SMARTCORES ARTICLES

Multiple and Parallel Execution Units In Digital Signal Processors

Ovadia Bat-Sheva, Wertheizer Gideon, Briman Eran

DSP Group Ltd., 5 Shenkar St., Herzliya 46120, Israel

Email: batsheva@dsp.co.il, gideon@dsp.co.il, eran@dsp.co.il

Abstract

The paper introduces a new trend in fixed-point DSPs architecture. It focuses on adding execution units and parallel operations. The paper reveals the difficulties the architecture-designers encounter while dealing with this issue, such as the memory-computation unit connectivity and operating the different arithmetic units. Solutions and examples are presented, taking into consideration the performance, power consumption, area and complexity. Discussions on related issues, such as code compactness, are included, as well as architecture approaches with regard to the instruction word (VLIW, PIW). The paper includes examples of new DSPs including DSP Group's new generations, PalmDSPCore^{TM} and TeakDSPCore^{TM}.

Introduction

The main computation unit within a Digital Signal Processor is constructed of a multiplier, an Arithmetic and Logic Unit (ALU), a shifter and accumulators (or registers). The ALU is capable of performing both arithmetic (add and subtract) and logic (and, or, xor, etc.) operations. Regarding multiplication, two approaches are well founded – using a multiplier that only generates the product to be accumulated, or a MAC (Multiply and ACcumulate) unit. A barrel shifter usually handles shift operations, both arithmetic and logic. The accumulators are used to store the intermediate or final results.

Current multimedia and communication standards such as GSM or IS-95 require between 60 to 80 MIPS (Million Instructions Per Second) to support a typical phone. New generation DSP algorithms like the third generation IMT-2000 will require several hundreds of MIPS. These, as well as other DSP oriented applications, will require DSPs with higher operation speed and intensive timing requirements for the design of multipliers, arithmetic units and memories.

Conventionally, the way to enhance the DSP MIPS was to increase the operation speed and improve local critical portions of algorithm through hardware elements or dedicated instructions. New advanced DSPs like DSP Group's PalmDSPCore and TeakDSPCore, Texas Instruments' TMS320C6x, or the Lucent Technologies' DSP16xxx take a new approach. These DSPs include several computation units that can work in parallel. This, on one hand, can increase the throughput of the DSP dramatically, and on the other hand, adds complexity and cost to the overall DSP design and usage.

Adding functional units and using them in parallel sets many challenges to the architecture designers, among them are:

Selection of the units within the main computation unit and considerations in operating them in parallel.

Connection of the various functional units to each other and to the memories.

The implications on the instruction set and the way it is encoded.

Arithmetic Units

Arithmetic operations play a major role in the DSP algorithm. These include addition, subtraction, multiplication, shift and any combination of the four. The way the units that carry out these operations are utilized and parallelized, suggests various implementations and has a major impact on the DSP performance.

When adding parallel execution units, trying to meet the increasing demands from the DSP, special attention should be given to these powerful, yet relatively large and power consuming units. Several considerations should be taken into account when deciding upon this issue, among them are:

Performance – boosting as many benchmarks as possible.

Area and power consumption.

Encoding the instruction set – more complex units means more combinations of instructions.

Usage of other resources – such as accumulators, addressing registers etc.

Connectivity between the different units and the memories.

Frequency degradation – the additional complexity and connectivity might affect the cycle time.

When trying to determine the units ought to be added, one should consider the benefits gained on the DSP algorithms.

For instance, in frequently used DSP routines, such as FIR (Finite-Impulse-Response), IIR (Infinite-Impulse-Response), vector dot multiplication, convolution and correlation, the basic operation is a repetition of a single cycle MAC instruction. Executing dual MAC in a single cycle has the potential to decrease the cycle count by two. However, one has to consider the interface with the memories and other arithmetic units when adding **a second multiplier** (or **a second MAC unit**).

