INTRODUCTION TO DIGITAL SIGNAL PROCESSOR

By,

Snehal Gor
snehalg@embed.isquareit.ac.in

PURPOSE

"Purpose is deliberately thought-through goal-directedness."

- http://en.wikipedia.org/wiki/Purpose

This document contains introduction to Digital Signal Processor, how it differs from general purpose processor, what extra features it has and many other fundamentals. I have also included some introduction to Computer Architecture which may be helpful for students who don't have full course on Computer Architecture. I have given example of Analog devices SHARC architecture and Texas Instrument's DSP processor architecture to give you more insight. This document doesn't contain full description of any processor for more detail one can read manuals and datasheets from Analog Devices and Texas Instrument.

This document is a compilation of materials available from net, books and some of my own. I hope this document is able to give you enough insight into the architecture of Digital Signal Processor. I welcome your comments.

-Snehal Gor

INDEX

1. DIFFERENCE BETWEEN DIGITAL SIGNAL PROCESSOR AND GENERAL PURPOSE PROCESSOR

Computers are extremely capable in two broad areas: (1) *data manipulation*, such as word processing and database management, and (2) *mathematical calculation*, used in science, engineering, and Digital Signal Processing. However, most computers are not optimized to perform *both* functions. In computing applications such as word processing, data must be stored, sorted, compared, moved, etc., and the time to execute a particular instruction is not critical, as long as the program's overall response time to various commands and operations is adequate enough to satisfy the end user. Occasionally, mathematical operations may also be performed, as in a spreadsheet or database program, but speed of execution is generally not the governing factor. In most general purpose computing applications there is no concentrated attempt by software companies to make the code efficient. Application programs are loaded with "features" which require more memory and faster processors with every new release or upgrade.

On the other hand, digital signal processing applications require that mathematical operations be performed quickly, and the time to execute a given instruction must be known precisely, and it must be predictable. Both code and hardware must be extremely efficient to accomplish this. The most fundamental mathematical operation or *kernel* in all of DSP is the sum-of-products (or *dot-product*). Fast execution of the dot product is critical to fast Fourier transforms (FFTs), real time digital filters, matrix multiplications, graphics pixel manipulation, etc.

Microprocessors, such as the Pentium-series from Intel, are basically single-chip CPUs which require additional circuitry to make up the total computing function. Microprocessor instruction sets can be either complex-instruction-set computer (CISC) or reduced-instruction-set computer (RISC). The complex-instruction-set computer (CISC) includes instructions for basic processor operations, plus single instructions that are highly sophisticated; for example, to evaluate a high-order polynomial. But CISC has a

price: many of the instructions execute via microcode in the CPU and require numerous clock cycles plus silicon real estate for code storage memory.

In contrast, the reduced-instruction-set computer (RISC) recognizes that, in many applications, basic instructions such as LOAD and STORE - with simple addressing schemes - are used much more frequently than the advanced instructions, and should not incur an execution penalty. These simpler instructions are hardwired in the CPU logic to execute in a single clock cycle, reducing execution time and CPU complexity.

Although the RISC approach offers many advantages in general purpose computing, it is not well suited to DSP. For example, most RISCs do not support single instruction multiplication, a very common and repetitive operation in DSP. The DSP is optimized to accomplish these tasks fast enough to maintain real-time operation in the context of the application. This requires single-cycle arithmetic operations and accumulations.

2. BASIC TERMINOLOGIES

As we all know from the days of our graduation there is two types of basic architectures:

- ➔ Von Neumann Architecture
- ➔ Harvard Architecture

Basic operation involved in any processor is transferring information to and from memory. This includes *data*, such as samples from the input signal and the filter coefficients, as well as *program instructions*, the binary codes that go into the program sequencer. For example, suppose we need to multiply two numbers that reside somewhere in memory. To do this, we must fetch three binary values from memory, the numbers to be multiplied, plus the program instruction describing what to do.

2.1 Von Neumann Architecture

Figure 1 shows how above seemingly simple task is done in a traditional microprocessor. This is often called **Von Neumann architecture**, after the brilliant American mathematician John Von Neumann.

