CPE 512 Godwin Moni Oghenero,

INTRODUCTION

To run high quality multimedia applications with sound and graphics requires very fast and sophisticated mathematical operations, such complex operations are normally performed by a highly specialized chip called DSP (digital signal processing). DSP chips are the main engines performing task such as 2 and 3-D graphics, video and audio compression, fax/modem, PC-based telephoning with live pictures and image processing.

There are three approaches to equip the PC or an embedded system with DSP capability.

1. Use a full-fledged DSP chip on the board along with the main CPU. This is the best and ideal approach since there are some very powerful DSP chip out there. However, the problem is that there is no industry-wide standard to be followed and the lack of such standard can lead to incompatibility both in hardware and software.

2. Use CPU x86 and CPU x87(floating) instructions to emulate the function of DSP. This is slow and performance is unacceptable.

3. The third approach is to incorporate some DSP function into the general microprocess. This approach leaves everyone at the mercy of the vendor, yet it brings compatibility and a unified approach to the issue. Although the performance is not as good as the first approach, it is much better than the second approach. The third approach is exactly what happened in early 1997, intel introduced a series of Pentium and Pentium pro chips with somewhat limited DSP capability called MMX technology, software compatibility, both on the BIOS and operating system levels, was the most important goal. It needs to be noted that although MMX does not have a rich set of instruction normally associated with DSP chips such as Texas instruments 320xxx, it still performs many of the DSP functions reasonably well.

DEFINITION OF DSP

Digital Signal Processors (DSP) take real-world signals like voice, audio, video, temperature, pressure, or position that have been digitized and then mathematically manipulate them. A DSP is designed for performing mathematical functions like "add", "subtract", "multiply" and "divide" very quickly.

Signals need to be processed so that the information that they contain can be displayed, analyzed, or converted to another type of signal that may be of use. In the real-world, analogue products detect signals such as sound, light, temperature or pressure and manipulate them. Converters such as an Analog-to-Digital converter then take the real-world signal and turn it into the digital format of 1's and 0's. From here, the DSP takes over by capturing the digitized information and processing it. It then feeds the digitized information back for use in the real world. It does this in one of two ways, either digitally or in an analogue format by going through a Digital-to-Analog converter. All of this occurs at very high speeds.

Digital signal processing is different from other engineering field in terms of distinguishing feature, signal as a data type. Signal data in most of time is real word signal data obtain from sensor or transducers which is in form of speech, image, vibration, sound, seismic etc.

To process this data we need mathematical operation, algorithms and some special techniques used to manipulate the signal i.e. for enhancement, compression, modulation, conversion, etc. In the decades of 1960 and 1970, this field is limited to some specific task such as security, space,

application in which data is irreplaceable and where cost is not matter. After 1990 the DSP is used for commercial application such as smart phone, CD players, voice devices, cameras and many types of such portable and processing devices. Now day's different fields are dominated by DSP i.e. Space, Medical, Telephone, Security, Communication, Audio & Video applications, Vision System, Video game etc.

DSP processors are more efficient than microprocessors in terms of power efficiency due to which they are faster and suitable in portable devices such as mobile phones, PDA and different IT peripherals. These processors use distinct memory architecture which able to fetch multiple data and instructions in parallel way. This processor provides low-cost solution with good performance with lower latency, no requirement for specialized cooling or large batteries. The architecture of such processors is specially designed and elevates to handle discrete signals and their Processing with effectively less power consumption.

Figure 1 shows the different footsteps involve in the digital signal Processing of any real time signal which is analog in nature. Analog signals are generally audio, video or similar type of analog signals. This type of signals converted in form of digital signal using analog to digital conversion.

Fig. 1

DSPs appeared on the market in the early 1980s. Over the last 15 years they have been the key enabling technology for many electronics products in fields such as communication systems, multimedia, automotive, instrumentation and military. Table 1 gives an overview of some of these fields and of the corresponding typical DSP applications.

Figure 2 shows a real-life DSP application, namely the use of a Texas Instruments (TI) DSP in a MP3 voice recorder–player. The DSP implements the audio and encode functions. Additional tasks carried out are file management, user interface control, and post-processing algorithms such as equalization and bass management.

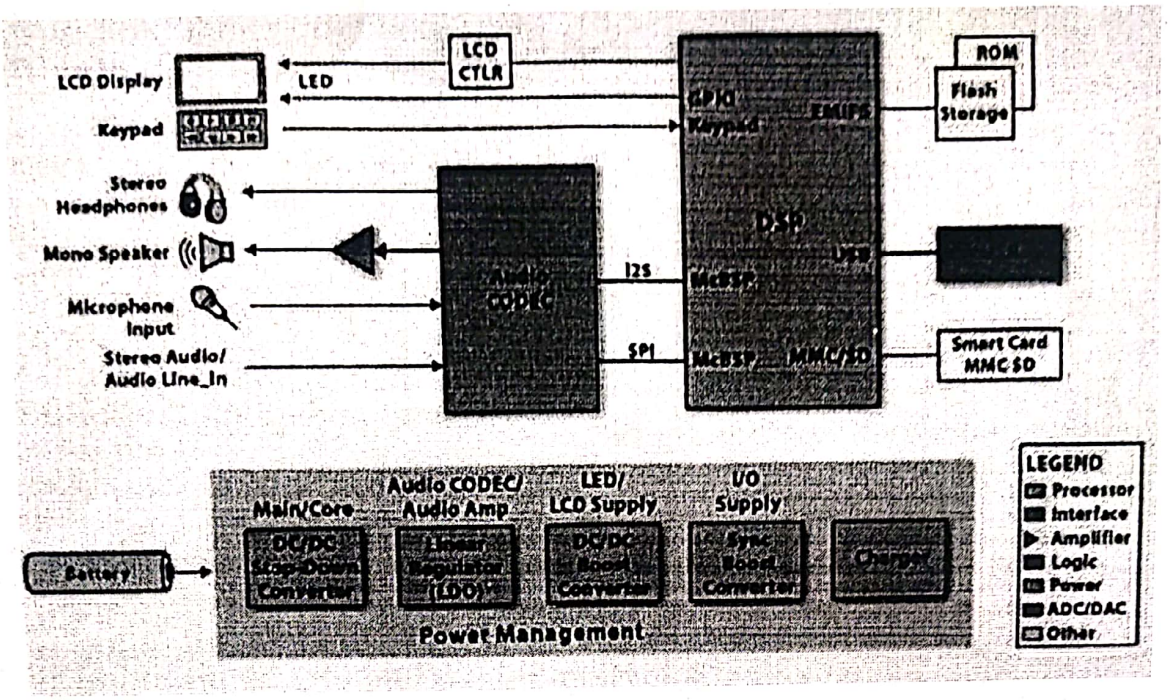


Fig. 2: Use of Texas Instruments DSP in a MP3 player/recorder system. Picture courtesy of Texas Instruments.

To illustrate this concept, the diagram above shows how a DSP is used in an MP3 player. During the recording phase, analog audio is input through a receiver or other source. This analog signal is then converted to a digital by ADC and passed to the DSP performs the MP3 encoding and save the file to memory. During the playback phase, the file is taken from memory, decoded by the DSP and then converted back to analog signal through DAC so it can output through the speaker system. in a more complex example, as volume control, equalization and user interface. A DSP's information can be used by a computer to control such things as security, telephone, home theater systems and video compression signals maybe compressed so that they can be transmitted quickly and more efficiently from one place to another (eg teleconferencing can transmit speech and video via telephone lines).

Signals may also be enhanced or manipulated to improve their quality or provide information that is not sensed by humans (eg echo cancellation for cell phones or computer-enhanced medical images).

Although, real world signals can be processed in their analog form, processing signals digitally provide the advantage of high speed and accuracy. Because it is programmable, a DSP can be used in a wide variety of applications. You can create your own software or use software provided by AD1 (...) and its third parties to design a DSP solution for an application.

In order for an ADC to provide enough samples to accurately describe the real-world signal, the sampling rate must be at least twice the highest frequency component of the analog signal. For example, to accurately describe an audio signal containing frequencies up to 20kHz, the ADC must sample the signal at a minimum of 40KHz. Since arriving signal can easily contain component frequencies above 20KHz (including noise), they must be removed before sampling

by feeding the signal through a low pass filter ahead of the ADC. This filter, known as anti-aliasing filter, is intended to remove the frequency above 20KHz that could corrupt the converted signal.

However, the anti-aliasing filter has a finite frequency roll-off, so additional bandwidth must be provided for the filters transition band. For example, for an input of about 20KHz, one might allow 2 to 4KHz of extra bandwidth. Therefore, the anti-aliasing will reject any frequency above 24KHz, ADC= 48KHz.

One way to achieve this level of rejection without a highly sophisticated analog filter is to use an over sampling converter, such as a sigma-delta ADC. It typically obtains low-resolution (e.g 1-bit) samples at megahertz rate much faster than twice the highest frequencies. An internal digital filter (DSP at work!) restores the required resolution and frequency response. For many applications, oversampling converters reduce system design effort and cost.

The ADC sampling rate depends on the bandwidth of the analog signal being sampled. This sampling rate sets the pace at which samples are available for processing. Once the system bandwidth requirement has established the A/D converter sampling rate, the designer can begin to explore the speed requirement of the DSP processor. The processing speed at a required sample rate is influenced by algorithm complexity. As a rule, the DSP needs to finish all operations relating to the first sample before receiving the second sample. The timing between the samples is the time budget for DSP to perform all the processing tasks. For audio example, a 48KHz sampling rate corresponds to a 20.833 microseconds sampling interval.

Consider the relationship between the speed of the DSP and complexity of the algorithm (software containing the transform or other set of numeric operations). Complex algorithm requires more processing tasks. Note that because the time between sampling is fixed, the higher complexity calls for faster processing.

For example, suppose that the algorithm requires 50 processing operations to be performed between samples. Using the previous example, 48KHz sample rate (20.833 microseconds sampling interval), one can calculate the minimum required DSP processor speed, in millions of operations per seconds (MOPS) as follows:

DSP speed = operations/ sampling interval = 50/20.833 = 2.4 MOPS

Thus, if all of the time between samples is available for operations to implement the algorithm, a processor with a performance level of 2.4MOPS is required. Note that the two common ratings for DSPs, based on operation per seconds (MOPS) and instruction per second (MIPS) are not the same. A processor with a 10-MIPS rating can perform 8 operations per instruction has basically the same performance as a faster processor with 40MIPS rating that can only perform 2 operations per instruction.

How to acquire data?

There are two basic ways to acquire data, either one sample at a time or one frame at a time (continuous processing or vs batch processing).

1. Sample-based system, like a digital filter, acquire data one sample at a time. As shown below. Diagram

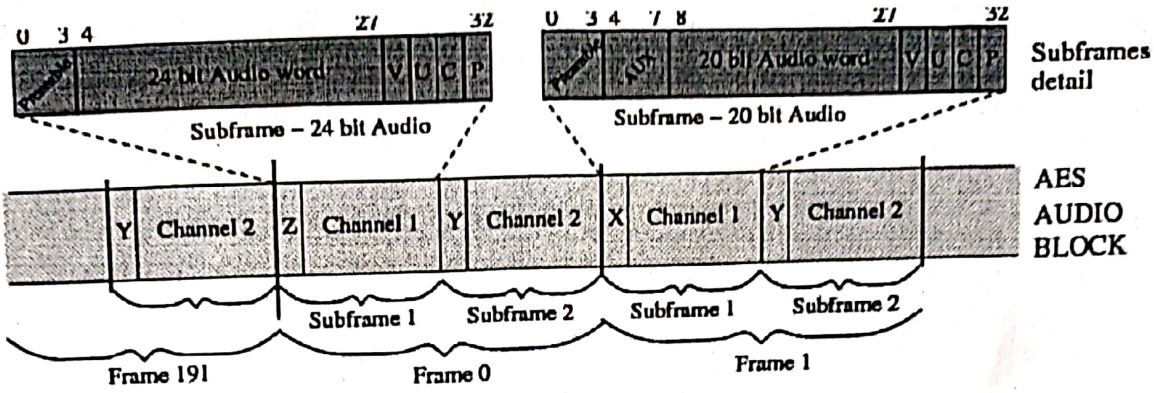


Fig.