Clearly, when using two multipliers instead of two MAC units, another arithmetic unit should be added in order to handle the results of the second multiplier (see Figure 1). **An additional ALU** can also boost other DSP routines, such as vector addition (even if a MAC unit is used) and control routines. However, should the programmer use all possible combinations of instructions? What about the additional area and power consumption this method inflicts?

A derivative of the above-suggested method is adding **an Adder/Subtractor Unit** (**ASU**) instead of the additional ALU. This unit cannot perform any logic instructions, hence the programmer will not be able to execute a logic instruction in parallel to another logic instruction (which is not a very commonly used combination).

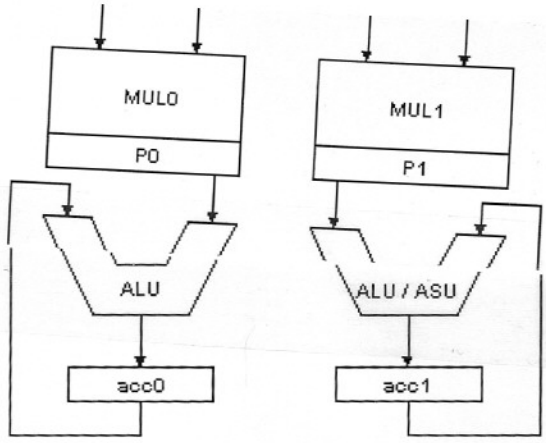

Figure 1 Two ALUs Structure

As of the multipliers, one can also carry out similar simplification on the additional multiplier. For example, the second multiplier will be able to multiply signed numbers only. This simplification of the multiplier is more problematic than the ALU's, since in many double-precision applications such unsigned multiplications are a must.

Another conceptual problem that arises while duplicating arithmetic units, is the usage of other related resources, such as accumulators, pointers and data busses. As for the accumulators, an example is a FIR calculation that is performed in two multipliers and two ALUs (or the ALU and the ASU). Since both arithmetic units are utilized, two accumulators must be used in order to save the intermediate results. Moreover, an extra addition cycle should be carried out, in order to obtain the final result. Therefore, in order to generate a single result, two accumulators are activated, and the total cycle count is slightly increased.

As an alternative to the method of duplication, **a three input ALU** (**3IALU**) can be used. This unit is capable of adding three independent values simultaneously, and to lock the result in a **single** accumulator. Regarding the previous FIR example, the two products are accumulated together with one accumulator in this unit and there is no need for the extra addition cycle.

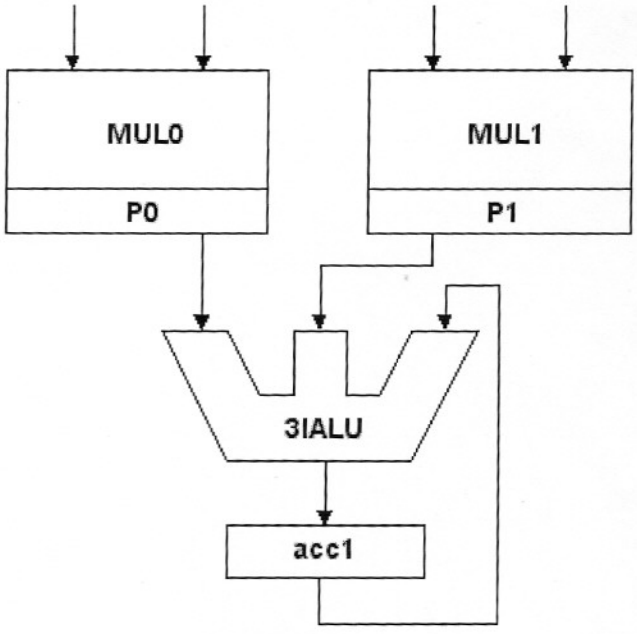

Figure 2 Three Input ALU Structure

On one hand, when using this unit only one accumulator is used, only one set of flags is required (e.g. to check overflow or sign) and less power is consumed (the three input ALU consumes much less power than an ALU and an ASU). On the other hand, some benchmarks, such as complex multiplication (including FFT), are two-results native. They require two independent results (e.g. real and imaginary), so that two separate units and accumulators would fit best. A drawback of using a 3IALU is the added complexity in calculating the flags, such as the carry and overflow flags.