As shown in fig 1, Von Neumann architecture contains a single memory and a single bus for transferring data into and out of the central processing unit (CPU). Multiplying two numbers requires at least three clock cycles, one to transfer each of the three numbers over the bus from the memory to the CPU. The Von Neumann design is quite satisfactory when you are content to execute all of the required tasks in serial. We only need other architectures when very fast processing is required, and we are willing to pay the price of increased complexity.

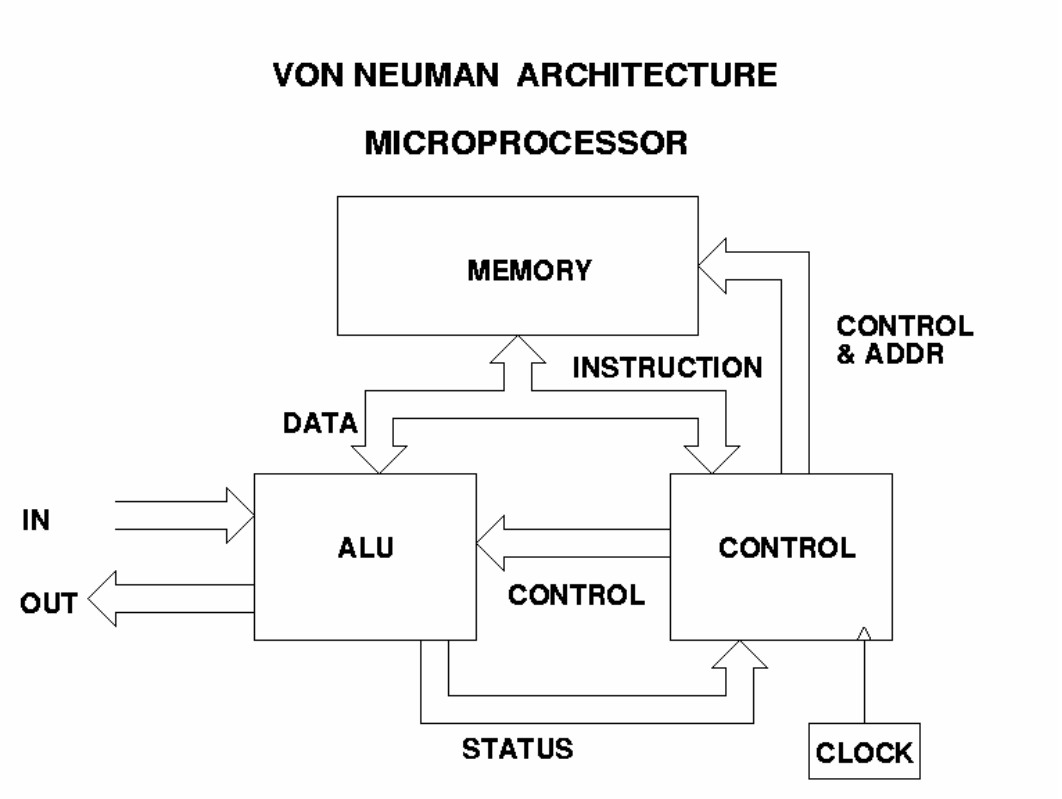

FIG 1 VON NEUMANN ARCHITECTURE

2.2 Harvard Architecture

This leads us to the **Harvard architecture**, shown in figure 2. This is named for the work done at Harvard University in the 1940s under the leadership of Howard Aiken. As shown in this illustration, Aiken insisted on separate memories for data and program instructions, with separate buses for each. Since the buses operate independently, program instructions and data can be fetched at the same time, improving the speed over the single bus design. Most present day DSPs use this dual bus architecture.

