M.Tech. Credit Seminar Report, Electronics Systems Group, EE Dept, IIT Bombay, submitted November '02

DSP ARCHITECTURES FOR SYSTEM DESIGN

Vinay Savla (02307910)
Supervisor: Prof A. N. Chandorkar

ABSTRACT:

In this report we discuss a few issues that are important in a digital signal processor. These include issues like bus architectures that are most optimum for a DSP, parallelism and pipelining, fixed and floating point issues, etc. We then see the basic blocks required in any digital signal processor in section 3. The basic computational blocks include multipliers & accumulators (MACs), arithmetic & logic unit (ALUs) and shifters. Other blocks that are required for the proper control of these are program sequencers, data address generators, IO controllers and most important of all memory. In section 4 some issues related to power dissipation are included using an example of FIR filter realization.

1. INTRODUCTION:

With the advent of Digital Signal Processors one can easily classify processors as follows. A General Purpose Processor (GPP) is very efficient in data manipulation. A Digital Signal Processor (DSP) is designed specially to efficiently perform mathematical calculation. DSPs have their architectures optimized so that all the operations it undertakes are finished within as low clock cycles as possible. A GPP cannot perform well for DSP applications. DSP based applications demand a lot of number crunching, very high data bandwidth, real time constraints, attention to subtle numeric effects (in fixed-point implementations), and specialized peripherals/interfaces.

A Digital Signal Processor's average speed performance is far better than a corresponding GPP clocked by the same clock. Thus any GPP architecture would want to move towards DSP architecture. A general trend shows that GPP Architecture approaches to DSP:

- Initially we had Baseline GPPs (moderate performance, no DSP features) like the 8085,8086,80386 etc. These low/moderate performance GPPs with no DSP features perform poorly on DSP tasks because they have poor multiplication throughput, limited memory bandwidth, loop overhead, address generation overhead. Also, being fixed- point processors they lack hardware to support fast overflow protection, convergent rounding, etc.
- Then there were high-performance GPPs with few/no DSP-oriented features like Pentium (P54C). These processors performed well on DSP tasks as they had high clock rates (200+ MHz; 2- 5 X those of typical DSPs), multiplication and fast arithmetic operations, Good memory bandwidth, Loop overhead reduced via branch prediction, and floating point arithmetic block included. However, dynamic features complicate optimization of DSP code and make real- time development difficult.
- Then there were GPPs with major DSP-oriented features. For example, Intel MMX Pentium (P55C). These processors achieve outstanding DSP performance by combining the features of "conventional" high-performance GPPs with new SIMD (single instruction multiple data path) capabilities by partition existing data path (or adding a new partitioned data path), Multiple operations/cycle on small data types (e.g., 4 multiplies), Single-cycle operations on various fixed- point data types.

  In addition there were DSP co-processors like ARM Piccolo co-processor(for ARM7 designs), which gave better performance on DSP tasks as compared to their corresponding processor systems without a co-processor.

  Thus it is observed a Digital Signal Processor is very important block not only for DSP applications but also for General Purpose Processors.

2. DSP ISSUES

2.1 REAL TIME & OFFLINE PROCESSES

In off-line processing, the entire input signal resides in the memory at the same time or is available for the processing. In real-time processing, the output signal is to be produced at the same time that the input signal is being acquired. Here the entire processing that needs to be done on the input samples, should be completed so fast such that the new samples or group of samples that arrive are not lost or output unprocessed. This requires very high processing speeds, which are achieved by DSPs while difficult to achieve in GPPs.

2.2 MAC UNIT

The most typical feature that differentiates a DSP from any GPP is the Multiply and ACcumulate unit. All DSP Algorithms would require some form of the Multiplication and Accumulation Operation of the form

$\sum X1(k,n)\ X2(k,n) = Y(n)$.

Thus arises the need to multiply and add the product to the previous result as fast as possible, ideally single clock. The GPPs would not have a multiply instruction ready for use, and even if it were present, would take lot of cycles to complete a single multiplication.

A DSP has a specialized block called the MAC unit (Multiply and Accumulate Unit) that will multiply the 2 input operands and add the result to a register that has stored all previous results. The figure 1 shows the simplest way to implement the MAC.

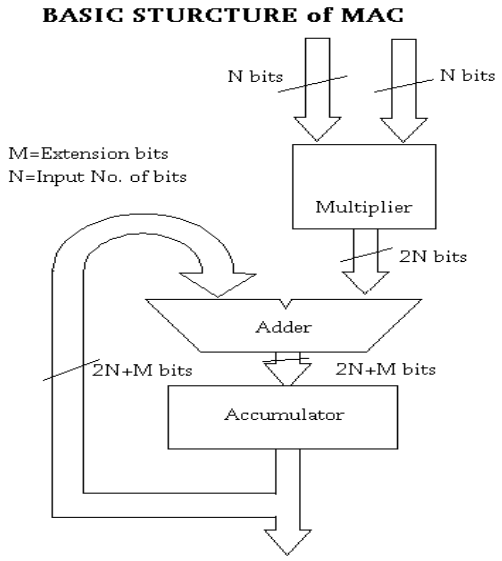

Figure 1: Basic structure of a MAC adapted from [2]

2.3 BUS ARCHITECTURES

Typically a GPP would have VonNeuman Architecture while a DSP would have Harvard Architecture. VonNeuman architecture has a single data bus and an address bus and same memory where the data and instructions are stored. Here instruction fetching would cause a bottleneck. Refer figure 2a. Harvard architecture separates the data and address memories and so the busses to access them are different too. This allows instructions and data to be processed separately and concurrently for improved throughput and thus performance. Refer figure 2b.

For a MAC operation, the 2 operands and a result all need access of the data bus in the same clock cycle. This bottleneck is overcome by using two read buses, which fetch an instruction and a data value or two data values in a single operation and a write bus to simultaneously perform a write operation. Figure 2c illustrates the next level of improvement; the Super Harvard Architecture (SHARC®) DSPs. SHARCs optimize DSP by inclusion of an instruction cache and an I/O controller [1].

A problem with the basic Harvard design is that the data memory bus is busier than the program memory bus. So we might place

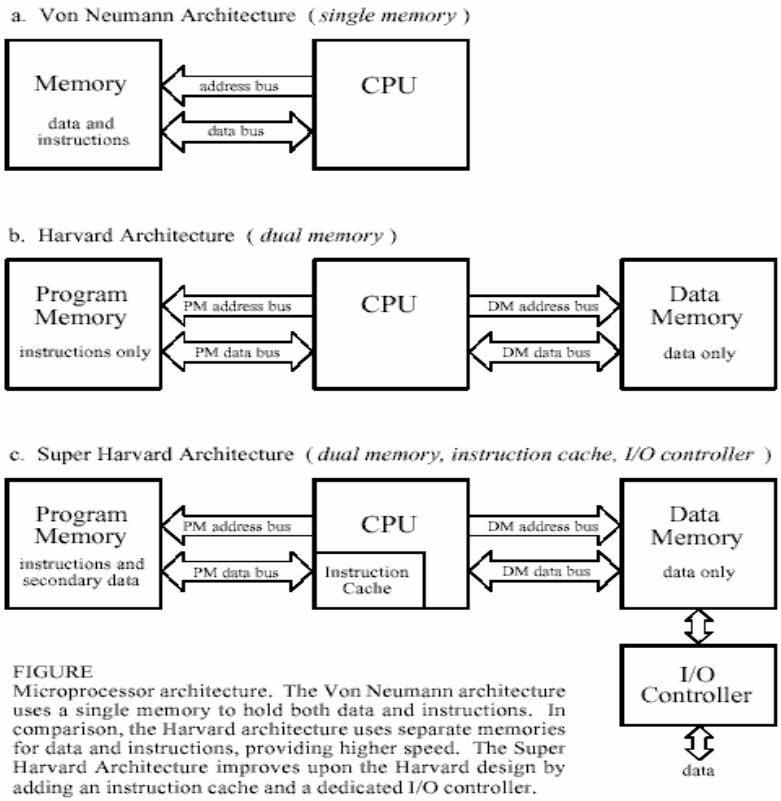

FIGURE
Microprocessor architecture. The Von Neumann architecture uses a single memory to hold both data and instructions. In comparison, the Harvard architecture uses separate memories for data and instructions, providing higher speed. The Super Harvard Architecture improves upon the Harvard design by adding an instruction cache and a dedicated I/O controller.