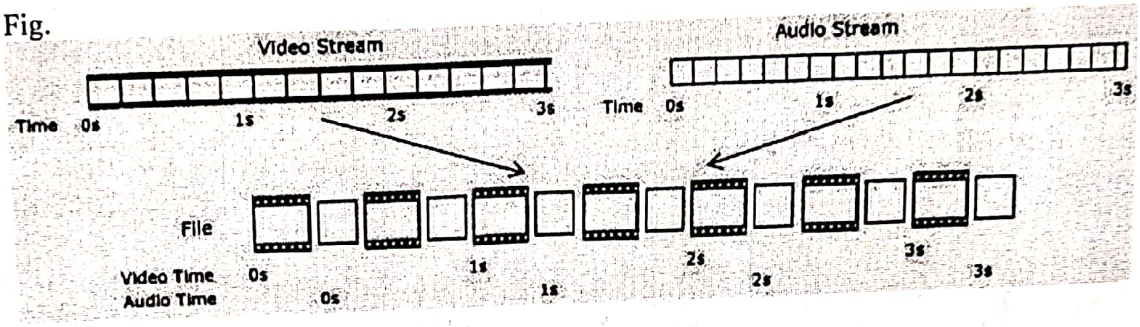


Fig.

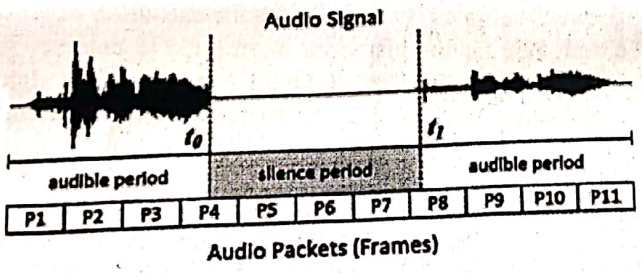


Fig.

At each tick of the clock, a sample comes into the systems and a processed sample is output waveform develops continuously.

2. Frame-based system, like a spectrum analyzer, which determines the frequency components of a time-varying waveform, acquire a frame (or block of samples) processing occurs on the entire frame of data and results in a frame of transformed data as shown below:

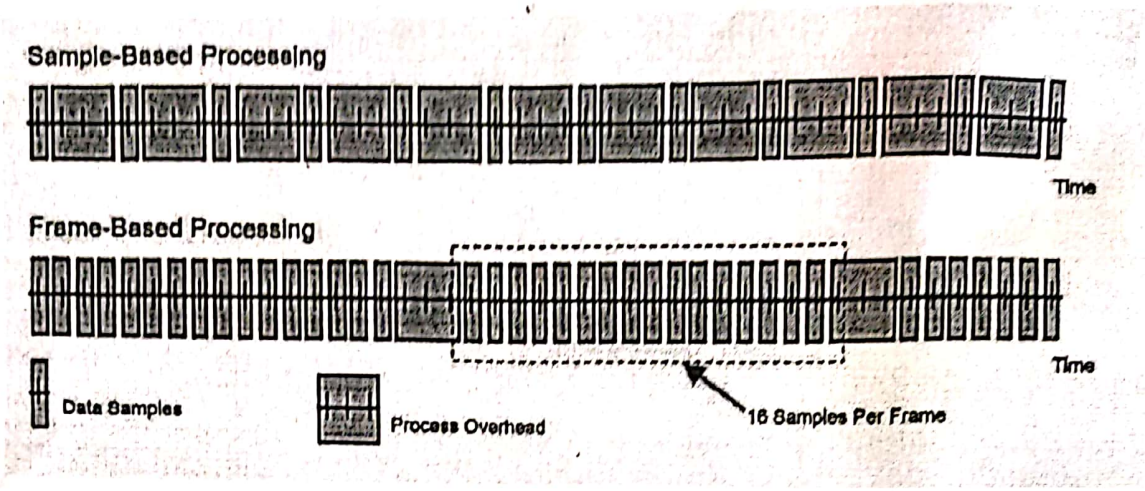


Fig.9

For an audio sampling rate of 48KHz, a processor working on a frame of 1042 samples have a frame acquisition interval of 21.22ms (ie 1024*20.833 microseconds). Here, the DSP has 21.33ms to complete all the required processing task for that frame of data. If the system handles signals in real time, it must not lose data; so, while the DSP is processing the first frame, it must also be acquiring the second frame.

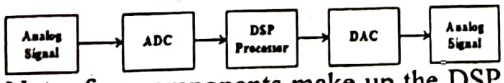
Acquiring the data is one area where special architectural features of DSPs come into play: seamless data acquisition is facilitated by a processor's flexible data addressing capability in conjunction with its direct memory accessing (DMA) channels.

One cannot assume that all the time between samples is available for the execution of processing instructions. In reality, time must be budgeted for the processor to respond to external devices, controlling the flow of data in and out. Typically, an external device (such as ADC) signals the processor using an interrupt or interrupt latency, directly influences how much time remains for actual signal processing.

Interrupt latency (response delay) depends on several factors, the most dominant pipelining. An instruction pipelining consists of number of instruction cycles that occur between the time an interrupt is received and the time that program execution resumes. More pipelining levels in a DSP result in longer interrupt latency. For example, if a processor has 20-ns cycle time and require 10 cycles to respond to an interrupt, 200ns elapse before it executes any signal processing instruction.

Developing a DSP system

Building block of a typical DSP system that could be used for data acquisition and control.



Note: few components make up the DSP system, because so much of the systems functionality comes from the programmable DSP.

A/D converters funnel data in and out of the DSP; the ADC timing is controlled by a precise sample clock. To simplify system design, many converter devices available today combine or all of the following (known as CODEC): An A/D converter, D/A converter, a sampling clock and

filters for anti-aliasing and anti- imaging. The clock oscillator in these types of I/O components is separately controlled by an external crystal.

Analog input: the analog input is appropriately band limited by the anti-aliasing filter and applied to the input of ADC. At the selected sampling time, the converter interrupts the DSP processor and makes the digital samples available. The choice between series and parallel interfacing between ADC and DSP depends on the amount of data, design complexity trade-offs, space, power and price.

Digital signal processing: the incoming data is handled by the DSP's algorithm software. Which the processor completes the required calculations, it sends the result to the DAC. Because the signal processing is programmable, considerable flexibility is available in handling the data and improving systems performance with incremental programming adjustments.

Analog output: the converters output is smoothed by a low-pass, anti-imaging filter (also called a reconstruction filter) to produce the reconstructed analog signal.

DSP APPLICATIONS

Table 1: A short selection of DSP fields of use and specific applications

Field		Application
Communication	Broadband	Video conferencing / phone
		Voice / multimedia over IP
		Digital media gateways (VOD)
	Wireless	Satellite phone
		Base station
Consumer	Security	Biometrics
		Video surveillance
	Entertainment	Digital still /video camera
		Digital radio
		Portable media player / entertainment console
	Toy s	Interactive toys
		Video game console
Industrial and entertainment	Medical	MRI
		Ultrasound
		X-ray
	Point of sale	Scanner
		Vending machine
	Industrial	Factory automation
		Industrial / machine / motor control
		Vision system
Military and aerospace		Guidance (radar, sonar)
		Avionics
		Digital radio
		Smart munitions, target detection

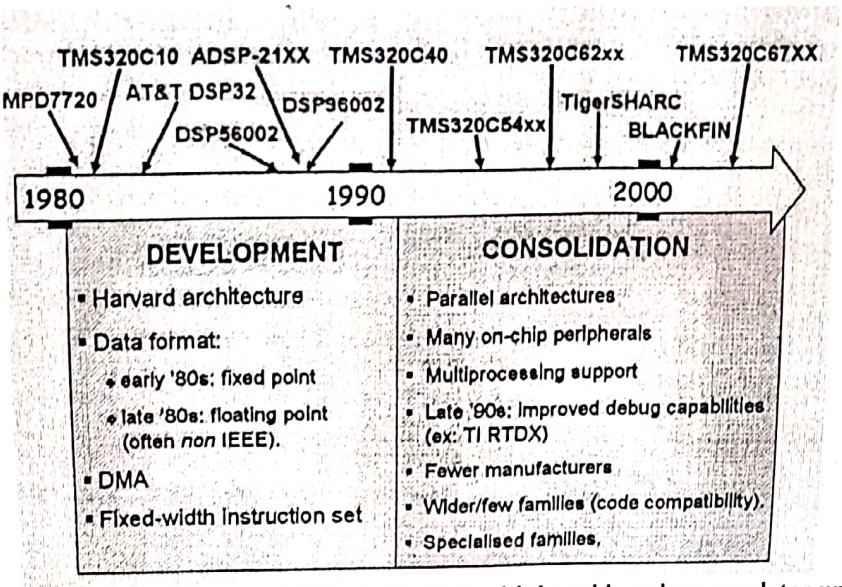
DSP evolution: hardware features

In the late 1970s there were many chips aimed at digital signal processing; however, they are not considered to be digital signal processing owing to either their limited programmability or their lack of hardware features such as hardware multipliers. The first marketed chip to qualify as a programmable DSP was NEC's MPD7720, in 1981: it had a hardware multiplier and adopted the Harvard architecture. Another early DSP was the TMS320C10, marketed by TI in 1982.

Figure 5 shows a selective chronological list of DSPs that have been marketed from the early 1980s until now. From a market evolution viewpoint, we can divide the two and a half decades of DSP life span into two phases: a development phase, which lasted until the early 1990s, and a consolidation phase, lasting until now. Figure 5 gives an overview of the evolution of DSP features together with the first year of marketing for some DSP families.

Fig. 5: Evolution of DSP features from their early days until now. The first year of marketing is indicated at the top for some DSP families.

During the market development phase, DSPs were typically based upon the Harvard architecture.



The first generation of DSPs included multiply, add, and accumulator units. Examples are TI's TMS320C10 and Analog Devices' (ADI) ADSP-2101. The second generation of DSPs retained the architectural structure of the first generation but added features such as pipelining, multiple arithmetic units, special address generator units, and Direct Memory Access (DMA). Examples include TI's TMS320C20 and Motorola's DSP56002. While the first DSPs were capable of fixed-point operations only, towards the end of the 1980s DSPs with floating point capabilities started to appear. Examples are Motorola's DSP96001 and TI's TMS320C30. It should be noted that the floating-point format was not always IEEE-compatible. For instance, the TMS320C30 internal calculations were carried out in a proprietary format; a hardware chip converter was available to convert to the standard IEEE format. DSPs belonging to the development phase were characterized by fixed-width instruction sets, where one of each instruction was executed per clock cycle. These instructions could be complex, and encompassing several operations. The width of the instruction was typically quite short and did not overcome the DSP native word width.
As for DSP producers, the market was nearly equally shared between many manufacturers such as AT&T, Fujitsu, Hitachi, IBM, NEC, Toshiba, Texas Instruments and, towards the end of the1980s, Motorola, Analog Devices and Zoran.

During the market consolidation phase, enhanced DSP architectures such as Very Long Instruction Word (VLIW) and Single Instruction Multiple Data (SIMD) emerged. These architectures increase the DSP performance through parallelism. Examples of DSPs with enhanced architectures are TI's TMS320C6xxx DSPs, which was the first DSP to implement the VLIW architecture, and ADI's TigerSHARC, that includes both VLIW and SIMD features.