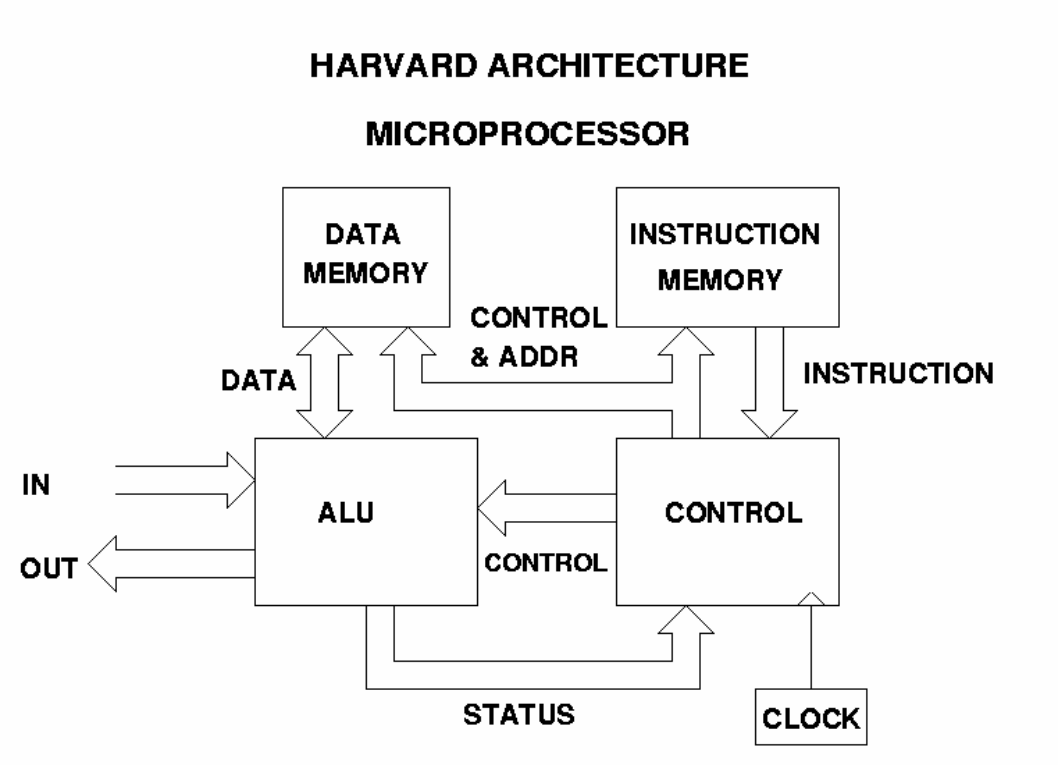

FIG 2 HARVARD ARCHITECTURE

2.2.1 SHARC Architecture

Have you people heard of DSP processor family SHARC from Analog devices; it uses modified form of Harvard Architecture, the **Super Harvard Architecture**. This term was coined by Analog Devices to describe the internal operation of their ADSP-2106x and new ADSP-211xx families of Digital Signal Processors. These are called **SHARC** DSPs, a contraction of the longer term, Super Harvard ARChitecture. The idea is to build upon the Harvard architecture by adding features to improve the throughput. While the SHARC DSPs are optimized in dozens of ways, two areas are important enough to be included in Fig. 3: an *instruction cache*, and an *I/O controller*.

Instruction Cache

Cache: a safe place for hiding or storing things.

Webster's New World Dictionary of the American Language, Second College Edition (1976)

Cache is the name given to the first level of the memory hierarchy encountered once the address leaves the CPU. Since the principle of locality applies at many levels, and taking advantage of locality to improve performance is popular, the term *cache* is now applied whenever buffering is employed to reuse commonly occurring items. Examples include *file caches, name caches,* and so on.

Since, DSP algorithms generally spend most of their execution time in loops; we can take advantage of it as per principle of locality. This means that the same set of program instructions will continually pass from program memory to the CPU. The Super Harvard architecture takes advantage of this situation by including an **instruction cache** in the CPU. This is a small memory that contains about 32 of the most recent program instructions. The first time through a loop, the program instructions must be passed over the program memory bus. This results in slower operation because of the conflict with the coefficients that must also be fetched along this path. However, on additional executions of the loop, the program instructions can be pulled from the instruction cache. This means that all of the memory to CPU information transfers can be accomplished in a single cycle: the sample from the input signal comes over the data memory bus, the coefficient comes over the program memory bus, and the program instruction comes from the instruction cache.

I/O controller

Figure 3 shows the **I/O controller** connected to data memory. This is how the signals enter and exit the system. For instance, the SHARC DSPs provides both serial and parallel communications ports. These are extremely high speed connections.

Dedicated hardware allows data streams to be transferred directly into memory (Direct Memory Access, or DMA) from above ports, without having to pass through the CPU's registers. No cycles are stolen from the CPU.