Figure 2: Bus Architectures [1]

the filter coefficients in program memory, while keeping the input signal in data memory. (This data kept in program memory is called "secondary data"). Now we must transfer one value over the data memory bus (the input signal sample), but two values over the program memory bus (the program instruction and the coefficient). DSP algorithms generally spend most of their execution time in loops. This means that the same set of program instructions will continually pass from program memory to the CPU. Inclusion of an instruction cache, which will store all the recent code, will help here. This means that all of the memory to CPU information transfers can be accomplished in a single cycle: the input signal sample comes over the data memory bus, the coefficient comes over the program memory bus, and the program instruction comes from the instruction cache. This efficient transfer of data is called a high memory-access bandwidth.

All IO devices are connected to dedicated hardware that allows these data to be transferred directly into memory (Direct Memory Access, or DMA), without having to pass through the CPU's registers. Thus if we are taking data input from ADC & feeding our data output to DAC, no cycles are stolen from the CPU for reading the ADC or loading the DAC.

2.4 PIPELINING & PARALLELISM ISSUES

Pipelining: Pipelining enables the more efficient use of on-chip silicon resources, allows multiple operations to occur, and enables much faster cycle times.

What happens in pipelining is explained as follows: Throughput of any system consisting of a series of operations is limited by the single slowest operation in the complete series. So if this single slow operation is further broken down into a number of stages, the intermediate results will be available faster for the next stage. Each stage result(s) are stored in registers that follow that stage, and so the next stage can start operating on these results of the previous stage & simultaneously the previous stage can start operating on the next succeeding set of data. Thus simultaneously no stage of the pipeline is free. DSPs can take advantage of pipelining as the data inputted in a DSP is always continuous. In process which has many level deep pipelines, the startups and terminations are slow while the pipeline fills or empties but when the pipeline is filled all results will appear continuously.

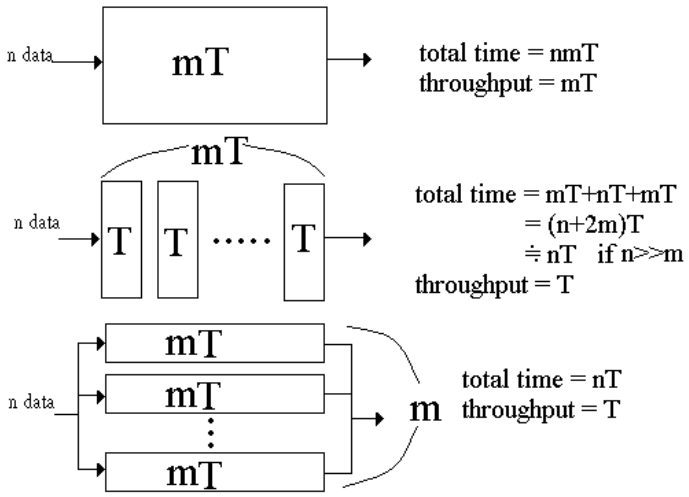

Figure 3: Pipelining & paralleling

Parallelism: To achieve still higher speeds on multiple bus architectures, parallelism is used. Parallelism may require more of on-chip area as there is hardware replication introduced but all the additional blocks are utilized simultaneously to perform a single operation so that the result is achieved faster. Parallelism will eliminate slow startups and terminations, but will require additional data distribution networks. Parallelism works only with processes that are interchangeable.

Example: FFT with pipelining and parallel butterflies see figure 4.

For a N point FFT computation requires $(N/2)\log_2 N$ butterflies. Figure takes the case of 8-point FFT where 12 butterflies are required. Let the T = butterfly time then depending upon the number or hardware butterflies available on chip the net time to compute the FFT can be adjusted.

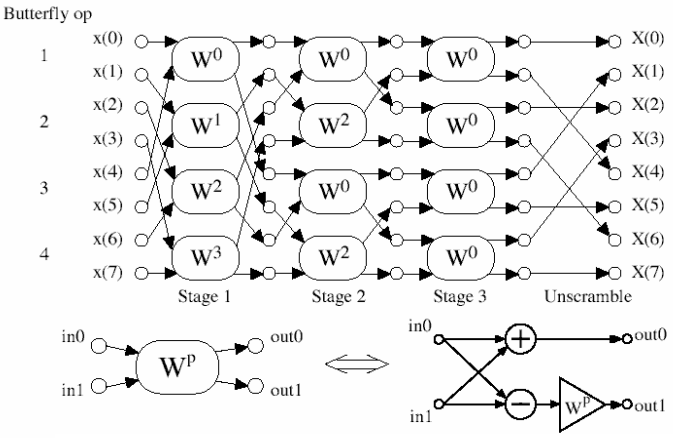

Figure 4: SFG of 8 point FFT

From the table we see that Serial implementation requires the least chip area but is the slowest. The Pipelined implementation requires a little more hardware but is also slow as compared to parallel implementations. Parallelism is fastest but also uses most hardware. Also it requires a very efficient method of data organization and address generation.

Paralleling DSP processors all-together is also another way to speed up processes.
In October 2002, BittWare, Inc. released the Tiger-PMC+, a new DSP card having FOUR 250MHz Analog Devices ADSP-TS101S TigerSHARC DSP, the industry's highest performance floating point DSP, with 6 GFLOPS of sustained processing power.

Architecture	Computation time(t), in units of T*	Number of Butterfly processors
Serial	$(N/2)\log_2 N$	1
Pipelined	N/2	$\log_2 N$
Parallel	$\log_2 N$	N/2
For example consider a 1024 point FFT		
Serial	5120	1
Pipeline	512	10
Parallel	10	512

*The computation times don't include the times for memory access, address generations.

Table 1: FFT Butterfly pipelining and paralleling [2]

2.5 SPECIAL ADDRESSING MODES

All DSP processors require special addressing modes as the data appears in sequential manner. The most common memory addressing mode is register indirect addressing with post-increment, which is used to execute operations on data stored sequentially in memory. Two other special addressing methods common in DSP processors are

Circular addressing: The circular addressing is achieved with the help of circular buffers which are special case of a linked list, where the last data item in the system is linked back to the first data item. Advantage here is we can easily re-use the data items. In case of say FIR filter implementations of order N, last N input samples are required. Each time a new sample appears, the Nth previous sample is discarded and other N -1 samples adjusted accordingly. Now , rather than shifting each of the N samples to different memory location every time a new sample appears, we simply change the pointer for the latest sample in the circular list.
Four parameters are needed to manage a circular buffer.
1. Pointer to the start of the circular buffer in memory,
2. Pointer to the end of the array, or a variable that holds its length,
3. The step size of the memory addressing,
4. The pointer to the most recent sample, which must be modified as each new sample is acquired.