The number of on-chip peripherals increased greatly during this phase, as well as the hardware features that allow many processors to work together. Technologies that allow real-time data exchange between host processor and DSP started to appear towards the end of the 1990s. This constituted a real sea change in DSP system debugging and helped the developers enormously. Another phenomenon observed during this phase was the reduction of the number of DSP manufacturers. The number of DSP families was also greatly reduced, in favour of wider families that granted increased code compatibility between DSPs of different generations belonging to the same family. Additionally, many DSP families are not 'general- purpose' but are focused on specific digital signal processing applications, such as audio equipment or control loops.

DSP evolution: software tools

The improvement of DSP software tools from the early days until now has been spectacular. Code compilers have evolved greatly to be able to deal with the underlying hardware complexity and the enhanced DSP architectures. At the same time, they allow the developer to program more and more efficiently in high-level languages as opposed to assembly coding. This speeds up considerably the code development time and makes the code itself more portable across different platforms.

Advanced tools now allow the programming of DSPs graphically, i.e., by interconnecting pre-defined blocks that are then converted to DSP code. Examples of these tools are MATLAB Code Generation and embedded target products and National Instruments' LabVIEW DSP Module.

High-performance simulators, emulator and debugging facilities allow the developer to have a high visibility into the DSP with little or no interference on the program execution. Additionally, multiple DSPs can be accessed in the same JTAG chain for both code development and debugging.

DSP core architecture

DSP architecture has been shaped by the requirements of predictable and accurate real-time digital signal processing. An example is the Finite Impulse Response (FIR) filter, with the corresponding mathematical equation (1), where y is the filter output, x is the input data and a is a vector of filter coefficients. Depending on the application, there might be just a few filter coefficients or many hundreds or more.

$$y(n) = \frac{3}{4}y(n-1) - \frac{1}{8}y(n-2) + x(n)$$

As shown in Eq. (1), the main component of a filter algorithm is the 'multiply and accumulate' operation, typically referred to as MAC. Coefficient's data have to be retrieved from the memory and the whole operation must be executed in a predictable and fast way, so as to sustain a high throughput rate. Finally, high accuracy should typically be guaranteed. These requirements are common to many other algorithms performed in digital signal processing, such as Infinite Impulse Response (IIR) filters and Fourier Transforms.

Table 4 shows a selection of processing requirements together with the main DSP hardware features satisfying them.

Table 4: Main requirements and corresponding DSP hardware implementations for predictable and accurate real-time digital signal processing. The numbers in the first column refer to the section treating the topic.

Processing requirements	Hardware implementations satisfying the requirement
3.2 Fast data access	• High-bandwidth memory architectures • Specialized addressing modes • Direct Memory Access (DMA)
3.3 Fast computation	• MAC-centered • Pipelining • Parallel architectures (VLIW, SIMD)
3.4 Numerical fidelity	• Wide accumulator registers, guard bits, etc.
3.5 Fast execution control	• Hardware-assisted, zero-overhead loops, shadow registers, etc.

High-bandwidth memory architectures

DSPs are typically based upon the Harvard architecture, shown in Fig. 4(b), or upon modified versions of it, such as the Super-Harvard architecture shown in Fig. 4(c). In the Harvard architecture there are separate memories for data and program instructions, and two separate buses connect them to the DSP core. This allows fetching program instructions and data at the same time, thus providing better performance at the price of an increased hardware complexity and cost.

The Harvard architecture can be improved by adding to the DSP core a small bank of fast memory, called 'instruction cache', and allowing data to be stored in the program memory. The last-executed program instructions are relocated at run time in the instruction cache.

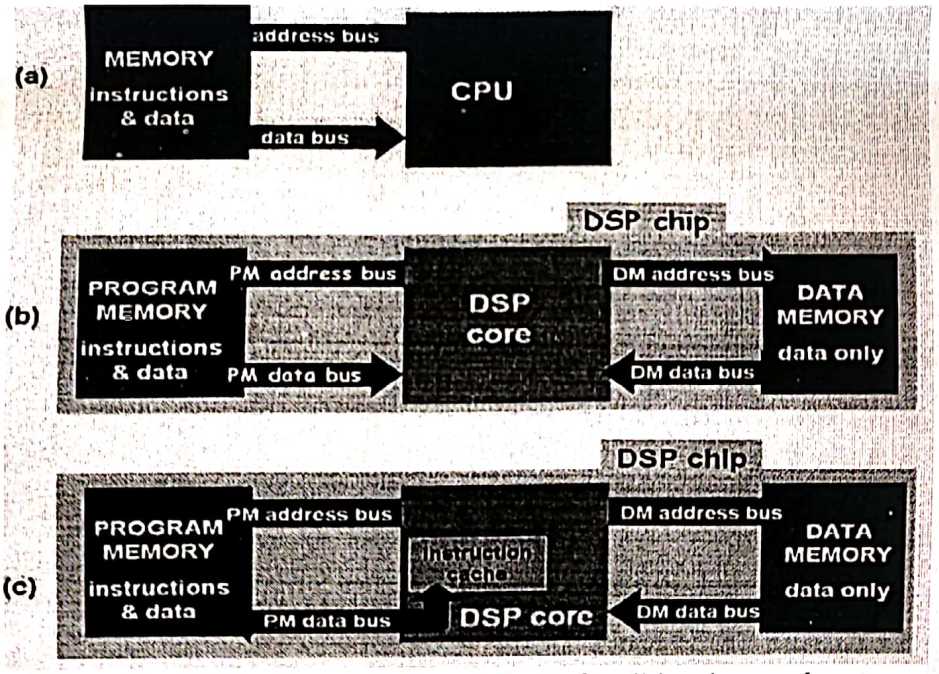


Fig. 4: (a) Von Neumann architecture, typical of traditional general-purpose microprocessors. (b) Harvard and (c) Super-Harvard architectures, typical of DSPs.

The fact of having the cache memory very close to the DSP allows clocking it at high speed, as routing wire delays are short. Figure 6 shows the cache architecture for TI TMS320C67xx DSP, including both program and data cache. There are two levels of cache, called Level 1 (L1) and Level 2 (L2). The L1 cache comprises 8 kbyte of memory divided into 4 kbyte of program cache and 4 kbyte of data cache. The L2 cache comprises 256 kbyte of memory divided into 192 kbyte mapped-SRAM memory and 64 kbyte dual cache memory. The latter can be configured as mapped memory, cache or a combination of the two.

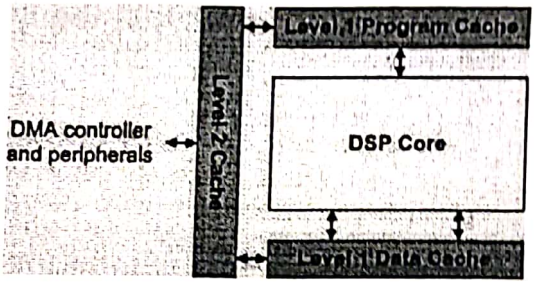


Fig. 5: TI DSP TMS320C67xx family two-level cache architecture

As shown above, cache memories improve the average system performance. However, there are drawbacks to the presence of a cache in DSP-based systems, owing to the lack of full predictability for cache hits. A missing cache hit happens when the data or the instructions needed by the DSP are not stored in cache memory, hence they have to be fetched from a slower memory with an execution speed penalty. A situation causing a missing cache hit is, for instance, the flow changes due to branch instructions. The consequence is a difficult worst-case-scenario prediction, which is particularly negative for DSP-based systems where it is important to be able to calculate and predict the system time response. There may, however, be methods used to limit these effects, such as the possibility for the user to lock the cache so as to execute time-critical sections in a deterministic way.

Specialized addressing modes

DSPs include specialized addressing modes and corresponding hardware support to allow a rapid access to instruction operands through rapid generation of their location in memory. DSPs typically support a wide range of specialized addressing modes, tailored for an efficient implementation of digital signal processing algorithms.

Figure 7 adds the address generator units to the basic DSP architecture shown in Fig. 4(c). As in general-purpose processors, DSPs include a Program Sequencer block, which manages program structure and program flow by supplying addresses to memory for instruction fetches. Unlike general- purpose processors, DSPs include address generator blocks, which control the address generation for specialized addressing modes such as indexing addressing, circular buffers, and bit-reversal addressing. The two last addressing modes are discussed below.

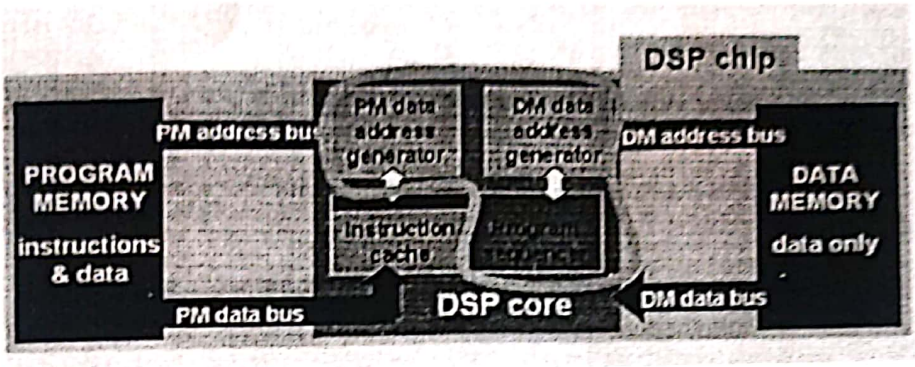


Fig. 7: Program sequencer and address generator units location within a generic DSP core architecture

Circular buffers are limited memory regions where data are stored in a First-In First-Out (FIFO) way; these memory regions are managed in a 'wrap-around' way, i.e., the last memory location is followed by the first memory location. Two sets of pointers are used, one for reading and one for writing; the length of the step at which successive memory locations are accessed is called 'stride'. Address generator units allow striding through the circular buffers without requiring dedicated instructions to determine where to access the following memory location, error detection and so on. Circular buffers allow storing bursts or continuous streams of data and processing them in the order in which they have arrived. Circular buffers are used for instance in the implementation of digital filters; strides higher than one are useful in case of multi-rate signal processing. Figure 8 shows the order in which data are accessed for a read operation in case of an eleven-element circular buffer and with a stride equal to four.

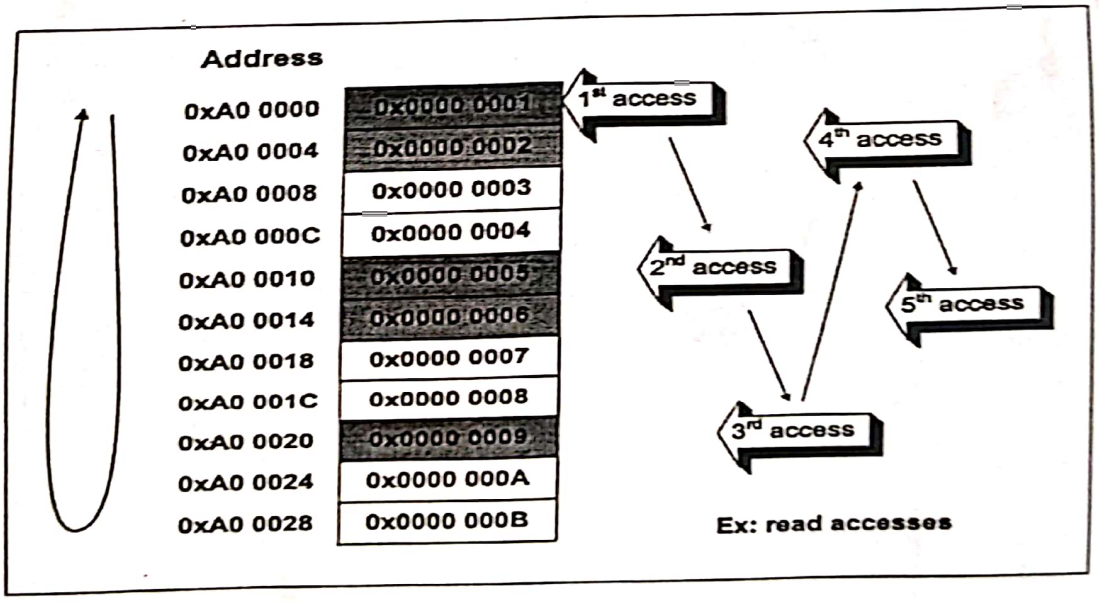


Fig. 8: Example of read data access order in a circular buffer composed of 11 elements and with stride equal to 4 elements

Bit-reversal addressing, shown in Fig. 9, is an essential step in the discrete Fourier transforms calculation. In fact, many implementations of the Fourier transforms require a re-ordering of either

the input or the output data that corresponds to reversing the order of the bits in the array index. Figure 9 gives an example of the bit-reversal mechanism. Carrying it out by software is very demanding and would result in using many CPU cycles, which are saved thanks to the hardware bit-reversal functionality.