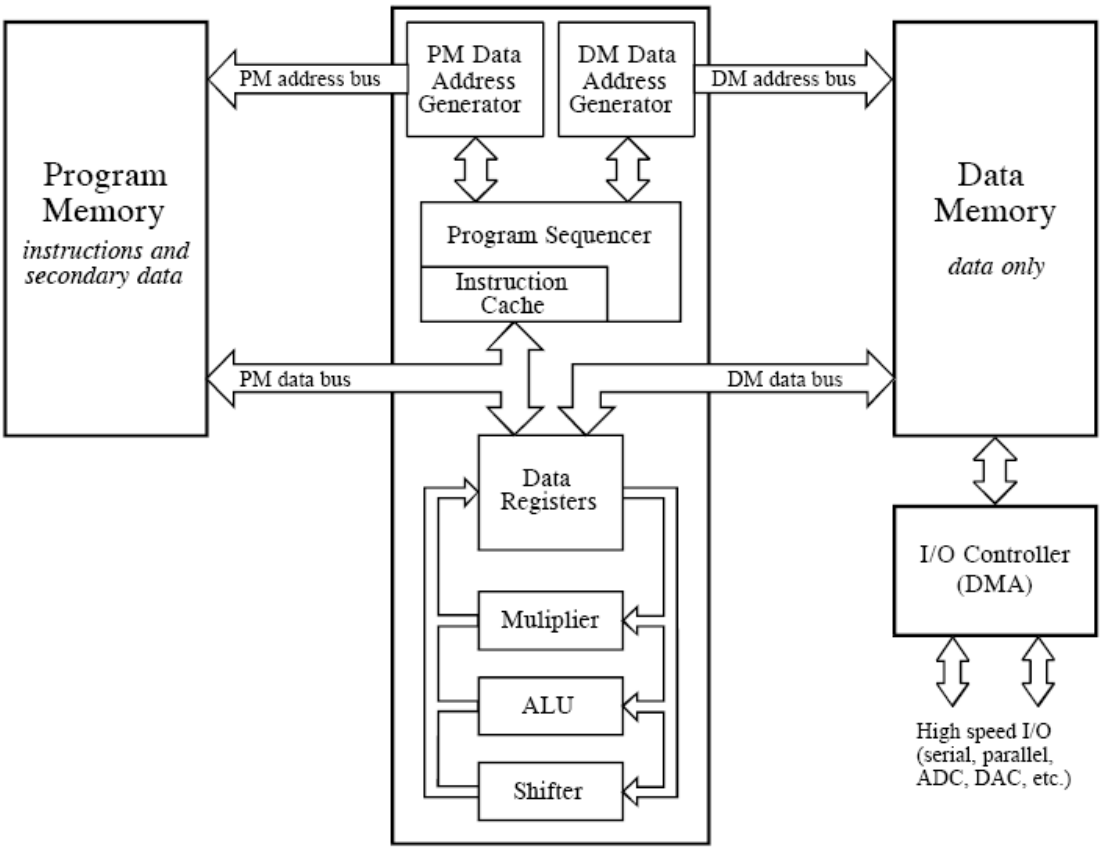

FIG 3 SIMPLIFIED DIAGRAM IS OF THE ANALOG DEVICES SHARC DSP.

2.2.2 TI's Architecture

TI's Architecture also contains similar features. TMS320C67x series has 32K-Bit (4K-Byte) L1P Program Cache (Direct Mapped), 32K-Bit (4K-Byte) L1D Data Cache (2-Way Set-Associative), 512K-Bit (64K-Byte) L2 Unified Mapped RAM/Cache (Flexible Data/Program Allocation). Similarly it has Enhanced DMA (EDMA) support which supports 16 independent channels.

2.3 Pipelining, Superscalar and VLIW architecture

In this subsection I have included some basic guidelines to pipelining, Superscalar architecture and VLIW (Very long Instruction word) Architecture.

2.3.1 Pipelining

Pipelining is an implementation technique whereby multiple instructions are overlapped in execution; it takes advantage of parallelism that exists among the actions needed to execute an instruction.

A pipeline is like assembly line. In an automobile assembly line, there are many steps each contributing something to the construction of the car. Each step operates in parallel to the other steps, although on different car. In a computer pipeline each step in the pipeline completes part of an instruction. Like an assembly line different steps are completing different parts of different instructions in parallel. Each of this step is called pipe stage or pipe segment. Fig 4 shows classical five stage RISC pipeline.