Bit-reversed addressing: The bit-reversed addressing mode is used specifically for implementing the fast Fourier transform (FFT) algorithm. The term bit-reversed comes from the fact that the ordering matches the output one would get from a binary counter if the bits where taken in reversed order (that is the least significant bit first). The problem with the FFT algorithm is that it either takes its input or leaves its output in a scrambled order, so at some point the order of the data has to be rearranged.
On a processor this is achieved as follows: The data address generators have 2 registers, one in which the address is stored in normal form, while bit reversed form in the other. Programs can now use these two address registers in tandem, with one generating sequentially ordered addresses and the other generating bit-reversed addresses, to perform memory reads for the input sequences and memory writes for the output sequence of the same data respectively. Thus, if the input is stored in sequential addresses, the output can be stored in the bit-reversed addresses or vice versa.

2.6 SATURATION ARITHMETIC

Normally if the result of an arithmetic operation in a GPP ALU is outside the data range the result will "wrap around".

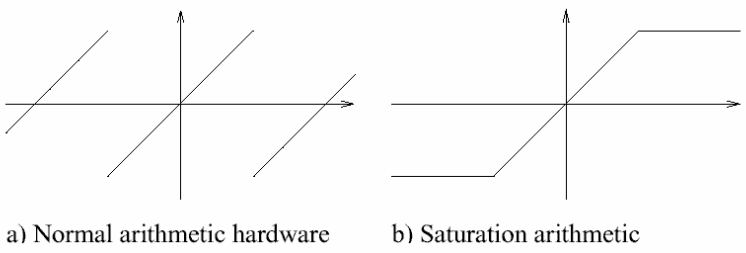

Figure 5: Wrap around and saturation arithmetic: X axis is real values y axis is calculated values [3]

Saturation on the other hand means that if the result is larger than what can be represented with the available number of bits, the output will be the highest possible value and if the result is lower than the lowest value that can be represented, the result will be the lowest possible value. See figure 5.

Using saturation arithmetic reduces distortion due to overflow [2] in the form of glitches as shown in figure 6 and may also prevent parasitic oscillations in recursive algorithms [3]. Saturation arithmetic is basically always necessarily supported for the MAC operations in DSP processors.

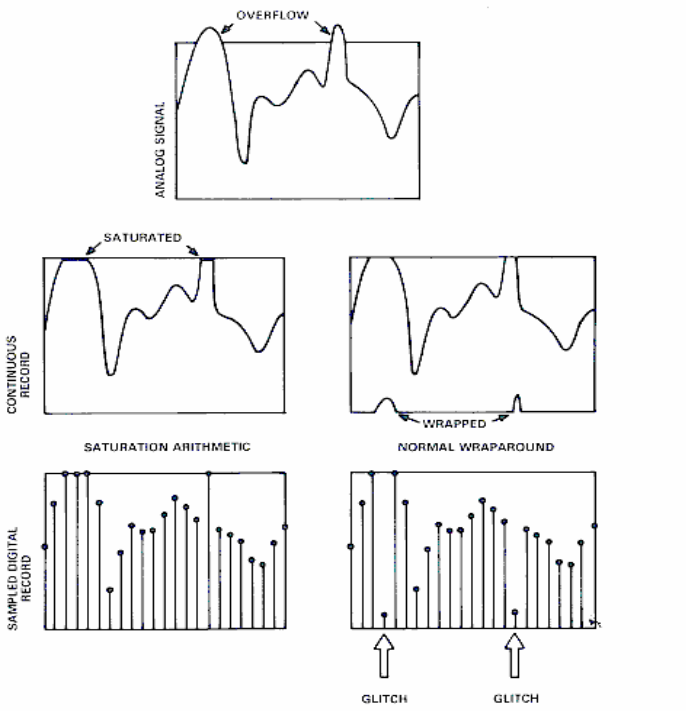

Figure Consequences of overflow in saturation arithmetic (left) as compared with normal wraparound (right). The glitch shown in the sampled data, while accurately representing the wraparound values, is erroneous and hard to filter. By contrast, saturation arithmetic has no glitches.

Figure 6: Wrap around & Saturation arithmetic [2]

2.7 HARDWARE LOOPING

Since many DSP algorithms are based on repetitive computations most DSP processors provide hardware support for efficient looping. Usually there is a loop or repeat instruction, which allows loops to be implemented without spending any extra clock cycles for testing and updating the loop counter, or for jumping back to the start of the loop. This is obviously an advantage as compared to GPPs where each loop has to have a test-and-branch operation which requires at least one clock more. Also nested loops being very common, DSP support hardware for several levels of nested loops. [4]

2.8 FIXED POINT & FLOATING POINT ISSUES

There are two formats to store and manipulate numerical data

Fixed point: The decimal point is fixed at one position which is predefined. The word length decides its range and precision. For example, if word length is 16 bits are used then the four common ways that these possible bit patterns can 2^{16}= 65,536 represent a number are *Unsigned integer*, which can take on any integer value from 0 to 65,535, *Signed integer*, which uses two's complement to make the range include negative numbers, from -32,768 to 32,767, *Unsigned fraction notation*, where the 65,536 levels are spread uniformly between 0 and 1, *Signed fraction notation*, which allows negative numbers too, so 65536 levels equally spaced between -1 and 1. For integers, the decimal is fixed to the extreme right of the number. For fractions, the decimal point is usually fixed to the right of the MSB.

Floating point: Here the data register is divided into two parts, the exponent and the mantissa. As per IEEE Standard 754 for floating point format, word length of 32 bits is divided as 1 bit for sign, 8 bits for exponent & 23 bits for mantissa [2]. So the largest and smallest numbers are $\pm 3.4\times10^{38}$ and $\pm 1.2\times10^{-38}$, respectively. All the values are unequally spaced between these two extremes, such that the gap between any two numbers is about ten-million times smaller than the value of the numbers. This is important because it places large gaps between large numbers, but small gaps between small numbers. Obviously floating point processors are more complicated as compared to fixed point processors, and so are more expensive than fixed point processors. But floating point has better precision and a higher dynamic range than fixed point. In addition, floating point programs often have a shorter development cycle, since the programmer doesn't generally need to worry about issues such as overflow, underflow, and round-off error. The floating point processor itself takes care of all that.

Now let's turn our attention to performance with respect to signal-to-noise ratio. Suppose we store a number in a 32 bit floating point format. As mentioned earlier, the gap between this number and its adjacent neighbor

is about one ten-millionth of the value of the number. To store the number, it must be round up or down by a maximum of one-half the gap size. In other words, each time we store a number in floating point notation, we add noise to the signal. The same thing happens when a number is stored as a 16-bit fixed point value, except that the added noise is much worse. This is because the gaps between adjacent numbers are much larger. For instance, suppose we store the number 10,000 as a signed integer (running from -32,768 to 32,767). The gap between numbers is one ten-thousandth of the value of the number we are storing. If we want to store the number 1000, the gap between numbers is only one one-thousandth of the value.
Fixed point DSPs handle this problem by using an extended precision accumulator.
This is a special register that has 2-3 times as many bits as the other memory locations. For example, in a 16 bit DSP it may have 32 to 40 bits or as high as 80 bits (in the SHARC DSPs for fixed point use). This extended range virtually eliminates round-off noise while the accumulation is in progress. The only round-off error suffered is when the accumulator is scaled and stored in the 16 bit memory. In comparison, floating point has such low quantization noise that these techniques are usually not necessary.

