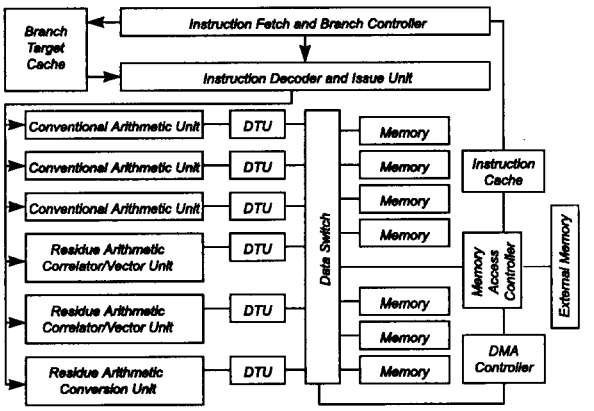

Figure 2. Block Diagram of VLIW DSP Architecture

In the block diagram of the architecture in Figure 2, the external memory access interface is an obvious potential bottleneck. A practical architecture must be essentially von Neumann — at least externally. Internally, the architecture may use whatever reasonable machinations that are required to prevent processing resources from stalling. In order to mitigate the von Neumann bottleneck, on-chip memory resources are included. These resources are not cache memories. The on-chip memories must be managed under programmed control. While this might appear to be difficult, since DSP applications have regular data access patterns, it is not an impossible task.

The correlator/convolution/filter accelerator engine uses arithmetic units based upon the residue number system (RNS). The architecture used for these units is influenced by the the previously reported Athena Sensor Arithmetic Processor (ASAP) [4], a special purpose device capable of performing video rate FFTs and image processing operations, see Figure 3. A block diagram of the correlator/convolution/filter accelerator engine is shown in Figure 4. The pictured accelerator block is shorter than the correlator array in ASAP. By using a shorter array of processors, smaller computations can be handled more efficiently than with the longer ASAP array. In order to handle longer computations a chaining option is provided. Additional capabilities have been added to the arithmetic units to aid in the execution of common video signal processing applications. By combining RNS arithmetic capabilities, which can provide an order of magnitude speed-area advantage over conventional arithmetic technologies, with conventional arithmetic units to offset the well-known disadvantages of residue arithmetic computing, a high-performance, low-cost architecture is achieved.

The developed architecture is scalable in that it is possible to synthesize processors with varying numbers of each

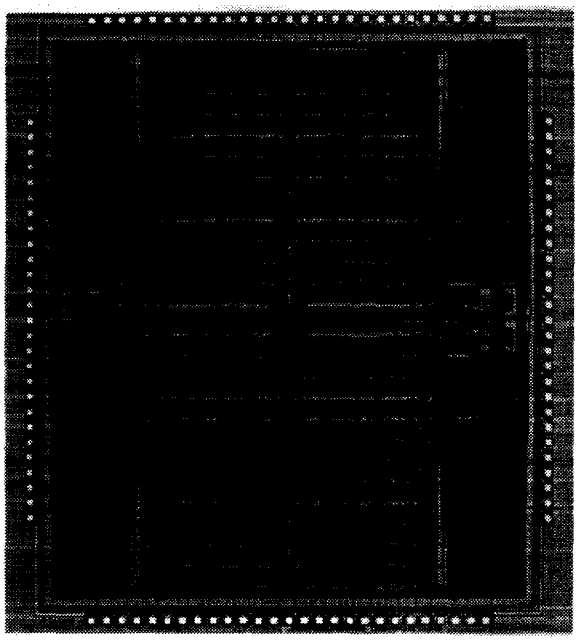

Figure 3. ASAP RNS Array Processor (courtesy of The Athena Group, Inc.)

functional unit. This allows the processor to be tailored to a particular problem in a relatively efficient manner. For example, if it is known that RNS computational units will not be used then they do not have to be included in the architecture. Also, if during the software development cycle it is determined that the processor has oversubscribed (or undersubscribed) functional units, then the processor architecture can be tuned appropriately with an incremental increase (or decrease) in expense. The high-level language software development environment, discussed in the the next section, allows code to be migrated into different processor performance levels with a recompilation.