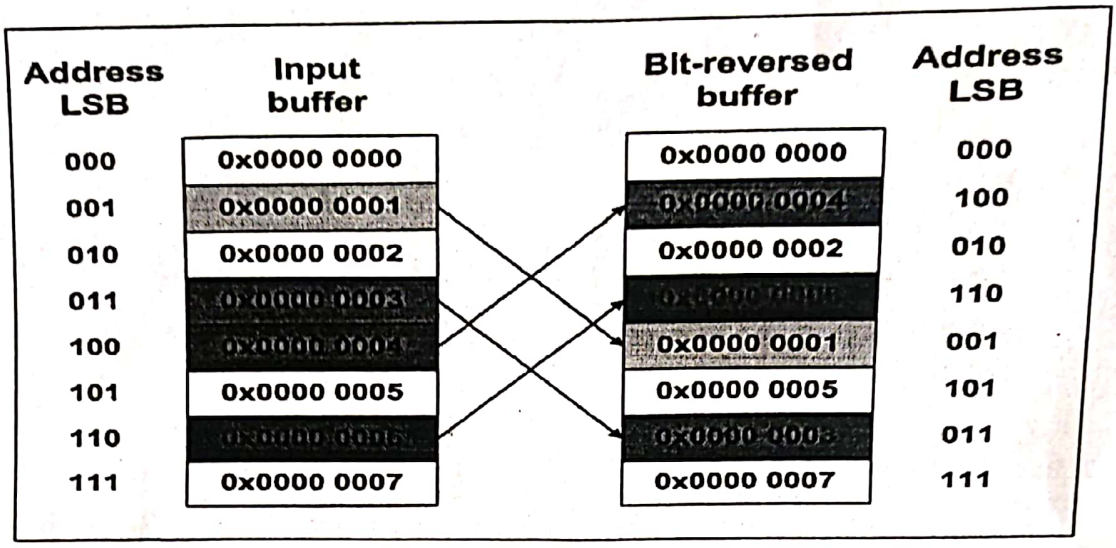


Fig. 9: Bit-reversal mechanism

Direct Memory Access (DMA) controller

The DMA controller is a second processor working in parallel with the DSP core and dedicated to transferring information between two memory areas or between peripherals and memory. In doing so the DMA controller frees the DSP core for other processing tasks. Figure 10 shows an example of the DMA location within a general DSP core architecture.

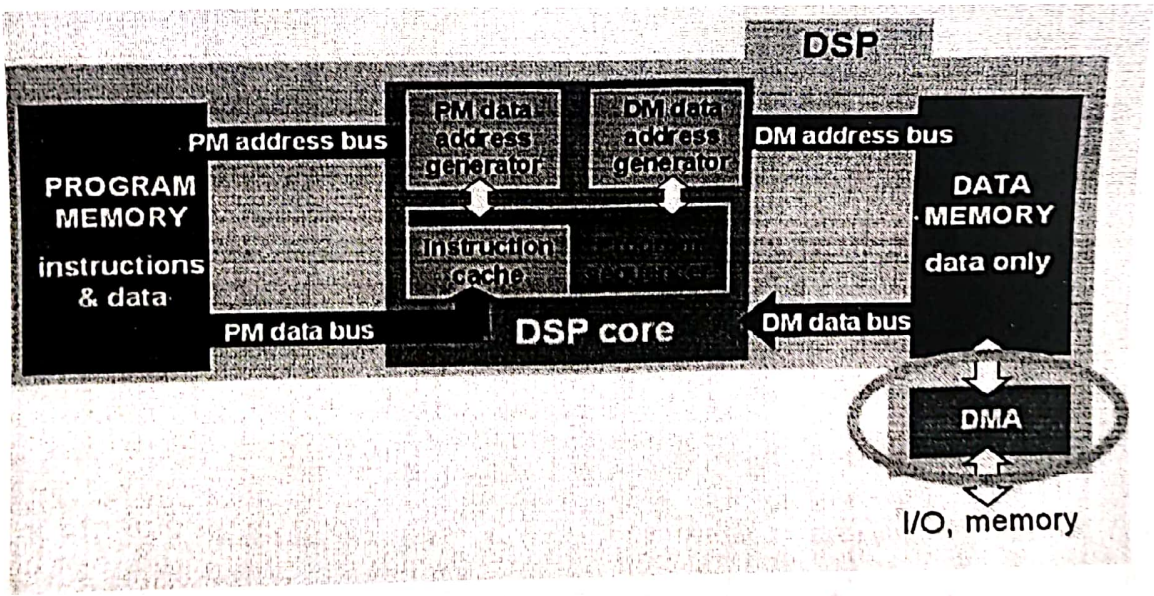


Fig. 10: An example of DMA controller location within a generic DSP core architecture

A DMA coprocessor can transfer data as well as program instructions, the latter transfer corresponding typically to the case of code overlay, i.e., of code stored in an external memory and moved to an internal memory (for instance L1) when needed. Multiple and independent DMA channels are also available for greater flexibility. Bus arbitration between the DMA and the DSP core is needed to avoid colliding memory accesses when the DMA and the DSP core share the same bus to access peripherals and/or memories. To prevent bottlenecks, recent DSPs typically fit DMA controllers with dedicated buses.

Figure 11 shows the advantages of DMA for the DSP core efficient use: the DSP core must set up the DMA but still there is a net gain in the DSP core availability for other processing activities. Nowadays there are two classes of DMA transfer configurations: register-based and RAM-based, the latter one also called descriptor-based. In register-based DMA controllers the transfer set-up is done by the DSP core via the registers set-up. This method is very efficient but allows mainly simple DMA operations. In RAM-based DMA controllers the set-up parameters are stored in memory. This method is preferred by powerful and recent DSPs as it allows great DMA transfer flexibility

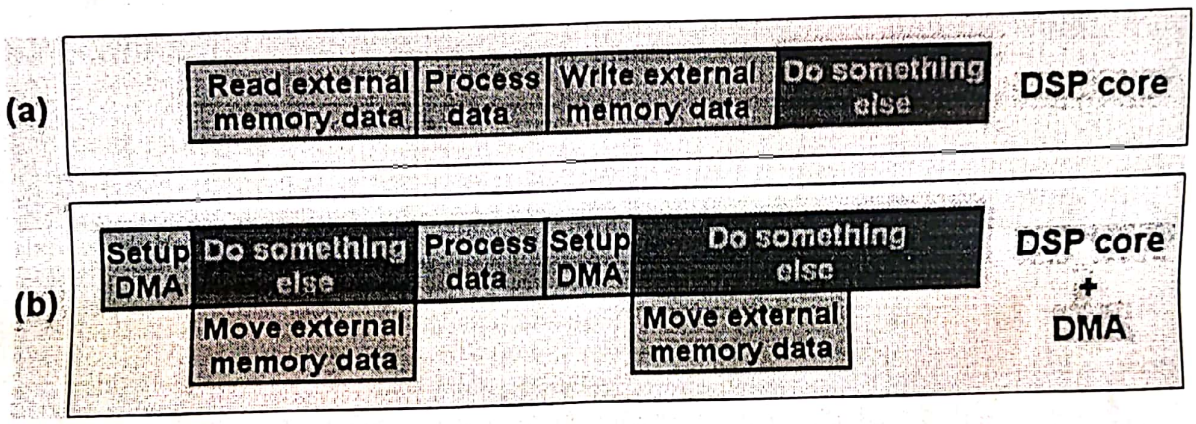


Fig. 11: (a) Read–process–write data when the DSP core only is present; (b) same activity when the DMA takes care of data transfers

Figure 12 provides two examples of transfer configurations. Plot (a) shows a chained DMA transfer, where the completion of a data transfer triggers a new transfer.

This type of data transfer is particularly suited to applications that require a continuous data stream in input. Plot (b) shows a multi-dimensional data transfer, obtained by changing the stride of the DMA transfer. This type of data transfer is particularly useful for video applications.

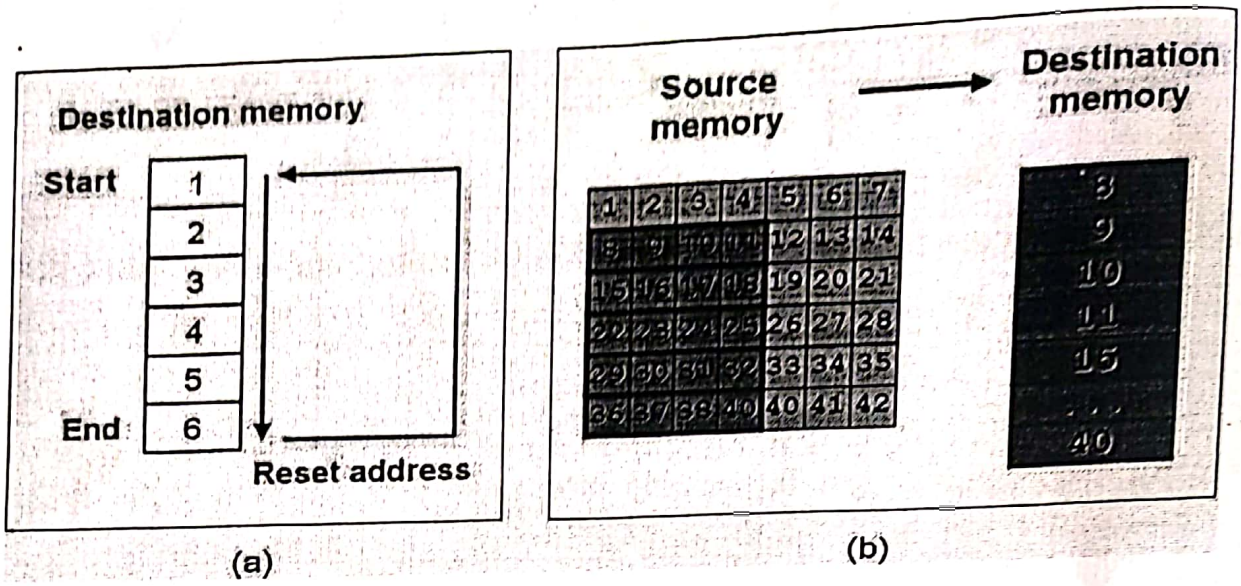


Fig. 12: Examples of DMA transfer configurations. (a): chained DMA transfer; (b): Multi-dimensional data transfer.

DSP external events and interrupts can be used to trigger a DMA data transfer. DMA controllers can also generate interrupts to communicate with the DSP core, for instance to inform it that a data transfer has been completed. An example of a powerful and highly flexible DMA controller is that implemented for TI's TMS320C6000 family.

Fast computation

Here we discuss techniques and architectures used in DSPs for a fast computation. The MAC-centered architecture has been common to all DSPs since their early days. The techniques and architectures were introduced from the 1990s onwards.