3. BLOCKS IN A DIGITAL SIGNAL PROCESSOR

The block diagram of typical DSP is shown in figure 7. The program memory and the data memory blocks were initially blocks that were outside the DSP but as the level of integration increased, these blocks can now fit on-chip. Each DSP has a large on-chip multi-ported memory. The data is stored in this memory. One port of this memory is connected to the I/O controller that will continuously accept data from ADCs and DACs. ADCs take the analog input from the external world, convert that into digital of appropriate width with appropriate sampling rate and the IO controller will write it into the data RAM. This data, after being processed by the DSP, is once again stored into the RAM from where the IO controller will take data and give it to the DAC. The DAC will provide analog output to the external world. Using mixed signal design, even the ADCs and DACs can be integrated onto the DSP.

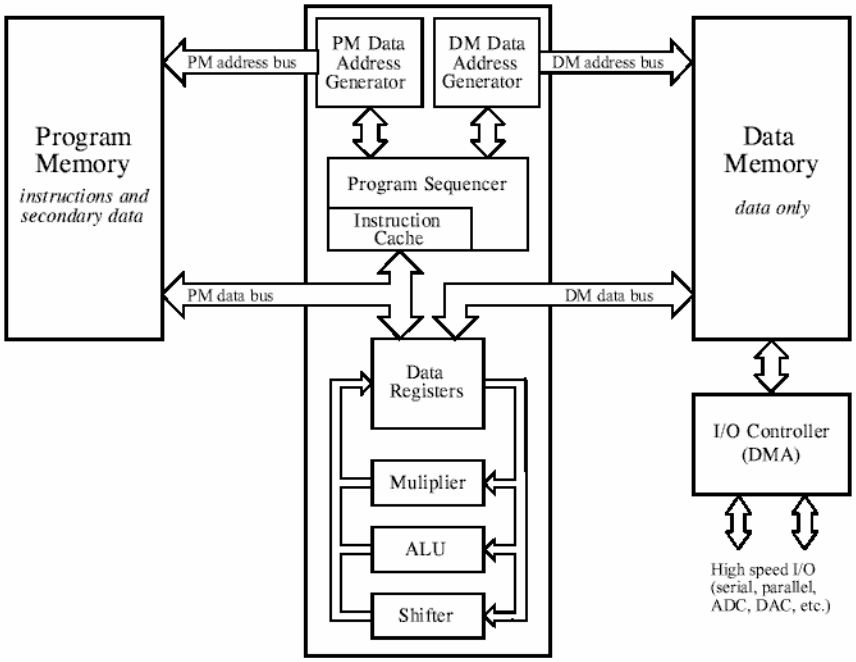

FIGURE Typical DSP architecture, simplified diagram is of the Analog Devices SHARC DSP.

Figure 7: Typical DSP architecture [1].

The program is usually stored on the available on chip ROM which is usually electrically programmable. Thus a code written for a particular application remains intact on the DSP until it is reprogrammed. In most cases even the coefficients (for say FIR filter etc) are stored onto this Program memory as explained earlier. This is because for given application these coefficients don't change. Putting them onto the program memory will speeded up the processing as explained earlier in Bus Architectures in section 2.

3.1 MULTIPLY & ACCUMULATE (MAC)

This is the most important block of the complete DSP. It is composed of an adder, multiplier and the accumulator. See figure 8. Usually adders implemented in DSPs are Carry-Select or Carry-Save adders as

speed is of utmost importance in a DSP. One implementation of the multiplier could be as a parallel array multiplier which computes partial products and adds them to the previous partial products. Other ways are multiplication based on LUTs. There are some Multiplier-less implementations too. Basically the multiplier will multiply the inputs and give the results to the adder, which will add the multiplier results to the previously accumulated results. This operation eases the computation of the most important formula i.e. **?b(n)x(n-k)** which is needed in filters, Fourier analyzers, etc. The inputs for the MAC are supposed to be fetched from some memory location and fed to the multiplier block of the MAC, which will perform multiplication and give the result to adder which will accumulate the result and then if needed will also store the result into a memory location. This entire process is to be achieved in a single clock cycle.
Desired features of MAC are summarized as below:

1. Speed: The faster the better
2. Input format: The MACs should have input format control so that any kind of numbers can be multiplied, whether 2's complement or unsigned, fractional or integer
3. Output precision: Single or double precision options should be available
4. Fixed point vs. floating point tradeoffs
5. For high throughput – use Pipelined Multiplier
6. Use of saturation arithmetic and not Wrap around
7. Appropriate number of extension bits be provided in the accumulator for fixed point implementations

The reason why extension bits are required is explained as follows see figure 8[2]: An N bit by N bit

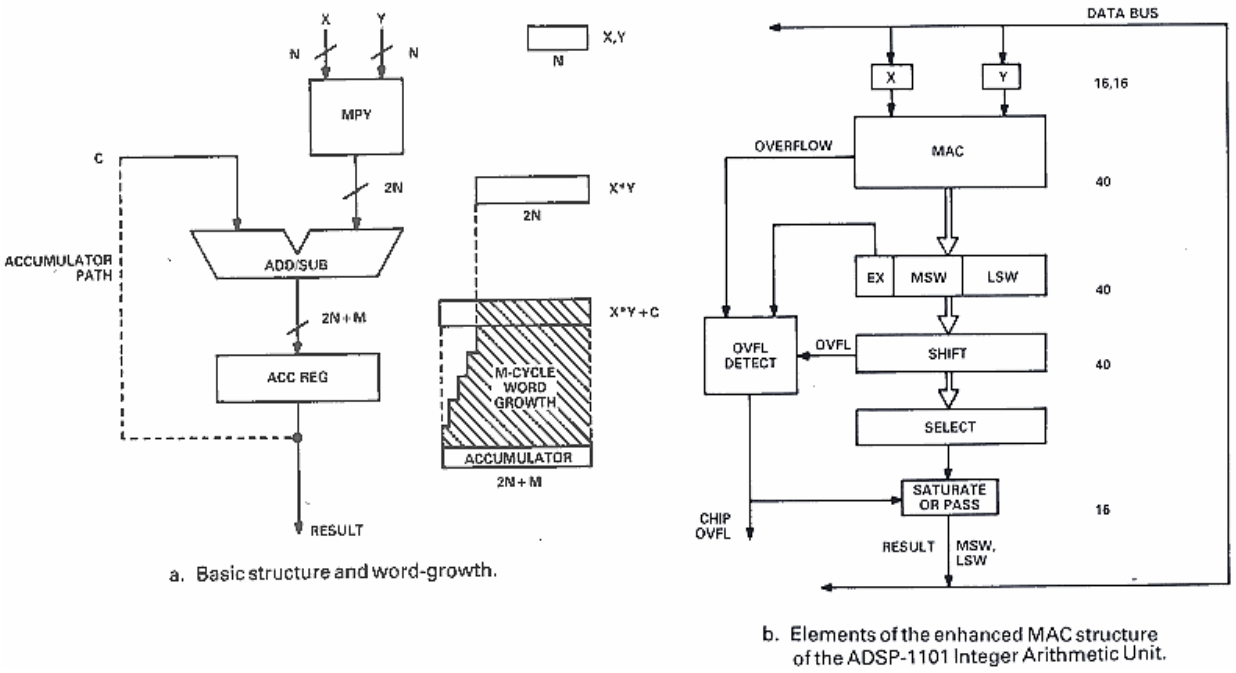

Figure 8: MAC Structure [2]

multiplication will result in a 2N bit result and a N bit by N bit addition will result in a N+1 bit result. Thus if Multiplier results (2N bits) are to be added then the result will be 2N+1. Thus extra guard bit needs to be provided. Again, addition operation will take place a number of times and each time there is a chance that an additional carry is generated. Thus a number of guard bits are required. These bits are called Extension bits. Larger the number of extension bits provided the better. A 32-bit processor provides 35, 40 or as high as 80 bit extended results. This reduces the chances of overflowing when the number of accumulations is large. Obviously larger the number of extension bits more accumulations can be performed before overflow. Even after that if the results exceed the limits then saturation logic is to be followed rather than "wrap-around" otherwise there are chances that the result might bring about parasitic oscillations.