The PalmDSPCore took advantage of merging the two structures of ALU and ASU and 3IALU. This implementation takes the benefits of both structures, achieving the performance increase, while managing the current consumption efficiently.

Adding **a separated arithmetic unit** in a **different block**, that is able to manipulate registers within this block, would reduce the total cost of such operations dramatically. Moreover, such unit accompanied by flags, together with correct pipeline planning may cut down the pipe penalty for conditional instructions, resulting in another boost in performance.

A common variation of using the arithmetic units are the SIMD (Single Instruction Multiple Data) operations. These are appearing both in DSPs and in general purpose processors. These exploit a limited amount of parallelism, by packing smaller precision data in larger containers (usually **a split arithmetic unit**), and by setting up the arithmetic unit to perform multiple smaller precision slices of same operation on the individual subwords of the inputs [1]. For example, SIMD mechanism is applicable for the Viterbi decoding used in telecommunication applications for error control coding. The core of this algorithm is an Add-Compare-Select (ACS) operation: A branch metrics is added to the two old state path metrics, providing two new path metrics, out of which one metric is selected, and saved as the new path metric. Packing the data and feeding it into a split arithmetic unit accelerates this routine.

The PalmDSPCore for example, contains seven arithmetic units, including multipliers, adders, ALUs and shifters, all of which are accompanied by the appropriate flags. Both SIMD and MIMD (Multiple Instruction Multiple Data) instructions are supported as well.

Connectivity between Memory and Computation Units

The operands for the various arithmetic units operations require simultaneous multiple memory accesses. The connectivity between the memory and the computation unit influences performance on one hand and the complexity on the other.

In a single cycle MAC instruction, the multiplication requires two data operands simultaneously, causing multiple memory accesses. When adding a second multiplier, four operands should be fed into the multipliers. We will examine two approaches to connect the memories to the multipliers.

One approach is to connect the first multiplier directly to the memory, feeding the multiplier's input registers with two different data operands, that reside in two different data locations (for example, using two memory banks or a dual port memory). The second multiplier's input registers, are connected to the first multiplier's input registers, fed with previous values or from the same memory locations.

Using this method, a Block FIR filtering routine is implemented as follows: each multiplier calculates a different output sample, using N input samples and N coefficients while accumulating the output result in a different accumulator (refer to Figure 3, illustrating three output samples' calculation). The main operation is a dual MAC in a single cycle, performed in the following manner. The first multiplier, multiplies a sample, x(i), by a coefficient c(i). At the next cycle the first multiplier will use the next sample x(i+1), multiplied by the next coefficient c(i+1), while the second multiplier will use the same sample, x(i+1), but with the previous coefficient c(i), saved at the input register of the first multiplier. Note the reduced memory access, hence the reduced power consumption that this solution offers.