MAC-centered

The MAC operation is used by many digital processing algorithms, as discussed at the beginning of Section; consequently, its execution must be optimized so as to improve the DSP overall performance. The basic DSP arithmetic processing blocks are a) many registers; b) one or more multipliers; c) one or more Arithmetic Logic Units (ALUs); d) one or more shifters. These blocks work in parallel during the same clock cycle thus optimizing MAC as well as other arithmetic operations. The blocks are shown in Fig. 13 and are briefly described below.

a) Registers: these are banks of very fast memory used to store intermediate data processing. Very often they are wider than the DSP normal word width, so as to provide a higher resolution during the processing.
b) Multiplier: it can carry out single-cycle multiplications and very often it includes very wide accumulator registers to reduce round-off or truncation errors. As a consequence, truncation and round-off errors will happen only at the end of the data processing, when the data is stored onto memory. Sometimes an adder is integrated in the multiplier unit.
c) ALU: it carries out arithmetic and logical operations.
d) Shifters: it shifts the input value by one or more bits, left or right. In the latter case, the shifter is called a barrel shifter and is especially useful in the implementation of floating point add and subtract operations.

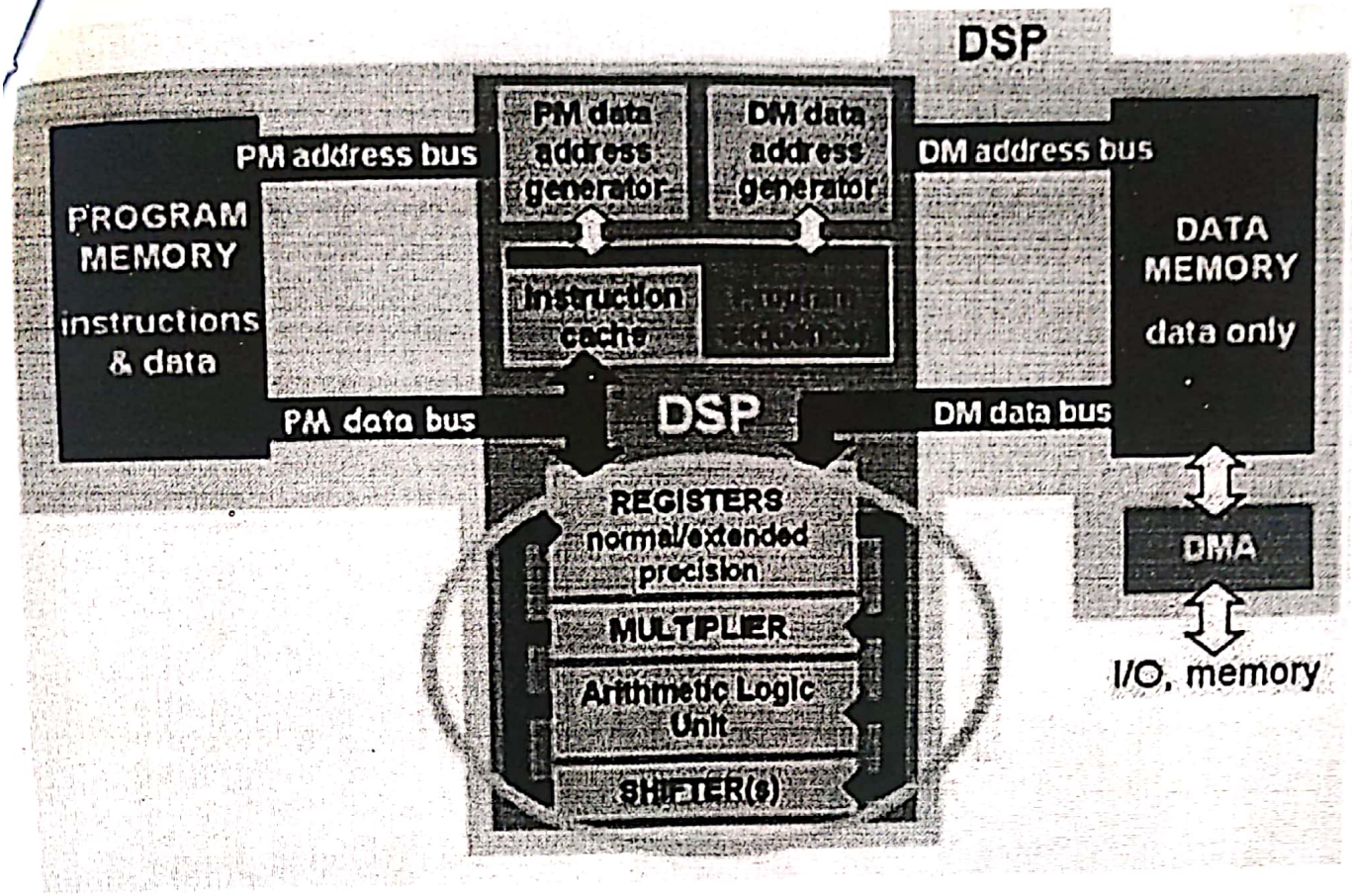


Fig. 13: Basic DSP arithmetic processing blocks. The structure shown is that of ADI SHARC.

Instruction pipelining

Instruction pipelining has become an important element to achieve high DSP performance. It consists of dividing the execution of instructions into different stages and executing the different instructions in parallel stages. The net result is an increased throughput of the instruction execution. The whole process can be compared to a factory assembly line, which produces cars for instance: more than one car is in the assembly line at the same moment, at different stages of assembly. This provides a production higher than the case where only one car at a time is produced, where many specialized crews are idle waiting for the next car to require their work. Table 5 shows the basic pipelining stage into which each instruction is divided:

1. Fetch. The DSP calculates the address of the next instruction to execute and retrieve the op- code, i.e., the binary word containing the operands and the operation to be carried out on them.
2. Decode. The op-code is interpreted and sent to the corresponding functional unit. The instruction is interpreted and the operands are retrieved.
3. Execute. The instruction is executed and the results are written onto the registers.

Table 5: The three basic pipelining stages and corresponding actions

Basic pipelining stages	Action
Fetch	• Generate program fetch address • Read op-code
Decode	• Route op-code to functional unit • Decode instruction • Read operands
Execute	• Execute instruction • Write results back to registers

Figure 14 shows the advantage of a pipelined CPU with respect to a non-pipelined CPU, in terms of processing time gain. In a non-pipelined CPU the different instructions are executed serially, while in a pipelined CPU only the same type of stages (e.g. Fetch, Decode and Execute) are serialized and different instructions are executed in parallel.

A pipeline is called fully-loaded if all stages are executed at the same time; this corresponds to the maximum possible instruction throughput. The depth of the pipeline, i.e., the number of stages into which an instruction is divided, can vary from one processor to another.

Generally speaking, a deeper pipeline allows the processor to execute faster, hence many processors sub-divide pipeline stages into smaller steps, each one executed at each clock cycle. The smaller the step, the faster the processor clock speed can be. An example of deep pipeline is the TI TMS320C6713 DSP, which includes four fetch stages, two decode stages, and up to ten execution stages.

There are drawbacks and limitations to the pipelining technique. One drawback is the hardware and programming complexity required by it, for instance in terms of capabilities needed in the compiler and the scheduler. This is especially true in the case of deep pipelines. A limitation in the effective instruction execution throughput is given by situations that prevent the pipeline from being fully-loaded. These situations include pipeline flushes due to changes in the program flow, such as code branches or interrupts. In this case, the DSP does not know which instructions it should execute next until the branch instruction is executed. Other situations are data hazards, namely when one instruction needs the result of a previous instruction to be executed. Apart from a reduced throughput,

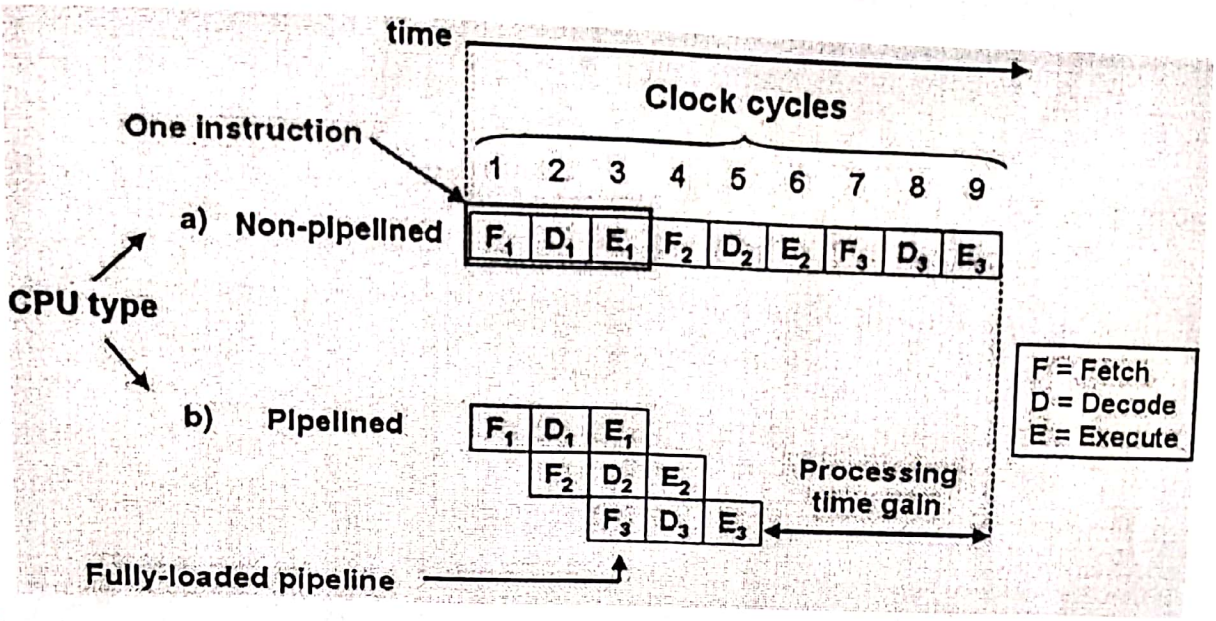


Fig. 14: Instruction execution and processing time gain of a pipelined CPU (plot b) with respect to a non-pipelined one (plot a)

these pipeline limitations cause a more difficult prediction of the worst-case scenario. Techniques not described here are available to provide the DSP programmer with a pipeline control; they include time-stationary pipeline control, data-stationary control, and interlocked pipeline.

Parallel architectures

The DSP performance can be increased by an increased parallelism in the instruction's execution. Parallel-enhanced DSP architectures started to appear on the market in the mid-1990s and were based on instruction-level parallelism, data-level parallelism, or a combination of both. These two approaches are called Very Long Instruction Word (VLIW) and Single-Input Multiple-Data (SIMD), respectively and are discussed below.

VLIW architectures are based upon instruction level parallelism, i.e., many instructions are issued at the same time and are executed in parallel by multiple execution units. As a consequence, DSPs based on this architecture are also called 'multi-issue' DSP. This is an innovative architecture that was first used in the TI TMS320C62xx DSP family. Figure 15 shows an example of the VLIW architecture: eight, 32-bit instructions are packed together in a 256-bit wide instruction which is fed to eight separate execution units.

Characteristics of VLIW architectures include simple and regular instruction sets. Instruction scheduling is done at compile-time and not at run-time so as to guarantee a deterministic behaviour. This means that the decision on which instructions have to be executed in parallel is done when the program is compiled, hence the order does not change during the program execution. A run-time scheduling would instead make the scheduling dependent on data and resources availability, which could change for different program executions. An important advantage of the VLIW architecture is that it can increase the DSP performance for a wide range of algorithms. Additionally, the architecture is potentially scalable, i.e., more execution units could be added to allow a higher number of instructions to be executed in parallel. There are disadvantages as well, such as the high memory use and power consumption required by this architecture. From a programmer's viewpoint, writing assembly code for VLIW architecture is very complex and the optimization is often better left to the compiler.