The MAC's output flows via a set of registers which permit its extra width output to be saturated, brought back to data bus width and/or transmitted in double precision as two-word increments, MSW & LSW. The accumulator is divided into LSW, MSW & EX portions. A shifter at the output left shifts to remove the redundant sign bit of the 2's complement multiplication bringing out a properly formatted number for later calculations. In case of single precision, rounding, not truncation, is used. This reduces errors in the result. It is implemented as follows: if the MSB of the LSH(less significant half) is '1' then '1' is added to the MSH and LSH is then truncated else it is simply truncated. For detailed block diagram of a refer figure 8.

In Most cases, 2's complement arithmetic is used for signed numbers, but for low power implementations [7], signed magnitude is preferred to 2's complement arithmetic. This is because on the basis of the study performed by Lewis [7], it was shown that in 2's complement arithmetic the switching activity is more as compared to signed magnitude arithmetic. Refer table. Since power consumed depends on switching activity, lesser the switching activity lesser is the power dissipation. And so signed magnitude was preferred.

POSITON	2's complement	Signed magnitude
Data input	7.5	5.8
Coefficient input	8.3	5.7
Multiplier output	20.7	10.9
Accumulator output	14.8	11.5

Table 2: Average no. of transitions per operations [7]

3.2 ARITHMETIC AND LOGIC UNIT

For an ALU in DSP, the following features are desired:

1. Adequate accuracy for High precision computation
2. Arithmetic functions: Add, subtract, Add with carry, subtract with borrow, absolute value.
3. Logical functions: AND, OR, eXOR, Negation.
4. Output flags: zero, equal, less than, greater than, carry, parity, overflow etc.
5. Data move and arithmetic in same cycle, 2 operands loadable into ALU on each cycle.
6. Adequate feedback paths: output to input (accumulate),
7. Pipeline/ Transparent options
8. Association with shifters for scaling inputs
9. Fast double-precision capability

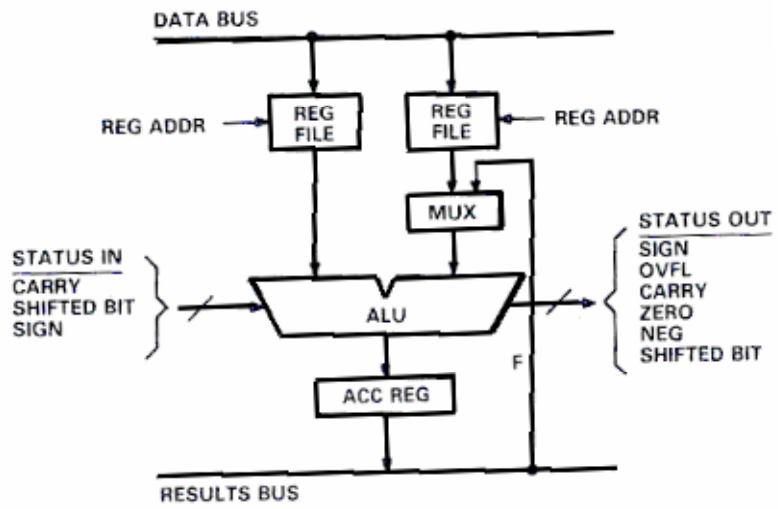

Figure 9: DSP ALU Structure [2]

An ALU for a DSP aims mainly at speed. An ALU can operate in parallel with other blocks like MAC, Shifters etc. So a large number of operations can be initialized by a single instruction and all results obtained almost simultaneously.

The Flags indicate the status of the result after an ALU operation. After each ALU operation the flags are refreshed. These are mostly similar to a GPP Flags. Sign and Zero flags are usually logically combined to give the ?, = & = functions.

In a GPP, the user has to check the results in the flags and make corresponding changes in the results after properly inspecting the flags. The special feature of DSP Flags is that other units inspect them and make changes on results automatically without user interference i.e. no additional instruction is required for such operations especially the overflag monitor by a Shifter.

Arithmetic plus conditional shift: An ALU which can conditionally shift its results up or down during the same cycle, depending upon the state of an input flag, enhances the ALU speeds during many DSP

operations. It is mainly helpful for in decimation-in-frequency FFT algorithms. In this case, the DFT block size decreases by ½ for every stage of the FFT. When completed, the DFT block size will be two and the address offset one. The lack of extra cycle's requirement in conditional shift is an asset: parallel operations of shifter need not halt for data to stay in synchronization [8].

As mentioned earlier, in addition to single cycle computations, even the data movement is required to be achieved in the same cycle. This requires the use of dual ported registers which provide the ALU input and output while simultaneously doing bus access. Dual ported registers can be, both written to and/or read from in one cycle. The previously stored value is read out of the register at the beginning of the cycle and the new value loaded into it at the end of the cycle. This saves unnecessary wastage of time to load the register with new data for the successive operation. But for this the data needs to be already available. This operation is advantageous in cases where there continuous data to be processed. If the internal data path is so optimized, the accumulator output can be an input for the next ALU cycle if the feedback path is activated.

Some of the latest ALUs have special features like division capabilities e.g. ADSP2191. Division capability is helpful in algorithms like Linear Predictive Coding in Speech Processing for the computation of inverse of a matrix. Since division is required in only specific DSP applications, not all, a separate divider is not kept aside like a multiplier is. But an ALU with, may be slow, but a ready-to-use division instruction will always have an upper hand as compared to the traditionally implemented division routines [9] [10].

3.3 SHIFTERS

A barrel shifter is a block of combinational logic that takes an N-bit input value and provides as output the N-bit value shifted left or right by P bits. The great advantage of a barrel shifter is of course speed. It does in a single cycle what it would take P cycles to achieve in GPPs without a barrel shifter. Right shifting in binary by P bits is really the same as division by 2P, and left shifting is the same as multiplication by 2P. Figure 10 illustrates the generic architecture of a barrel shifter.