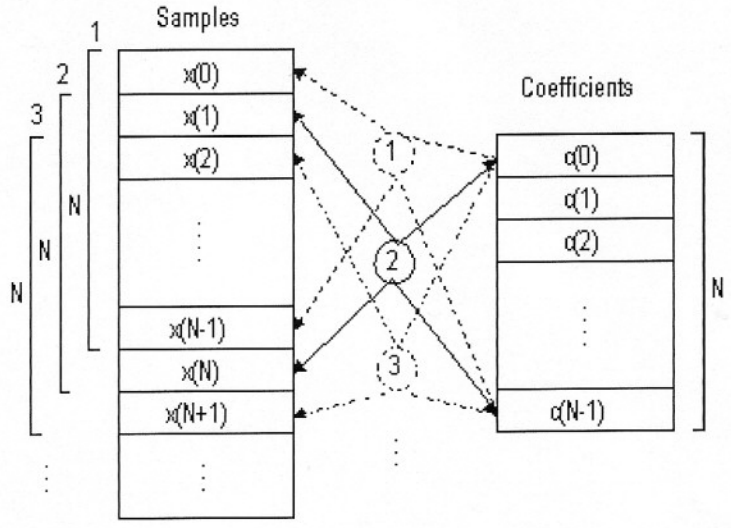

Figure 3 N-TAPs Block FIR

However, this method does not perform well in the case of a Single-Sample FIR filtering routine. In this routine, it is required to produce only a single output sample per invocation of the function [2], meaning that a single MAC should be used.

A different approach, which is the current trend implemented in many new DSPs such as TeakDSPCore, PalmDSPCore and DSP16xxx, is to use two independent multipliers by connecting the four inputs of the two multipliers to four different data memory locations. This method increases the data bandwidth in terms of memory transactions, thus providing a better performance than the first method.

In the example of FIR implementations, the Block FIR routine and the Single-Sample FIR routine, we can achieve the N/2 cycles per FIR by using a dual MAC in a single cycle. The code implementation itself depends on the connections between the multipliers and the accumulators and the instruction set. Rich instruction set and complex connections between the multipliers, as in PalmDSPCore, can offer cycles and power consumption reductions.

We discussed the connectivity between the memory and the multipliers. Note that the memories may also be connected to the other arithmetic units, in various manners.

More on Connecting the Computation Units to the Memory

Until now we reviewed the connections between the memories and the computation unit. The common suggested approach increases the bandwidth from the memory to the multipliers. It can be an advantage also to increase the bandwidth from the computation unit back to the memory. Examples can be seen in the instruction set of TeakDSPCore and PalmDSPCore.

In traditional DSPs, the programmer could store only one word to the memory in a single cycle, hence saving one portion of an accumulator. However, in double precision algorithms, it is required to store the two portions of an accumulator. This was usually done in two cycles, saving a different portion of the accumulator in each cycle. Increasing the memory bandwidth enables saving the two portions in a single cycle.

Additional advantage can be gained by adding to the instruction-set a single cycle instruction that saves the two high portions of two different accumulators. Different algorithms can benefit from this improvement:

Block FIR - saving two output-samples in a single cycle.

Complex algorithms, including FFT - saving the real and the imaginary parts in a single cycle.

LMS algorithm (adaptation of the filters' coefficients using a Least-Mean-Square algorithm) - two coefficients are saved in a single cycle

Defining and Encoding the Instruction Word

In the effort to attain high performance DSP, companies have recently introduced DSPs with multiple functional units. This, on the one hand provides higher throughput and increases the effective MIPS, but on the other hand, complicates the software development both for the programmer and compiler design.

A major factor in determining efficient utilization of multiple functional units is the method the instruction set is defined and encoded. In general there are two approaches with regard to the instruction word for the operation of multiple functional units:

Very Long Instruction Word (VLIW), as in the TI's TMS320C6x family.

Parallel Instruction Word (PIW) as in the DSP Group's PalmDSPCore or Lucent DSP16xxx.

The above mentioned two architectures are conceptually different. The following paragraphs will describe the main elements of the architectures with respect to the instruction set and the implication on the programmer.

VLIW Architecture

VLIW is an extension of the RISC philosophy. It is geared toward adding instruction level parallelism to a processor. It is composed of a set of design techniques that speeds up programs by executing in parallel several operations, such as memory loads and stores, additions, multiplication and more. These operations are taken from a single flow of execution, rather than from a parallel task. The parallelism is usually transparent to the user and it is the compiler's task to schedule the instruction flow into the parallel functional units.

TMS320C6x is the first signal processor to offer massively computational resources controlled by Very Long Instruction Word. It allows programming even of complicated mathematical functions in C, rather than in assembly language, thus ensuring program transportability and maintainability.