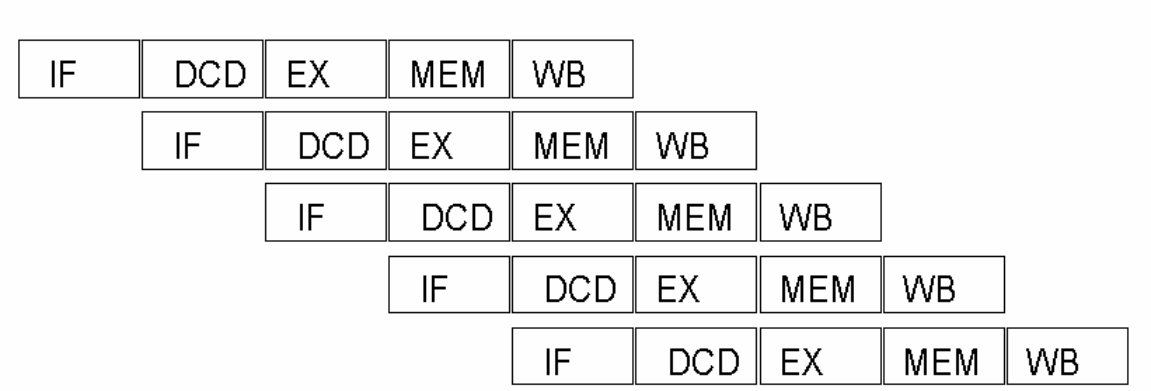

FIG 4 SIMPLE RISC PIPELINE. On each clock cycle, another instruction is fetched and begins its 5-cycle execution. If an instruction is started every clock cycle, the performance will be up to five times that of processor that is not pipelined. Remember each row represents one instruction and each block is one pipeline stage: IF = Instruction Fetch, ID = Instruction Decode, EX = Execution, MEM = Memory Access and WB = Write Back.

As one can see from figure speedup is initially equal to number of pipe stages but deteriorates after words, due to pipeline hazard. Discussion of pipeline hazard is beyond the scope of this document.

2.3.2 Super Scalar and VLIW approach to exploit Instruction level Parallelism

Pipelining as explained in previous section along with techniques to eliminate data and control stalls used to achieve an ideal CPI(Clock cycles per instruction) of 1. To improve performance further we would like to decrease the CPI to less than one. But the CPI cannot be reduced below one if we issue only one instruction every clock cycle.

The goal of the **multiple-issue processors** is to allow multiple instructions to issue in a clock cycle. Multiple-issue processors come in two basic flavors: **superscalar** processors and **VLIW** (very long instruction word) processors.

Superscalar processors issue varying numbers of instructions per clock and are either statically scheduled (using compiler techniques) or dynamically scheduled (using techniques based on Tomasulo's algorithm). Statically scheduled processor use in-order execution, while dynamically scheduled processors use out-of-order execution. Superscalar processors decide on the fly how many instructions to issue. A statically scheduled superscalar must check for any dependence between instructions in the issue packet as well as between any issue candidate and any instruction already in the pipeline. A statically scheduled superscalar requires significant compiler assistance to achieve good performance. In contrast, a dynamically-scheduled superscalar requires less compiler assistance, but has significant hardware costs.

VLIW processors, in contrast, issue a fixed number of instructions formatted either as one large instruction or as a fixed instruction packet with the parallelism among instructions explicitly indicated by the instruction (hence, they are also known as EPIC--Explicitly Parallel Instruction Computers). VLIW and EPIC processors are inherently statically scheduled by the compiler. Such an approach offers the potential advantage of simpler hardware while still exhibiting good performance through extensive compiler optimization.