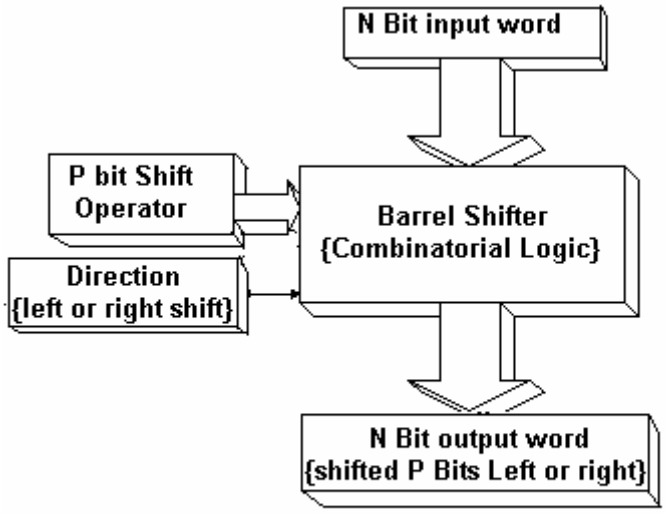

Figure 10: DSP Shifter, adapted from [10]

The main functions of a shifter in a DSP are summarized as follows:

1. Pre and/or Post scaling: To maintain accuracy without the benefit of a floating-point data path, fixed-point DSP processors have a good support for shifting operations. Many DSPs provide barrel shifters at both the input (pre-scaling) and output (post-scaling) of their MAC logic. The pre-scaler is generally activated by a pseudo instruction used in the code, for example, MACD, X, Y, 4, which divides the input data by 4 BEFORE the multiply occurs. The coefficient remains unaltered. An output post-scaler is generally set up prior to execution of a block of code (i.e., digital filtering, convolution, or an FFT) in order to scale the output data from the MAC operation by a specific factor. This is intended to reduce the probability of overflows, which can occur during long sequences of MAC operations. Use of this automatic scaling technique partially eliminates the overheads otherwise incurred when having to check your results for overflow and underflow (the DSP generally provides one or two flags, which need to be checked manually for overflow and underflow by your program after each MACD execution). If an overflow is indicated, you can either abort the program or recalculate the results.
2. Normalization & Denormalization: Normalization involves conversion from fixed point to floating point systems by generation of mantissa and exponent. This is a two process. The first one involves exponent detection which calculated the number of shifts needed. The next step provides this value to the shifter for

actually shifting the number. Denormalization involves conversion of floating point into fixed point systems i.e. inverse of normalization. The exponent provides number of shifts required in the mantissa.
3. Block floating point[10]: A block floating-point format enables a fixed-point processor to gain some of the increased dynamic range of a floating-point format without the overhead needed to do floating-point arithmetic. However, some additional programming is required to maintain a block floating-point format. It can also be used in floating point processors.

3.4 DATA ADDRESS GENERATORS

As explained earlier, Super Harvard Architecture with modifications is the most suited architectures for DSPs. Such an architecture has multiple busses to access the memory viz. Data memory bus & program memory bus. This is obviously an advantage w.r.t. increase in speed but this would demand additional hardware that would keep track of address on each of these busses.
The Data Address Generator (DAGs) keeps a track of the addresses of the input data and the coefficients that are stored in the data and program memories respectively. Now the CPU no longer has to bother about the addresses of the data, the DAG will do that task. This is most useful when a set of addresses are used repeatedly, for example, scanning a data array or in FFT computations. The address generation unit would require its own arithmetic unit to increment or decrement the addresses, and/or to add/subtract offsets from the base address in case of non-sequential addresses.
The increment/decrement step size for address in each case can be predefined as 1,2,4 or 8 and so on depending upon the size of each data item [9] [10].
Any DAG would require at least the following types of registers. The types of registers are:

- Index registers: An index register holds an address and acts as a pointer to memory.
- Modify registers: A modify register provides the increment or step size by which an index register is pre- or post-modified during a register move.
- Length and Base registers: Length and base registers setup the range of addresses and the starting address for a circular buffer.

These registers hold the values that the DAG uses for generating addresses.

3.5 PROGRAM SEQUENCER

A program sequencer generates instruction addresses like the DAG generated data addresses. It is this block that decides the exact program flow. A program sequencer removes overhead by keeping track of program-counter incrementing, conditional branching and looping, subroutine handling and interrupt handling.
Program flow in the DSP is usually linear with the processor executing program instructions sequentially. But this linear flow does vary when the program uses non-sequential program structures, such as those illustrated in figure 11. Non-sequential structures direct the DSP to execute an instruction that is not at the next sequential address. These structures include:

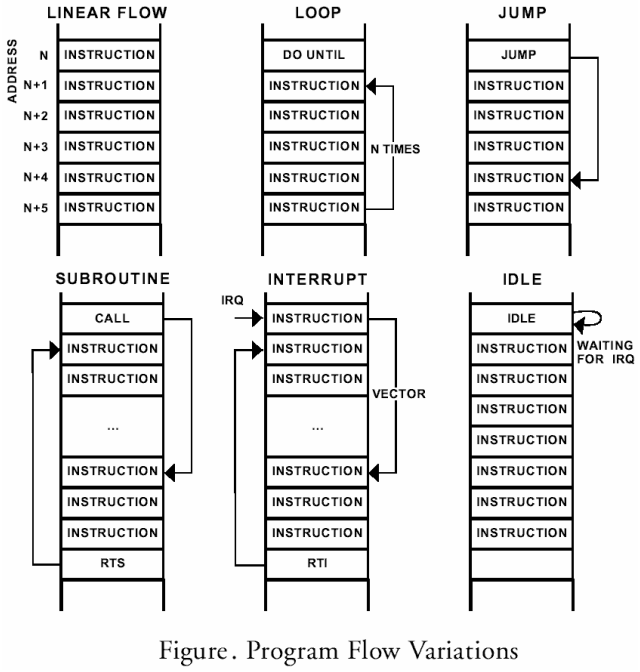

Figure. Program Flow Variations

Figure 11: Program Flow Variations [9]

- Loops: One sequence of instructions executes several times with near-zero overhead.
- Subroutines: The processor temporarily interrupts sequential flow to execute instructions from another part of program memory.
- Jumps: Program flow transfers permanently to another part of program memory.
- Interrupts: Subroutines in which a runtime event triggers the execution of the routine.

- Idle: An instruction that causes the processor to cease operations, holding its current state until an interrupt occurs. Then, the processor services the interrupt and continues normal execution.

The sequencer manages execution of these program structures by selecting the address of the next instruction to execute. As part of this process, the sequencer handles the following tasks: increments the fetch address, maintains stacks, evaluates conditions, decrements the loop counter, calculates new addresses, maintains an instruction cache & handles interrupts.

The most important part in a program sequencer is the cache. DSP algorithms generally spend most of their execution time in loops. For such structures cache can be of great advantage. The cache sizing and cache architecture are two other important areas that need to be properly analyzed.

A program sequencer must have adequate stack, large enough to handle nested subroutines and several levels of interrupts. Like in a GPP, both software testing (flags and counter) and hardware testing (interrupt) can cause a branch. Proper prioritization is required along with options of masking and unmasking particular/all interrupts. Sequencer pipelining is not recommended as that might delay the time to service the routine for a particular interrupt. The hardware looping provides no overhead to check for conditions during looping statements. The program sequencer must be capable of accessing both internal memory and external memory, if interfaced.

3.6 MEMORY

By now it is quite clear that a proper choice of memory is very important for DSPs. For completing the execution of an instruction, say a MAC, in a single cycle will require that all the following operations finish in a single cycle:

a. fetching of instruction from program memory,
b. decoding,
c. reading of coefficients from program memory,
d. reading of data from data memory,
e. multiplying and accumulating the results in the accumulator (i.e. actual execution in case of any other instruction),
f. writing the results back to the data memory.

For more complex operations like FIR filter implementation [11] more steps are required.

Here it is seen that 4 of the 6 steps require access to the memory. Separating the program and data memory is one way of increasing speed. Still 2 accesses are required each to, program and data memory per cycle. This requires that the memories used be very fast so as not to delay the cycle execution and to get a high throughput from the system. So memories with very low access time are required. SRAMs are most suited for that purpose. The load on memory can be reduced by providing an extensive set of registers on the chip so that separate memory accesses are reduced.

Cache memory also lessens the demand of high number of memory accesses to the program memory Pipelining the data memory can reduce the speed requirements of the memory. Usually data to be read from or written to the memory are sequential in nature and so pipelining the data memory helps. In addition to that, data memories are dual ported, for simultaneous and independent access by the DSP and IO controller.