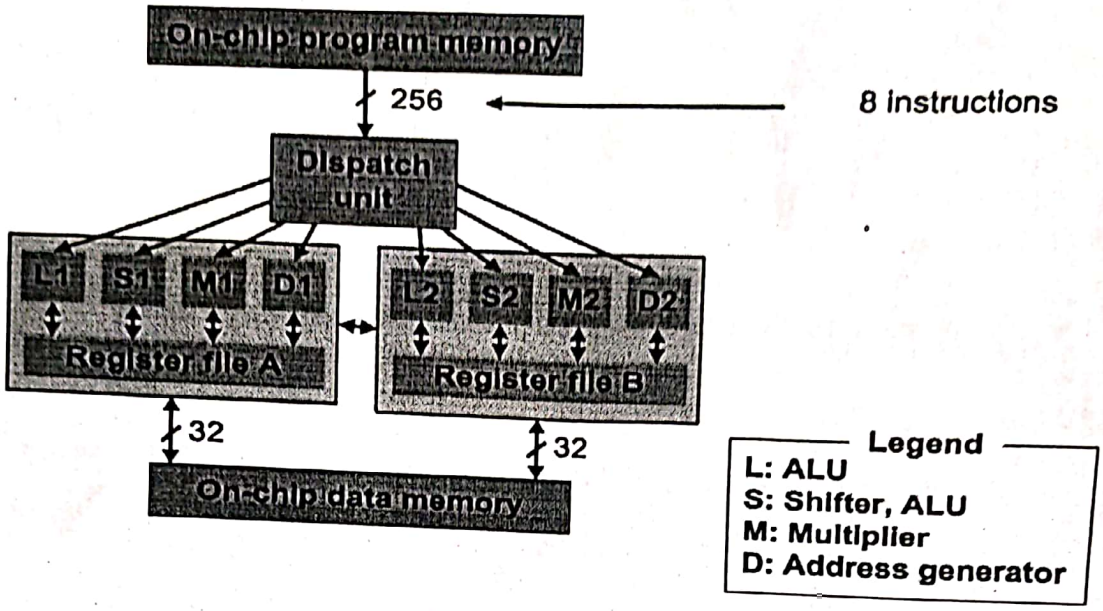


Fig. 15: TI TMS320C6xxx family VLIW architecture

SIMD architectures are based on data-level parallelism, i.e., only one instruction is issued at a time but the same operation specified by the instruction is performed on multiple data sets. Figure 16 shows the example of a DSP based upon the SIMD architecture: two 32-bit input registers provide four, 16-bit each, data inputs. They are processed in parallel by two separate execution units that carry out the same operation. The two, 16-bit data outputs are packed into a 32-bit register. Typical SIMD architecture can support multiple data width and is most effective on algorithms that require the processing of large data chunks. The SIMD operation mode can be switched ON or OFF, for instance in the ADI SHARC DSP. An advantage of the SIMD architecture is that it is applicable to other architectures; an example is the ADI TigerSHARC DSP that comprises both VLIW and SIMD characteristics. SIMD drawbacks include the fact that SIMD architectures are not useful for algorithms that process data serially or that contain tight feedback loops. It is sometimes possible to convert serial algorithms to parallel ones; however, the cost is in reorganization penalties and in a higher program- memory usage, owing to the need to re-arrange the instructions.

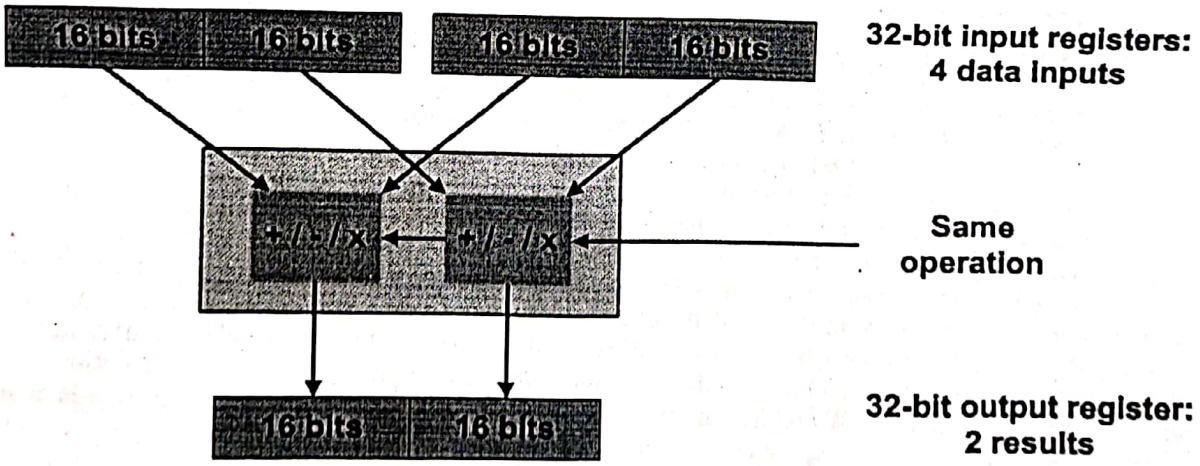


Fig. 16: Simplified schematics for ADI SHARC DSP as an example of SIMD architecture

Numerical fidelity

Arithmetic operations such as additions and multiplications are the heart of DSP systems. It is thus essential that the numerical fidelity be maximized, i.e., that errors due to the finite number of bits used in the number representation and in the arithmetic operations be minimized. DSPs have many ways to obtain this, ranging from the numeric representation to dedicated hardware features.
As far as the number representation is concerned, DSPs can be divided into two categories: fixed point and floating point.
Fixed-point DSPs perform integer as well as fractional arithmetic, and can support data widths of 16, 24 or 32 bits. A fixed-point format can represent both signed and unsigned integers and fractions. Fractional numbers can take values in the [□1.0, 1.0] range and are often indicated as Q x, y, where 'x' indicates the number of bits located before the binary point and 'y' the number of bits after it. Figure 17(a) shows how 16-bit signed fractional point numbers are coded. Signed fractional numbers with 24-bit and 32-bit data width are coded in an equivalent way as Q1.23 and Q1.31, respectively. They can take values in the same [□1.0, 1.0] range, however, their resolution is higher than the 16-bit implementation.

Floating-point DSPs represent numbers with a mantissa and an exponent, nowadays following the IEEE 754 standard shown in Fig. 17(b) for a 32-bit number. The mantissa dictates the number precision and the exponent controls its dynamic range. Numbers are scaled so as to use the full word- length available, hence maximizing the attainable precision.

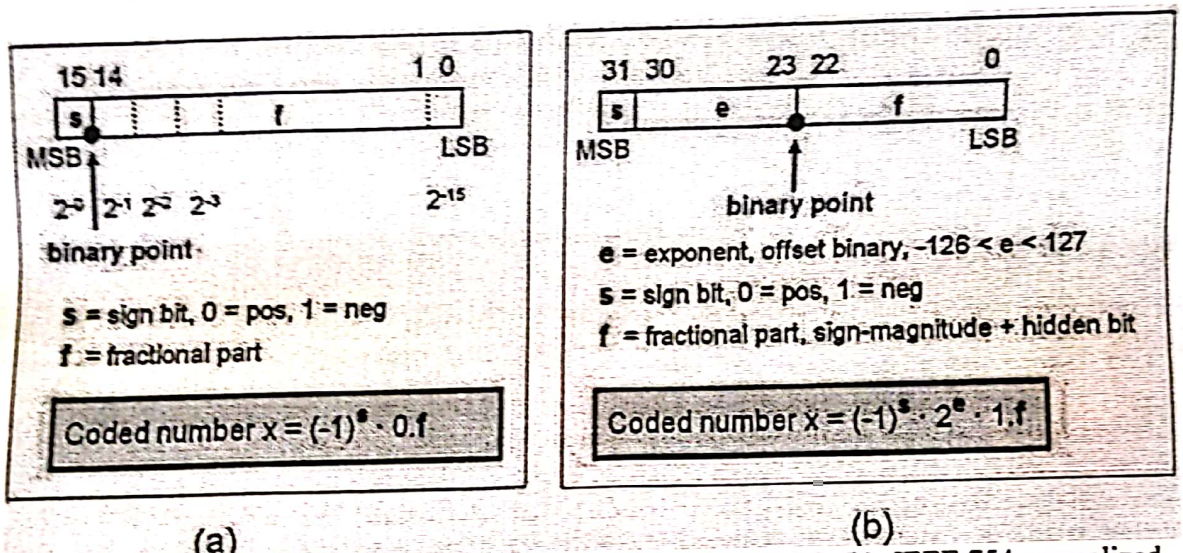


Fig. 17: (a): 16-bit signed fractional point, often indicated as Q1.15. (b): IEEE 754 normalized representation of a single precision floating point number.

Floating-point numbers provide a higher dynamic range, which can be essential when dealing with large data sets and with data sets whose range cannot be easily predicted. The dynamic range for a 32-bit number represented as fixed-point and as floating-point is shown in Fig. 18.

$$\text{Dynamic range}_{dB} = 20 \log_{10} \left[\frac{\text{largest value}}{\text{smallest value}} \right]$$

Fixed point ~ 180 dB

Floating point ~1500 dB

Fig. 18: Dynamic range for 32-bit data, represented as 32-bit signed fractional point and IEEE 754 normalized number

In addition to the different number formats available, DSPs provide hardware ways to improve numerical fidelity. One example is represented by the large accumulator registers, used to hold intermediate and final results of arithmetic operations. These registers are several bits (at least four) wider than the normal registers in order to prevent overflow as much as possible during accumulation operations. The extra bits are called guard bits and allow one to retain a higher precision in intermediate computation steps. Flags to indicate that an overflow/underflow has happened are also available. These flags are often connected to interrupts, thus allowing exception-handling routines to be called. Another means DSPs have to improve numerical fidelity is saturated arithmetic. This means that a number is saturated to the maximum value that can be represented, so as to avoid wrap-around phenomena.

Fast-execution control

Here we show two important examples of how DSP can fast-execute control instructions. The first example is the zero-overhead hardware loop and refers to the program flow control in loops. The second example refers to how DSPs react to interrupts.

Looping is a critical feature in many digital signal processing algorithms. An important DSP feature is the implementation by hardware of looping constructs, referred to as 'zero-overhead hardware loop'. This allows DSP programmers to initialize loops by setting a counter and defining the loop bounds, without spending any software overhead to update and test loop counters or branching back to the beginning of the loop.

The capability to service interrupts very quickly and in a deterministic way is an important DSP characteristic. Interrupts are internal (for instance generated by internal timers) or external (brought to the DSP code via pins) events that change the DSP execution flow when they are serviced. The latency is the time elapsed from when the interrupt event is triggered and when the DSP starts to execute the first instruction of the corresponding Interrupt Service Routine (ISR). When an interrupt is received and if the interrupt has a sufficiently-high priority, the DSP must carry out the following actions:

a) stop its current activity;
b) save the information related to the interrupted activity (called context) into the DSP stack;
c) start servicing the interrupt.

The context corresponding to the interrupted activity can be restored when the ISR has been executed and the previous activity is continued.

Table 6: Interrupt dispatchers available on the ADI ADSP21160M DSP. The instruction cycle is 12.5 μs, hence the number of cycles can easily be converted to time.

Interrupt dispatcher	Cycles before ISR	Cycles after ISR
Normal	183	109
Fast	40	26
Super-fast (with alternate registers set)	34	10
Final	24	15

DSP core example: TI TMS320C67x

Figure 19 shows TI's TMS320C6713 DSP core architecture, as an example of modern VLIW architecture implementing many of the characteristics described in Section above.
Boxes inside the yellow square belong to the DSP core architecture, which here is considered to include the cache memory as well as the DMA controller. The white boxes are components common to all C6000 devices; grey boxes are additional features on the TMS320C6713 DSP.