In VLIW type processor, the compiler has to translate a flow of instructions into a very long instruction word that will be executed in a single cycle. Controlling the machine resources via the compiler can be a mixed blessing. On one hand, it lets DSP developers do their work in C, ensuring transportability of software libraries and routines and facilitating code maintenance when design teams change. On the other hand, it introduces many challenges to the compiler design such as complicated instruction scheduling and parallel dispatching [3]. A loosely tuned compiler will undoubtedly waste machine cycles, power and code size.

PIW Architecture

A PIW DSP maintains the basic architecture of the conventional DSPs but with the additions of significant capabilities in the data path, instruction set and memory bandwidth.

A PIW DSP relies on complex instructions that can encode many parallel operations. It issues these

complex instructions at a maximum rate of one per cycle. The processor also includes dedicated hardware, provided for specific targeted applications such as Viterbi decoding. The data path, the dedicated hardware and the special complex instructions allow numerous operations to be encoded into single instructions and executed simultaneously.

In a PIW type DSP, a common way to increase performance through parallelism is to have a wider instruction word. For example, adding 32 bit instructions to PalmDSPCore architecture allowed DSP Group to increase the parallelism and to expand the instruction set compared to its previous generations, the TeakDSPCore and the OakDSPCore^{R}, which use mostly 16 bits instruction word.

Intuitively, increasing the size of the instruction word increases the overall code size. This due to the fact that complex instruction such as Dual MAC would need more bits to present it various options. Moreover, in order to simplify the design, simple instructions that do not have many fields to present its various options, such as no operation (NOP), would be encoded with longer instruction words.

However, having longer instruction word does not necessarily add to the overall code size. For example, sequences of instructions such as *add* followed by a *shift*, which used to occupy two single words in previous generation DSPs, can now be put together and occupy the same amount of code memory but will be executed in one cycle rather than in two cycles. Another example is the PalmDSPCore's Multi-Parallel-Instructions (MPI). These instructions consist of several single instructions, executed in a single cycle and encoded in two words (32-bit), thus achieving both code compactness and boost in performance.

In attempt to maintain a good code density while improving the per cycle performance, PIW type processors can support variable size instruction set. Both DSP Group's PalmDSPCore and Lucent DSP16xxx have a mixed instructions set of both 16- and 32-bit instructions.

A major challenge when defining a variable instruction word size, is to select the instructions that will be encoded in a single word and those that will be encoded in double word.

A few criteria have to be considered while assigning the code length of specific instruction:

Complexity. Instructions that operate several functional units require dedicated bit-fields for each unit.

Orthogonality. Instructions that require support for large amount of registers or addressing mode would need more bits to be encoded

Commonality. Instructions that are often used (e.g. MOV) will tend to be encoded with a single word rather than with double word.

Selection of the instruction set based on the above criteria has to be dealt with caution while optimizing the inner loops of the targeted applications to gain performance and the rest of the code for code compactness and ease of programming [3].

Conclusion

We discussed the new trend in DSP architecture of adding multiple computation units that are operated in parallel. This, on one hand, complicates the overall architecture of the DSP and its programming style. On the other hand, it significantly increases the applications' performance.

The current trend of executing several instructions in parallel gives a different meaning to what is defined as an instruction. The traditional metrics of MIPS (Million Instructions Per Second) and MOPS (Million Operations Per Second) have become less relevant as processor architecture diversify [4]. Different ways of definition and performance evaluation should be used.

References

[1] Faraboschi P., Desoli G., Fisher J. A., "The Latest Word in Digital and Media Processing", IEEE Signal Processing Magazine, March 1998, p 65.

[2] BDTI, Buyer's Guide to DSP Processors, Third Edition, 1997, pp 631, 718-719.

[3] Hennessy J. L., Patterson D., "Computer Architecture A Quantitative Approach", Second Edition, pp 80-89.

[4] EDN's 1998 DSP-Architecture Directory, EDN Magazine, April 1998, p 40.