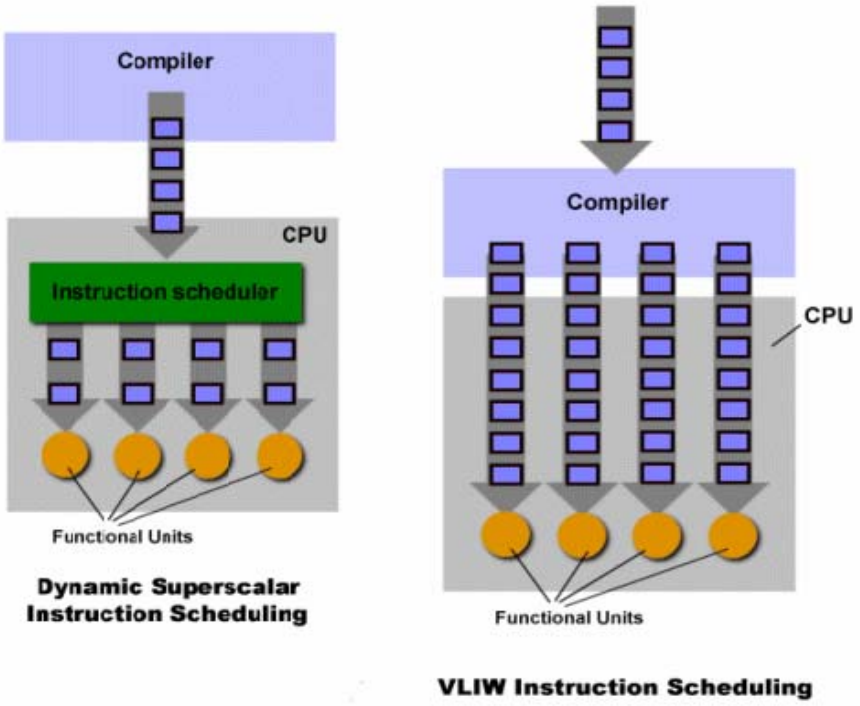

FIG 5 CONCEPTUAL VIEW OF SCHEDULING IN SUPERSCALAR AND VLIW APPROACH

TI TMS320C67x Series

Let's see how TI's 62x and 67x series CPU core has been designed. The CPU fetches **VelociTI** advanced very-long instruction words (**VLIW**) (256 bits wide) to supply up to eight 32-bit instructions to the eight functional units during every clock cycle. The VelociTI VLIW architecture features controls by which all eight units do not have to be supplied with instructions if they are not ready to execute. The first bit of every 32-bit instruction determines if the next instruction belongs to the same execute packet as the previous instruction, or whether it should be executed in the following clock as a part of the next execute packet. Fetch packets are always 256 bits wide; however, the execute packets can vary in size. The variable-length execute packets are a key memory-saving feature, distinguishing the C67x CPU from other VLIW architectures.