Thus a mixture of fast and slow memories reduces the system costs. Program memory must be fast, since the system clock speed is decided by the sum of sequencer delay and memory access time. However data memories can be slower if pipelining is made available.

3.7 IO CONTROLLER

The IO controller on a DSP allows it transfer data to or from ADCs, DACs, other DSPs, etc. It is basically this block that is responsible for the DSPs communication with the external world. In most GPPs, IO operations have to be a part of the code to perform any operation. If that were allowed in DSPs then it would have tremendously deteriorated the speeds. This is because the IO operations are comparatively slower as compared to individual computational operations. The computation on each sample(or set of samples) has to complete before the next sample (or set of samples) appear.

Thus in DSPs, the IO block and the processing blocks are kept separate. The IO processor once programmed will independently perform data transfers from the memory to the IO device and vice versa without

interfering in the DSP's operation. This it does with the help of a DMA controller within itself and the fact that the memory is dual ported.

4. POWER DISSIPATION ISSUES taking FIR FILTER REALIZATION EXAMPLE

In a CMOS implementation the major source of power dissipation is dynamic power dissipation $P_{dynamic}$ which is given by $\mathbf{P_{dynamic} = C_{switch} . V^2 . f}$

C_{switch} is the product of physical capacitance being charged or discharged and the corresponding switching probability or circuit activity($C_{switch} = C_{physical} . a$), V is the supply voltage, f is the operating frequency and a is the switching activity of the device.

4.1 SOURCES OF POWER DISSIPATION

The main sources of power dissipation of a device are as follows [12]:

- DATA & ADDRESS BUSSES
  - Hamming distance between successive values on the busses: The address and data busses in any processor see very large capacitive loads and hence signal switching on these busses will generate significant amount of power. Signal switching is directly depends on the Hamming distance between successive values on the busses.
  - Number of adjacent signals toggling in opposite direction: The power due to intersignal capacitance varies depending upon the adjacent signal values. The current requirements for signal to switch between 5h (0101b) and Ah(1010b) is about 25% more than the current requirement for the signal to switch between 0h(0000b) and Fh(1111b)
- MULTIPLIER
  - Hamming distance between successive inputs ("transition density"): Transition density is a measure of circuit activity.
  - Number of 1s in the inputs (for an array multiplier): For an array multiplier it can be shown that power dissipation is directly dependent on number of 1s in the input.
- MEMORIES
  - Number of reads and writes: More the number of times the memory is accessed more will be the dissipation of power.
  - Hamming distance between successive address/data values: Again the hamming distance will decide the signal activity and thus the power dissipation.

4.2 TECHNIQUES TO REDUCE POWER DISSIPATION

A few techniques to reduce the power dissipation in FIR Filter implementations [12] are given below:

1. *Using Gray Coding*: All the sources of power dissipation mentioned above show that hamming distance between successive values is the major source of power dissipation. Thus if GRAY coding is used, the hamming distance between successive addresses reduces to 1. Obviously this would require code converters at the memory and CPU ends but if the power dissipation is reduced then it is worth using chip area. But this too can be reduced by directly saving the programs in gray order rather than in binary. This will save the need of a gray to binary converter at the memory end.
2. *Using T0 coding*: We know that most accesses to the memory are consecutive in nature. This Fact is taken advantage of by sending the address just once and then generating them at the memory end with the help of a counter. This saves the power of driving the entire address bus (which has huge capacitive loads associated with it) by driving just a single *counter-increment signal*.
3. *Using Bus-invert coding*: As mentioned earlier, it is seen that power dissipation also depends upon the number of ones being transmitted. This fact can be taken advantage of by finding the number of 1s in a sequence and sending the inverted sequence if the number of 1s is greater than the number of zeros in the original sequence. Again, an additional signal called *bus invert* will have to be transmitted
4. *Using Multirate-Architectures*: This involves implementing a particular filter, say FIR, in terms of its decimated sub-filters. So that each individual sub-block works at the lower sampling rate i.e. requires lesser multiplications and additions as compared to the full higher rate filter. Example: [12] For direct form FIR

structure: N multiplication and N-1 additions are required while for Multirate architecture 3N/4 multiplications and (3N+2)/4 additions are required. Power reduces by about 25%.

5. Using techniques such as *transposing*: In case of FIR filter if the filter is implemented as a transposed SFG rather than the original SFG, it is seen that transposed structure involves multiplying all the coefficients by the same input data x[n]. Now during filter computation, one of the multiplier is always a constant, so the power dissipation is reduced.

6. *Coefficient scaling*: Scaling the output of a filter will maintain the shape of the FIR filter characteristics but the magnitude will become the scale factor rather than 1. $K \cdot Y_n = K. S (A_i \cdot X_{n-i}) = S ((K \cdot A_i) \cdot X_{n-i})$. So if a scaling factor K is selected such that the coefficients $(K \cdot A_i)$ have minimum hamming distance, power dissipation can be reduced.

7. *Selective Coefficient Negation*: The number of 1s in A[i] and -A[i] (where negatives are stored in 2's complement format) can differ significantly. This fact can be taken advantage of. If rather than storing A[i]s we store –A[i] for all such numbers for which –A[i] has lesser number of ones and in the MAC instead of multiply/add we use multiply/subtract, the result will be same. But significant power dissipation reduction is observed in multiplier and coefficient data bus power.

Using different coding (points 1 to 3) will require hardware modifications. With available hardware, say a programmable DSP, points 4 through 7 fit in very well. While point 4 will need to be considered while programming only, points 5 through 7 need consideration while programming & writing the coefficients into the memory.

4.3 DIFFERENT IMPLEMENTATIONS OF AN FIR FILTER: The figure 16 shows the basic FIR structure and figure 12 & 18 how these get implemented on a MAC based DSP. Here the MAC cycle has to repeat until the coefficient memory is scanned from A[0] to A[N-1] and correspondingly input data memory from X[n] to X[n-N+1] to get a single Y[n]. This cycle has to repeat for as many times as the number of input samples are present or as many times as the number of output samples are required.

A basic 4-tap FIR filter would take 9 clock cycles for a non-pipelined MAC and in 6 clocks for a pipelined MAC to compute a single Y[n] using 1 multiplier and 1 adder as shown in figure 17 & 14.

Taking a transpose of the FIR filter will give the result still faster as seen in figure 13 & 15.

Note that in the above case, the values for the inputs X[n], X[n-1], X[n-2] & X[n-3] need to be stored into the data memory. These inputs are being continuously updated by the IO controller without any intervention by the main processor. If these inputs are stored in a circular buffer then individual data items need not be shifted in the memory, simply the pointer to X[n] can be updated as explained earlier. The Data address generator is responsible for this. The coefficients A[0], A[1], A[2] & A[3] are to be stored in the program memory along with the program. Once the program sequencer loops through the entire loop for the 1st time it will store it in the cache and then on it will retrieve the program instructions directly from the cache rather than calling the memory again and again. The Program Sequencer will handle all this.

$$Y[n] = \sum_{i=0}^{N-1} A[i].X[n-i]$$

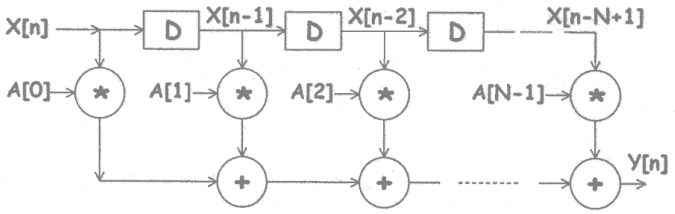

Figure 16: FIR Flter Implementation [13]