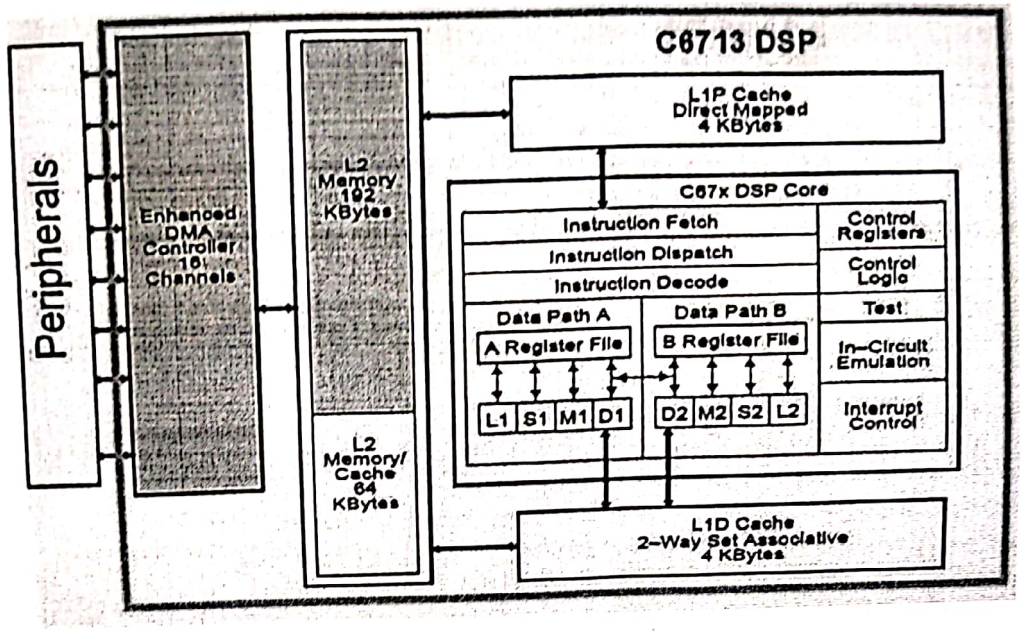


Fig. 19: TI TMS320C6713 DSP core architecture. Picture courtesy of TI [23].

The TMS320C6713 DSP is a floating-point DSP with VLIW architecture. The internal program memory is structured so that a total of eight instructions can be fetched at every cycle. To give a numerical example, with a clock rate of 225 MHz the C6713 DSP can fetch eight, 32-bit instructions every 4.4 ns. Features of the C6713 include 264 kBytes of internal memory: 8 kB as L1 cache and 256 kB as L2 memory shared between program and data space. The processing of instructions occurs in each of the two data paths (A and B), each of which contains four functional units (.L, .S, .M, .D). An Enhanced DMA (EDMA) controller supports up to 16 EDMA channels. Four of the sixteen channels (channels 8–11) are reserved for EDMA chaining, leaving twelve EDMA channels available to service peripheral devices.

DSP peripherals

The available peripherals are an important factor for the DSP choice. Peripherals are here considered as belonging to two categories:

a) interconnect;
b) services, such as timers, PLL and power management.

DSP developers must in fact carefully evaluate the needs of their system in terms of interconnect and services required, to avoid bottlenecks and reduced system performance.
Modern DSPs often have several peripherals integrated on-chip, such as UARTs, serial, USB and video ports. There are benefits in using embedded peripherals, such as fast performance and reduced power consumption. There are, however, drawbacks, in that embedded peripherals can be less flexible across applications and their unit cost might be higher.
The evolution of DSP-supported peripherals has been terrific over the last 20 years. From the original few parallel and serial ports, DSP can now support a wide peripherals range, including those needed by audio/video streaming applications. Often the DSP chip does not have pins to allow using all supported peripherals at the same time. To overcome this limitation, the pins are

multiplexed, i.e., the DSP developer must select at boot time which peripherals he/she needs to have available. Hints on different interfacing possibilities to external memories and data converter memories are provided in Sections below.

Interconnect

The amount of supported interconnect and data I/O is huge, so only a few examples are given below, divided per interconnect type.

Serial interfaces

d) Serial Peripheral Interface (SPI): this is an industry-standard synchronous serial link that supports communication with multiple SPI compatible devices. The SPI peripheral is a synchronous, four-wire interface consisting of two data pins, one device select pin, and a gated clock pin. With the two data pins, it allows for full-duplex operation to other SPI compatible devices. An example of DSP fitted with a SPI port is ADI's Blackfin ADSP- BF533.

e) Multichannel Buffered Serial Ports (McBSP) on TI's DSPs:this serial interface is based upon the standard serial port found in TMS320C2x and TMS320C5x DSPs.

f) Multichannel Audio Serial Port (McASP) on TI's DSPs: this is a serial port optimized for the needs of multichannel audio applications. Each McASP includes transmit and receive sections that can operate synchronized as well as completely independent, i.e., with separate master clocks, bit clocks, and data stream formats.

Services

System services provide functionality that is common to embedded systems; the on-chip hardware is generally accompanied by an API that allows one to easily interface to them. A few examples of services are given below.

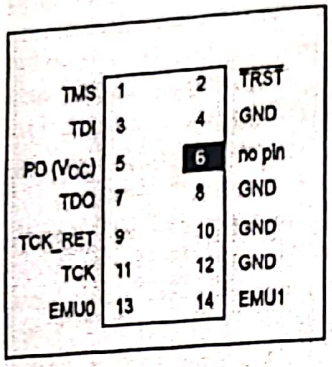
g) Timers: DSPs are typically fitted with one or more general-purpose timers that are used to time or count events, generate interrupts to the CPU, or send synchronization events to a DMA/EDMA controller. More information on timers for TI's TMS320C6000 DSPs.

h) PLL controller: it generates clock pulses for the DSP code and the peripherals from internal or external clock signals. More information on PLL controllers for TI's TMS320C6000 DSPs.

i) Power Management: the power-down logic allows the reduction of clocking so as to reduce power consumption. In fact, most of the operating power of CMOS logic dissipates during circuit switching from one logic state to the other. Significant power can be saved by preventing some of these level switches. More information on power management logic of TI's TMS320C6000 DSPs.

j) Boot configuration: a variety of boot configurations are often available in DSPs. They are user-programmable and determine what actions the DSP performs after it has been reset to prepare for the initialization. These actions include loading the DSP code load from external memory or from an external host. Some boot modes are outlined in Section 4.7. More information on boot modes, device configuration, and available boot processes for TI TMS320C62x/67x.

k) JTAG: this interface implements the IEEE standard 1149.1 and allows emulation and debugging. A detailed description of its use can be found in Section 7.2. Figure 20 shows a typical JTAG connector and corresponding signals.



Signal	Description	Emulator state	Target state
TMS	Test mode select	OUT	IN
TDI	Test data input	OUT	IN
TDO	Test data output	IN	OUT
TCK	Test clock: 10.368 MHz clock source from emulation cable pod, that can be used to drive the system test clock.	OUT	IN
$\overline{TRST}$	Test reset	OUT	IN
EMU0	Emulation pin 0	IN	IN/OUT
EMU1	Emulation pin 1	IN	IN/OUT
PD(Vcc)	Presence detect: it indicates that the emulation cable is connected and that the power is powered up	IN	OUT
TCK_RET	Test clock return, input to the emulator	IN	OUT
GND	Ground		

Fig. 20: Fourteen-pin JTAG header and corresponding signals. Picture courtesy of TI.

TI C6713 DSP example

The peripherals available on TI's TMS320C6713 DSP are shown in Fig. 21 as boxes encircled by a yellow shape. The white boxes are components common to all C6000 devices, while grey boxes are additional features on the TMS320C6713 DSP.
Many peripherals are available on this DSP; however, there are pins that are shared by more than one peripheral and are internally multiplexed. Most of these pins are configured by software via a configuration register, hence they can be programmed to switch functionality at any time. Others (such as the HPI pins) are configured by external pullup/pulldown resistors at DSP chip reset; as a consequence, only one peripheral has primary control of the function of these pins after reset.

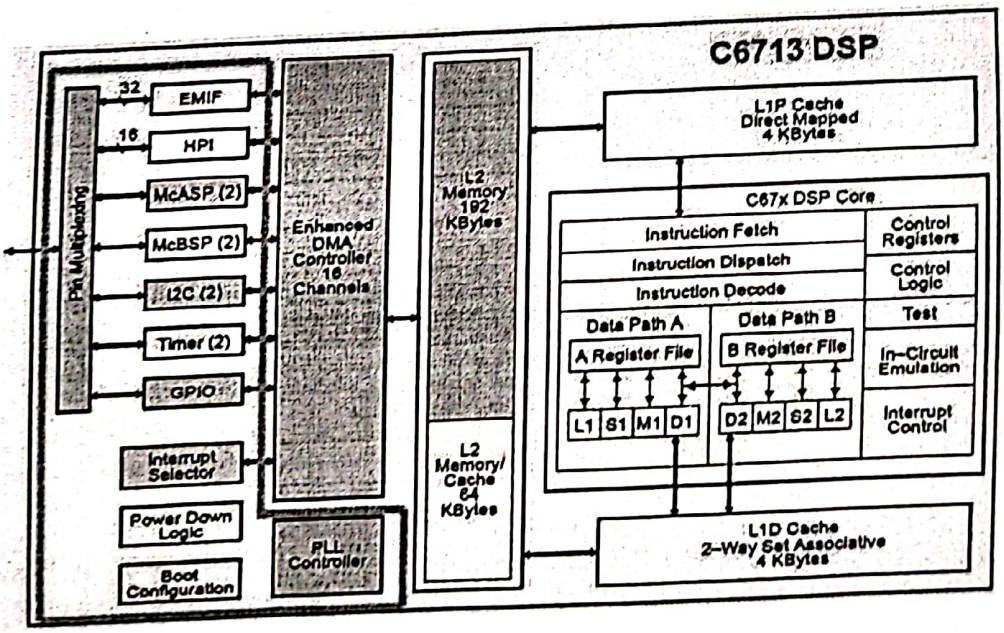


Fig. 21: TI TMS320C6713 DSP available peripherals. Picture courtesy of TI.

Memory interfacing

DSPs often have to interface with external memory, typically shared with host processors or with other DSPs. The two main mechanisms available to implement the memory interfacing are to use hardware interfaces already existing on the DSP chip or to provide external hardware that carries out the memory interfacing. These two methods are briefly mentioned below. Hardware interfaces are often available on TI as well as on ADI DSPs. An example is TI External Memory Interface (EMIF), which is a glueless interface to memories such as SRAM, EPROM, Flash, Synchronous Burst SRAM (SBSRAM) and Synchronous DRAM (SDRAM). On the TMS320C6713 DSP, for instance, the EMIF provides 512 Mbytes of addressable external memory space. Additionally, the EMIF supports memory width of 8 bits, 12 bits and 32 bits, including read/write of both big- and little-endian devices.

When no dedicated on-chip hardware is available, the most common solution for interfacing a DSP to an external memory is to add external hardware between memory and DSP, as shown in Fig. 22. Typically, this is done by using a CPLD or an FPGA which implements address decoding and access arbitration. Care must be taken when programming the access priority and/or interleaved memory access in the CPLD/FPGA. This is essential to preserve the data integrity. Synchronous mechanisms should be preferred over asynchronous ones to carry out the data interfacing.

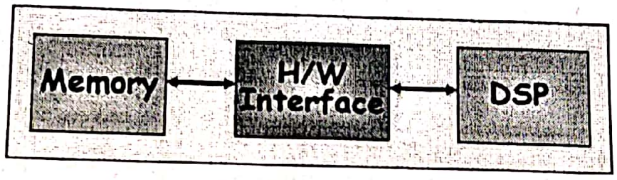


Fig. 22: Generic DSP–external memory interfacing scheme. Very often the h/w interface consists of a CPLD or an FPGA.

1. Data converter interfacing

DSPs provide a variety of methods to interface with data converters such as ADCs. On-chip peripherals are a very convenient data transfer mechanism, since data converters are typically much slower than the DSPs they are interfaced with, hence asking the DSP core to directly retrieve data from the converters is a waste of valuable processing time.