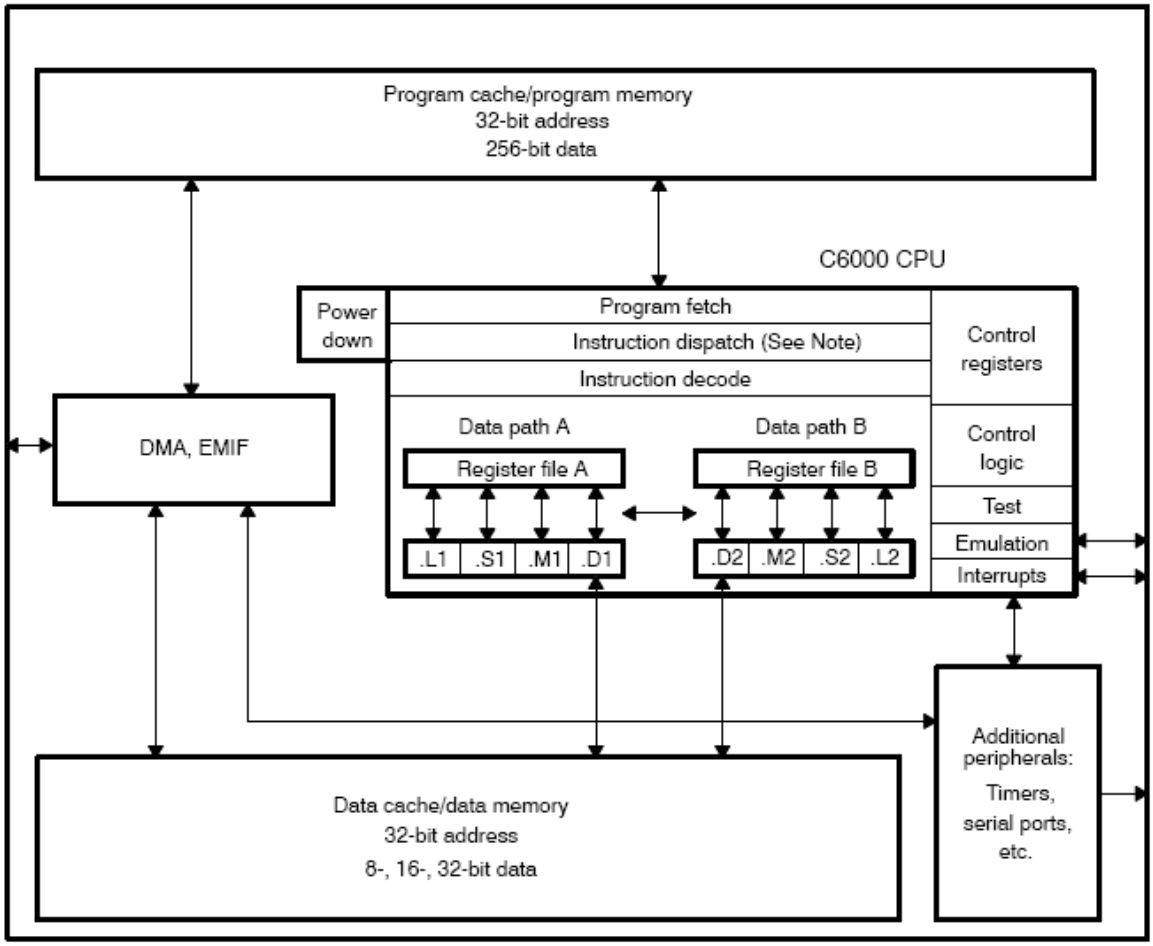

FIG 6 TMS320C67x DSP BLOCK DIAGRAM

The CPU features two sets of functional units. Each set contains four units and a register file. One set contains functional units .L1, .S1, .M1, and .D1; the other set contains units .D2, .M2, .S2, and .L2. The two register files each contain 16 32-bit registers for a total of 32 general-purpose registers. The two sets of functional units, along with two register files, compose sides A and B of the CPU (see the functional block diagram Figure 6). The four functional units on each side of the CPU can freely share the 16 registers belonging to that side. Additionally, each side features a single data bus connected to all the registers on the other side, by which the two sets of functional units can access data from the register files on the opposite side. While register access by functional units on the same side of the CPU as the register file can service all the units in a single clock cycle, register access using the register file across the CPU supports one read and one write per cycle.

Another key feature of the C67x CPU is the load/store architecture, where all instructions operate on registers (as opposed to data in memory). Two sets of data-addressing units (.D1 and .D2) are responsible for all data transfers between the register files and the memory. The data address driven by the .D units allows data addresses generated from one register file to be used to load or store data to or from the other register file. The two .M functional units are dedicated for multiplies. The two .S and .L functional units perform a general set of arithmetic, logical, and branch functions with results available every clock cycle.

3. IMPORTANT FEATURES

This section contains some important features of DSP processor. Examples have been taken from Analog DSP SHARC architecture.

EXTENDED PRECISION

Apart from the obvious need for fast multiplication and addition (MAC), there is also a requirement for extended precision in the accumulator register. For example, when two 16-bit words are multiplied, the result is a 32-bit word. The Analog Devices ADSP-21xx 16-bit fixed-point core architecture has an internal 40-bit accumulator which provides a high degree of overflow protection. While floating point DSPs eliminate most of the problems associated with precision and overflow, fixed-point processors are still popular for many applications, and therefore overflow, underflow, and data scaling issues must be dealt with properly.

DUAL OPERAND FETCH

Regardless of the nature of a processor, performance limitations are generally based on bus bandwidth. In the case of general purpose microprocessors or microcontrollers, code is dominated by single memory fetch instructions, usually addressed as a base plus offset value. This leads architects to embed fixed data into the instruction set so that this class of memory access is fast and memory efficient. DSPs, on the other hand, are dominated by instructions requiring two independent memory fetches. This is driven by the basic form of the convolution (dot product) $\sum h(i)x(i)$. The goal of fast dual operand fetches is to keep the MAC fully loaded. Dual operand fetch is implemented in DSPs by providing separate buses for program memory data and data memory data. In addition, separate program memory address and data memory address buses are also provided. The MAC can therefore receive inputs from each data bus simultaneously. This architecture is often referred to as the Harvard Architecture (explained above).