3. SOFTWARE SUPPORT FOR VLIW DSP

Programming a VLIW architecture is potentially a daunting task. Fortunately, DSP application codes tend to be small and therefore are approachable. However, there are a number of problems where the difficulty and expense of working in assembler cannot be justified. Furthermore, the software development cycle is having an ever increasing impact on time-to-market for many product development efforts. In order to speed the software development cycle, more DSP programmers are turning to high-level languages to speed up the software development cycle. The language of choice is the C programming language. C was originally developed as a high-level assembly language to allow the implementation of Unix on a general purpose machine. Many of the features of the C language reflect the underlying architecture of the machine for which it was developed; those machines that have come after have, to a large extent, been optimized to run programs written in C. The adoption of the C programming language as a "high-level assembly language" for DSP processors has lead to poor results. The features of the C programming language do not all map efficiently to DSP processors, and many of the special features of a DSP processor are not reflected in the C language. Two techniques have been used in order to mitigate this problem, namely hand-coded libraries and idiomatic translation. Hand coded libraries are a good solution for many particular DSP problems but suffer from the fact that they must be coded for each architecture in which one might use them. Idiomatic translation has the disadvantage of layering an additional level of "language" over the core language and serves to further complicate the software development task. What is really needed is a high-level assembly language for DSP.

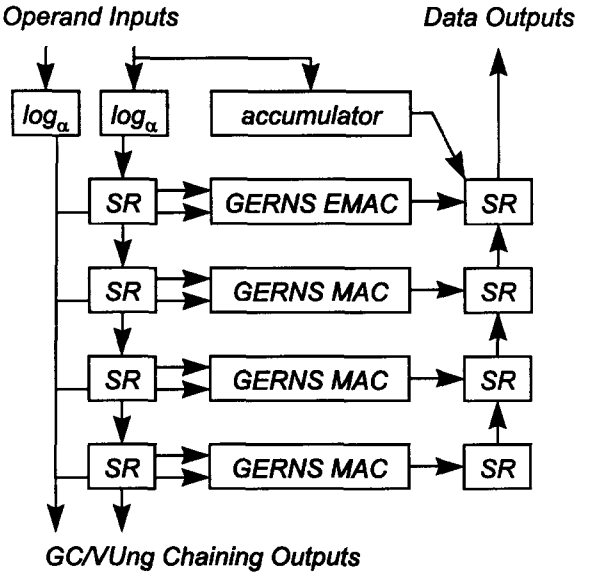

Figure 4. Block Diagram of RNS Correlator/Convolver Vector Unit

The authors have developed and are refining a high-level assembly language for digital signal processing that is based loosely upon the C language and is called C_{dsp}. The C_{dsp} language has constructs that reflect the architecture of DSP processors and eliminates those constructs that do not translate efficiently into DSP code. The language also includes features that allow the programmer to explicitly identify opportunities for block-level parallelism along with relaxed data dependency relationships. By working in a high level language the programmer is also insulated from architectural shifts. As previously stated, it is conceivable that within the course of a software development effort that the number of functional units in a VLIW processor might change. This would impact all assembly code (since the size of the instruction word would change), but would not effect a high-level language program at the source level. The advantages of this are manifold, and well understood in the general purpose computing market but have not yet been realized in DSP applications.

In order to demonstrate the advantage of this approach, consider the inner product — the work-horse of DSP. The inner product,

$$y = \sum_{n=0}^{N-1} a_n x_n, \qquad (1)$$

is typically implemented using a tight loop on a DSP microprocessor. Generally, the sum is accumulated in N cycles using a sequence of multiply-accumulate instructions. It is clear that the inner product is parallelizable by breaking the sum into two smaller sums and then adding:

$$y = \sum_{n=0}^{\lceil N/2 \rceil - 1} a_n x_n + \sum_{n=\lceil N/2 \rceil}^{N-1} a_n x_n. \qquad (2)$$

On a multicomputer configured DSP system such as that shown in Figure 1, little or no speedup may be achieved by parallelizing this sum due to interprocessor communication and barrier synchronization overhead. In a VLIW environment, the sum is parallelized at compile time according to the number of functional units available. Barrier synchronization comes at little or no run-time expense since an optimal instruction schedule where timing relationships between instruction streams are known, is generated at compile time. Fine grained instruction level parallelism comes at no hardware expense since all instruction scheduling is performed at compile time.