Serial interfaces are often available in TI's DSPs: peripherals such as McBSP and McASP plus the powerful DMA allow an easy interface to many data converter types. Another possible solution for TI DSPs is to use the EMIF in asynchronous mode together with the DMA.

A general solution for implementing the DSP–data converter interface is to use an FPGA between DSP and converter, so as to re-buffer the data. Additional pre- processing, such as filtering or down-conversion, can also be carried out in the FPGA. This is the case for instance in CERN's LEIR LLRF system, where converters such as ADCs and DACs are hosted on daughtercards. Powerful FPGAs located on the same daughtercards carry out pre-processing and diagnostics actions under full DSP control. Finally, mixed-signal DSPs, i.e., DSPs with embedded ADCs and/or DACs, are also available. An example of mixed-signal DSP is ADI's ADSP-21990, containing a pipeline flash converter with eight independent analog inputs and sampling frequency of up to 20 MHz.

DSP booting

The actions executed by the DSP immediately after a power-down or a reset are called DSP boot and are defined by a certain number of configurable input pins. This paragraph will focus on how the executable file(s) is uploaded to the DSP after a power-down or reset. Two methods are available, which typically correspond to differently built executables.

The first method is to use the JTAG connector to directly upload to the executable in the DSP.

Upon a DSP power-down the code will typically not be retained in the DSP and another code upload will be necessary. This method is used during the system debugging phase, when additional useful information can be gathered via the JTAG.
On operational systems the DSP loads the executable code without a JTAG cable. Many methods are available for doing this, depending on the DSP family and manufacturer; some general ways are described below.

l) No-boot. The DSP fetches instructions directly from a pre-determined memory address, corresponding to EPROM or Flash memory and executes them. On SHARC DSPs, for instance, the pre-defined start address is typically 0x80 0004.

m) Host-boot. The DSP is stalled until the host configures the DSP memory. For TI TMS320C6xxx DSPs, for instance, this is done via the HPI interface. When all necessary memory is initialized, the host processor takes the DSP out of the reset state by writing in a HPI register.

n) ROM-boot. A boot kernel is uploaded from ROM to DSP at boot time and starts executing itself. The kernel copies data from an external ROM to the DSP by using the DMA controller and overwrites itself with the last DMA transfer. After the transfer is completed, the DSP begins its program execution. Figure 23 visualizes the TI DSP process of booting from ROM memory: the program (shown in green) has been moved from ROM to L2 and L1 Program (L1P) cache via EMIF and DMA.

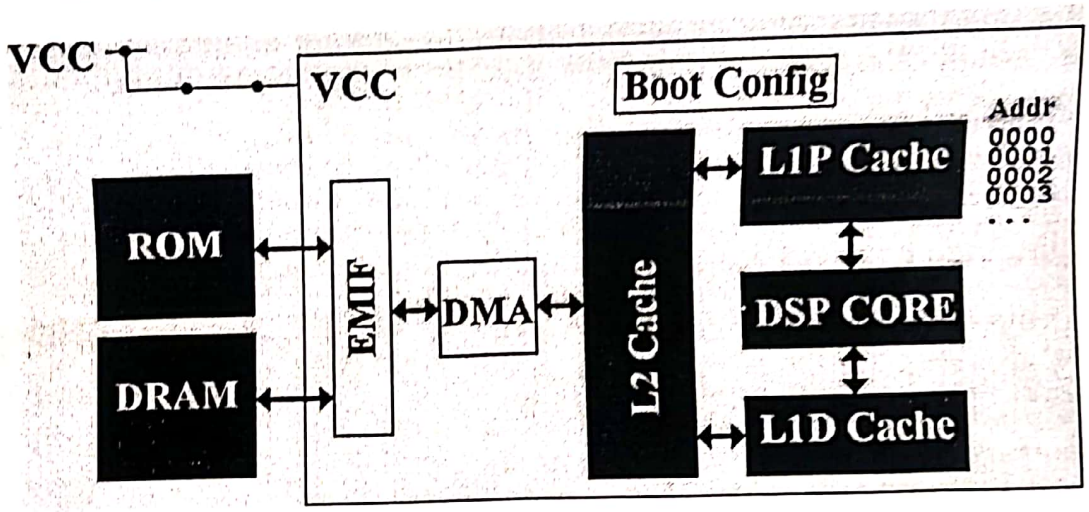


Fig. 23: Example of TI TMS320C6x DSP booting from ROM memory. The picture is courtesy of TI.

Real-time design flow: introduction

Figure 24 shows a time-ordered view of the various activities or phases that a real-time system developer may be required to carry out during a new system development.

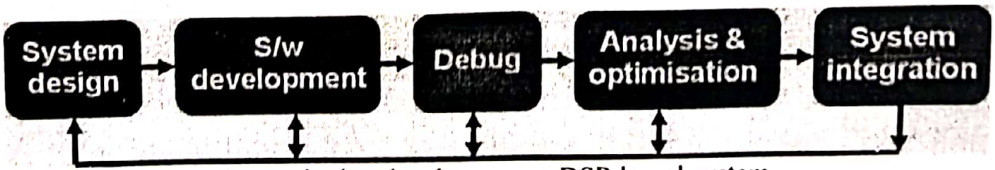


Fig. 24: Activities typically required to develop a new, DSP-based system

The 'system design' phase may include both hardware and software design. For software design, choices such as the code structure, the data flow and data exchange interfaces must be made.
The 'software development' phase includes creating the DSP project and writing the actual DSP code.
The 'debug' phase is a very critical one, where the developer must verify that the code executes

what it was meant to. Some debugging techniques as well as different methodologies available (such as simulation and emulation).
The 'analysis and optimization' phase allows the developer to optimize the system for different goals, such as speed, memory, input/output bandwidth, or power consumption.
Finally, the 'system integration' is the essential phase where the system is integrated within the existing infrastructure and is therefore made fully operational.

Real-time design flow: software development

DSPs are programmed by software via a cross-compilation. This means that the executable is created in a platform (such as a Windows- or a SUN-based machine) different from the one that it runs on, i.e., the DSP itself.
The choice of programming languages is vast, including native assembly language as well as high-level languages such as C, C++, C extensions and dialects, Ada and so on. High-level software tools such as MATLAB and National Instruments allow one to automatically generate code files from graphical interfaces, thus providing rapid prototyping methods.
The code-building tools are very often provided by the DSP manufacturers themselves. Compilers and Integrated Development Environments (IDEs) are also available from other sources, such as Green Hills Software.

Development set-up and environment

DSP executables are developed by using Integrated Development Environments (IDEs) provided by DSP manufacturers; they integrate many functions, such as editing, debugging, project files management, and profiling. Very often the licences are bought on a 'per-project' basis, even if ADI provides also floating (i.e., networked) licences. The development environment for TI and ADI DSPs are called 'Code Composer Studio' and 'VisualDSP++', respectively; they provide very similar functionalities. It should be underlined that TI has recently made available free of charge the compiler, assembler, optimizer and linker to non-commercial users. However, neither the IDE nor a debugger were included, thus the developer must still use the proprietary tools.
Figure 25 gives an example of a typical Code Composer screen. On the left-hand side there is the list of all files included in the software project. At the centre of the screen two windows show the code, as a C file (*process.c*) and as assembly code (Dis-assembly window). A breakpoint has been set and the execution is stopped there.
Below the code windows, two memory windows are also visible, detailing the data present at addresses 0x80000000 and following, and at addresses 0x40000030 and following. Data at address 0x80000002 is of a different colour because its value changed recently. At the bottom of the IDE screen the following item are displayed: a) the Compile/Link window, which details the results from the last code compilation; b) the Watch window, which displays the value assumed by two C-language variables and c) the Register window, which details the contents of all DSP registers. On the right-hand side there are three graphs: the yellow ones show memory regions, while the green one shows the Fast Fourier Transform of data stored in memory as calculated by the

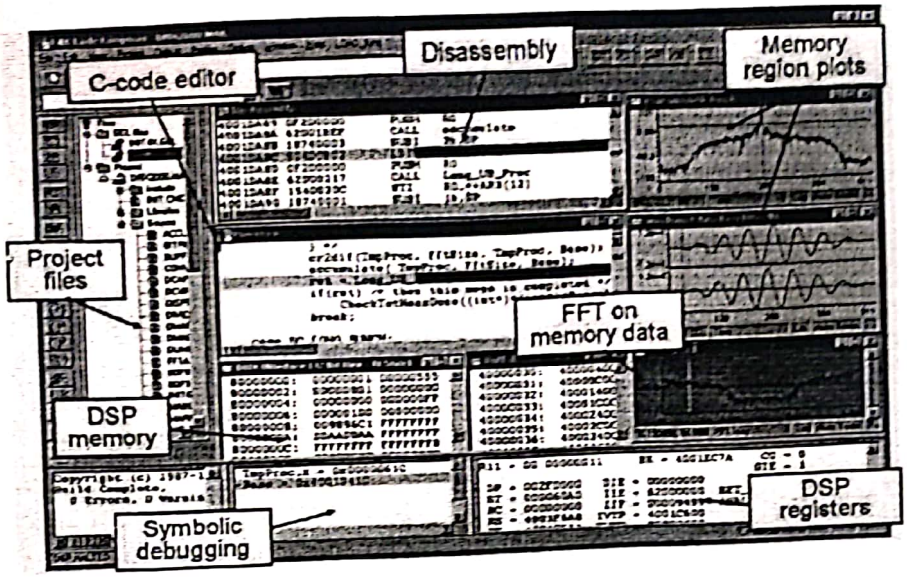


Fig. 25: Screenshot from Code Composer, i.e., the TI DSP IDE. The picture was taken in 1998 from the development of CERN's AD Schottky system.

Figure 26 shows a typical DSP-based system set-up. On the left-hand side the DSP IDE runs on a PC, which is connected to the DSP via a JTAG emulator and pod. This allows one to edit the code, compile it, download it to the hardware and retrieve debug information. On the right-hand side the system exploitation is shown whereby the DSP runs its program and a PowerPC board, running LynxOS and acting as master VME, controls the DSP actions, downloads the control parameters, and retrieves the resulting data.

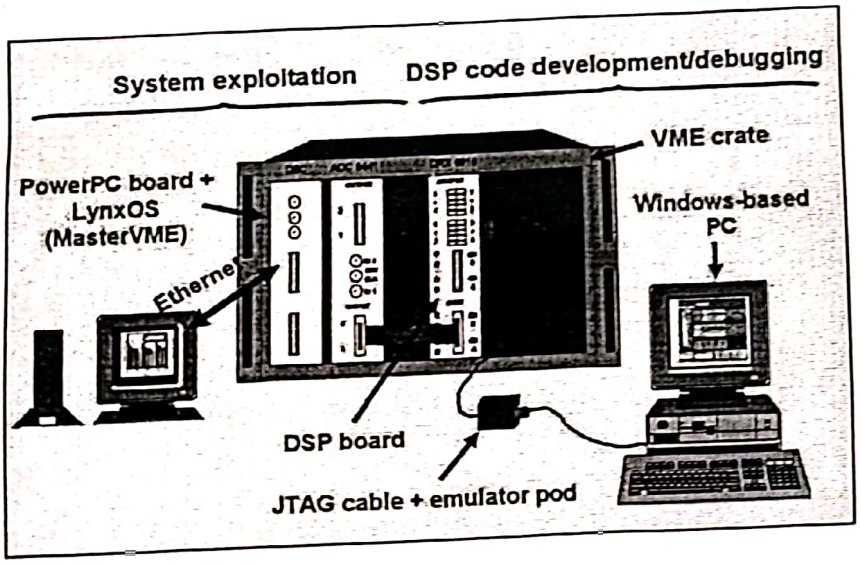


Fig. 26: Typical system exploitation (on the left-hand side) and code development (on the right-hand side) set-ups