CIRCULAR BUFFERING

If we examine the convolution equation more carefully, the advantages of circular buffering in DSP applications become apparent. A Finite Impulse Response (FIR) filter is used to demonstrate the point.

$$y[n] = a_0 x[n] + a_1 x[n-1] + a_2 x[n-2] + ...$$

First, coefficients or tap values for FIR filters are periodic in nature. Second, the FIR filter uses the newest real-world signal value and discards the oldest value when calculating each output. In the series of FIR filter equations, the N coefficient locations are always accessed sequentially from a_0 to a_{N-1}. The associated data points circulate through the memory as follows: new samples are stored replacing the oldest data each time a filter output is computed. A fixed boundary RAM can be used to achieve this circulating buffer effect. The oldest data sample is replaced by the newest with each convolution. A "time history" of the N most recent samples is kept in RAM. This delay line can be implemented in fixed boundary RAM in a DSP chip if new data values are written into memory, overwriting the oldest value. To facilitate memory addressing, old data values are read from memory starting with the value one location after the value that was just written. In a 4-tap FIR filter, for example, x(4) is written into memory location 0, and data values are then read from locations 1, 2, 3, and 0. This example can be expanded to accommodate any number of taps. By addressing data memory locations in this manner, the address generator need only supply sequential addresses regardless of whether the operation is a memory read or write. This data memory buffer is called *circular* because when the last location is reached, the memory pointer must be reset to the beginning of the buffer. The coefficients are fetched simultaneously with the data. Due to the addressing scheme chosen, the oldest data sample is fetched first. Therefore, the last coefficient must be fetched first. The coefficients can be stored backwards in memory: a_{N-1} is the first location, and a_0 is the last, with the address generator providing incremental addresses. This allows direct support of the FIR filter unit delay taps without software overhead. These data characteristics are DSP algorithm-specific and must be supported in hardware to achieve the best DSP performance. Implementing circular buffers in hardware allows buffer parameters (i.e. start, length, etc.) to be set up

outside of the core instruction loop. This eliminates the need for extra instructions within the loop body. Lack of a hardware implementation for circular buffering can significantly impact MAC performance.

SHADOW REGISTER

Another feature is the use of **shadow registers** for all the CPU's key registers. These are duplicate registers that can be switched with their counterparts in a single clock cycle. They are used for *fast context switching*; the ability to handle interrupts quickly. When an interrupt occurs in traditional microprocessors, all the internal data must be saved before the interrupt can be handled. This usually involves pushing all of the occupied registers onto the stack, one at a time. In comparison, an interrupt in the SHARC family is handled by moving the internal data into the shadow registers in a *single clock cycle*. When the interrupt routine is completed, the registers are just as quickly restored.

DSP OPERATIONS

DSPs also provide operations for data transfer, control and arithmetic and logical, but they change the semantics a bit. First, because they are often used in real time applications, there is not an option of causing an exception on arithmetic overflow (otherwise it could miss an event); thus, the result will be used no matter what the inputs. To support such an unyielding environment, DSP architectures use **saturating arithmetic**: if the result is too large to be represented, it is set to the largest representable number, depending on the sign of the result. In contrast, two’s complement arithmetic can add a small positive number to a large positive number and end up with a negative result. DSP algorithms rely on saturating arithmetic, and would be incorrect if run on a computer without it.

A second issue for DSPs is that there are several modes to round the wider accumulators into the narrower data words, just as the IEEE 754 has several rounding modes to choose from.

Finally, the targeted kernels for DSPs accumulate a series of products, and hence have a **multiply-accumulate or MAC instruction**. MACs are key to dot product operations for vector and matrix multiplies. In fact, MACs/second is the primary peak-performance metric that DSP architects brag about. The wide accumulators are used primarily to accumulate products, with rounding used when transferring results to memory.

REFERENCES:

1. John L. Hennessy & David A. Patterson, Computer Architecture: A Quantitative Approach, Third Edition, Morgan Kaufmann Publishers.
2. Steven W. Smith, The Scientist and Engineer's Guide to Digital Signal Processing, Second Edition, California Technical Publishing.
   www.dspguide.com
3. Mixed-Signal and DSP Design Techniques, Analog Devices
   www.analog.com
4. ADSP-2100 Family Users Manual
   www.analog.com
5. TMS320C67x series User Manual
   www.ti.com
6. TI DSP tutorials by Dr. Naim Dahnoun, University of Bristol, UK