3.1. Compiler Implementation

The C_{dsp} compiler is implemented in the usual manner [5]. An LALR grammar, Lex, and Yacc are used to generate a translator that produces an intermediate three-address code. This intermediate code is then given to an optimizer that provides many of the usual optimizations expected in a high level language compiler [6]. Instruction scheduling and code emission is controlled by a target processor profile that determines the number of each computational resource available in a particular target configuration. A block diagram of the compiler flow is shown in Figure 5. The instruction scheduler uses well understood techniques for exploiting instruction level and block level parallelism, such as trace scheduling [7].

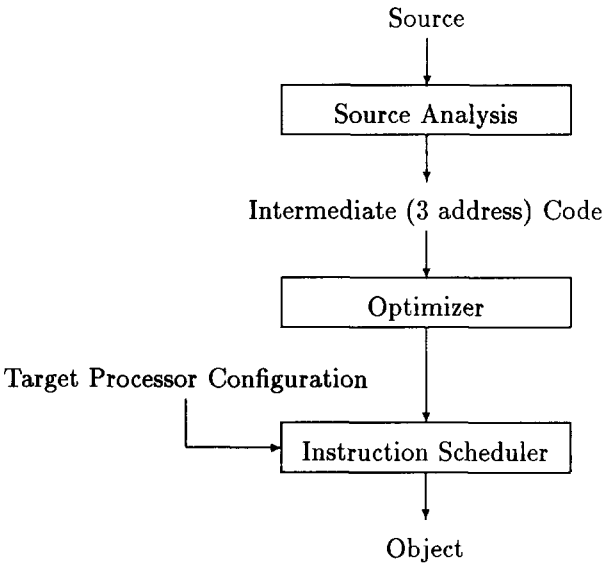

Figure 5. Block Diagram of Compiler Flow

4. CONCLUSIONS

A VLIW architecture is an attractive alternative to MIMD/multicomputer approaches for implementing digital signal processors with multiple functional units. VLIW offers the advantages of superscalar implementations without the overhead expense of instruction scheduling hardware since all instructions are scheduled at compile time. Digital signal processing algorithms are particularly well suited to VLIW architectures due to the dominance of simple loops and regular dataflow. With high-level language software development tools it is possible to write applications and target them to processors with just enough of each processor resource (e.g., arithmetic units, memory) to meet the requirements of the particular problem.

This work is preliminary in nature. As the work proceeds it will be possible to judge the ultimate limitations of this approach. Of particular interest are the upper limitations on the number of functional units that may effectively be scheduled for typical DSP code and the level of performance that is represented by that upper limit. It will also be necessary to produce a fair comparison of this processor architecture with current, state-of-the-art digital signal processors.

REFERENCES

[1] G. D. Hutcheson and J. D. Hutcheson, "Technology and economics in the semiconductor industry," *Scientific American*, vol. 274, pp. 54–62, Jan. 1996.

[2] M. Gokhale and W. Carlson, "Introduction to compilation issues for parallel machines," *Journal of Supercomputing*, vol. 6, pp. 283–314, Dec. 1992.

[3] J. Fisher and B. Rau, "Instruction-level parallel processing," *Science*, vol. 253, pp. 1233–1241, Sept. 1991.

[4] J. D. Mellott, M. Lewis, and F. J. Taylor, "ASAP – a 2D DFT VLSI processor and architecture," in *Proc. IEEE International Conf. on Acoustics, Speech, and Signal Processing*, (Atlanta), 1996.

[5] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools*. Reading, Massachusetts: Addison-Wesley, 1986.

[6] M. Wolfe, *High Performance Compilers for Parallel Computing*. Reading, Massachusetts: Addison-Wesley, 1996.

[7] J. A. Fisher, "Trace scheduling: A technique for global microcode compaction," *IEEE Trans. Computers*, vol. 30, pp. 478–490, July 1981.