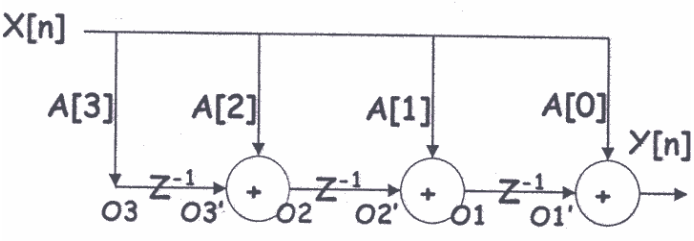

Figure 15: Transposed FIR Filter[13]

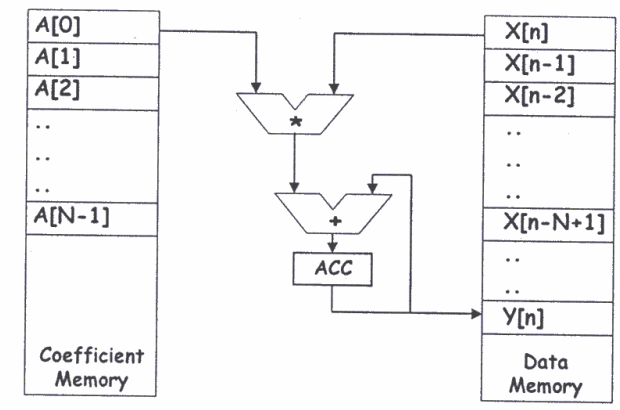

Figure 18: MAC Based Implementation of FIR Filter [13]

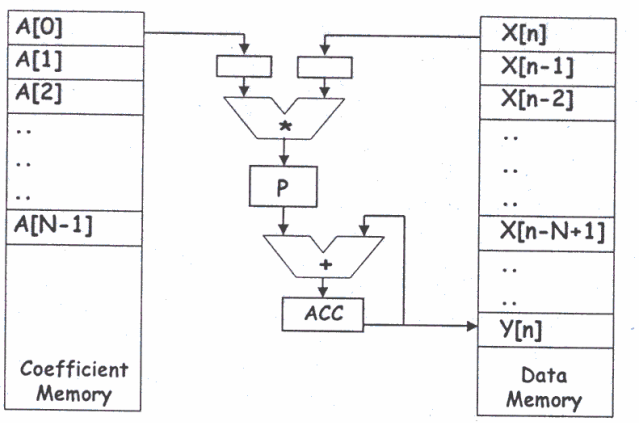

Figure 12: MAC based implementation of Pipelined FIR Filter [13]

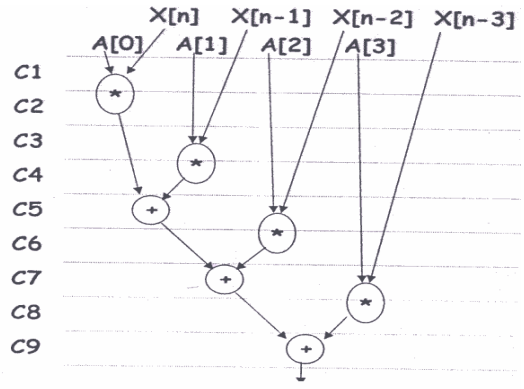

Figure 17: Schedule for 4 Tap FIR Filter [13]

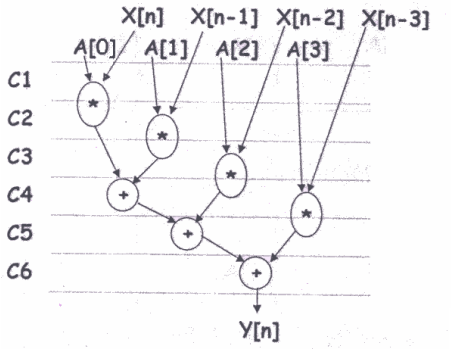

Figure 14: Schedule for 4 Tap FIR Filter with pipelined Multiplier [13]

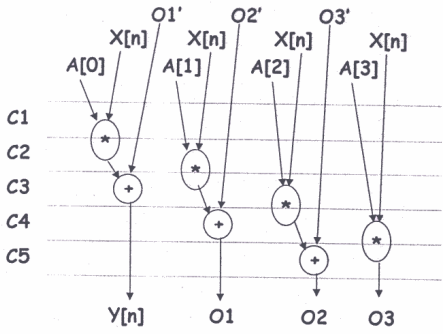

Figure 13: Schedule for transposed 4 Tap FIR filter [13]

5. CONCLUSION

Digital signal processing is one of the core technologies in rapidly growing application areas such as wireless communications, audio and video processing, and industrial control. As GPP architectures are moving towards a DSP architecture a time will come when DSPs will complete replace GPPs. There are a number of DSP manufacturers in the market today all providing different combinations of features. The right choice DSP depends on the application. In this report, a review has been made on the main DSP issues for designing/selecting a DSP for a particular application based on the given requirements. Constraints laid down by VLSI Technology in terms of Speed, Chip area and power dissipation demand all designs to optimize these three parameters based on given requirements.

REFERENCES

[1] S. W. Smith, The Scientist and Engineer's Guide to Digital Signal Processing, California Technical Publishing, 1997, pp. 503-534.

[2] R. J. Higgins, Digital Signal Processing in VLSI, Analog Devices Technical Reference, Prentice Hall, Eaglewood Cliffs NJ, 1990, pp. 235-355.

[3] L. Wanhammar, DSP Integrated Circuits, Academic press, 1999, pp. 191-192.

[4] J. McGuire, “For Efficient Signal Processing in Embedded Systems, Take a DSP, not a RISC”, Analog Dialogue, Vol. 30, No. 3, Analog Devices Publications, 1996, http://www.analog.com/library/analogDialogue/archives/30-3/efficient.html , Oct. 2002.

[5] ---, Choosing a DSP Processor, Berkeley Design Technology, Inc. http://www.bdti.com/articles/choose_2000.pdf , Oct 2002.

[6] E. E. Swartzlander, Jr., “VLSI Signal Processing Systems”, Kluwer Academic Publishers, 1986.

[7] M. J. G. Lewis, “LOW POWER ASYNCHRONOUS DIGITAL SIGNAL PROCESSING”, thesis submitted to the University of Manchester for the degree of PhD in the Faculty of Science & Engineering, http://www.selftimedsolutions.co.uk/papers/cadre.pdf , Oct. 2002.

[8] K. Larson, “TMS320C25 Logical Shifts in Parallel with ALU Operations”, TMS320 DSP Designer’s Notebook, Application Brief: SPRA207, Digital Signal Processing Products, Semiconductor Group, Texas Instruments, Jan. 1993.

[9] ADSP-219x/2191 DSP Hardware Reference, Analog Devices, July 2001.

[10] ADSP-218x DSP Hardware Reference, Analog Devices, Inc., Feb. 2001.

[11] E. A. Lee, “Programmable DSP Architectures: Part 1”, IEEE ASSP Mag., Oct. 1988.

[12] M. Mehendale, S. D. Sherlekar, and G. Venkatesh, “Low-Power Realization of FIR Filters on Programmable DSP’s”, IEEE Trans. Signal Processing, Dec. 1998, pp. 546-553.

[13] M. Mehendale and G. Venkatesh, “VLSI Architectures for Digital Signal Processing” in National seminar, VLSI: System, Design & Technology, IIT Bombay, Dec. 2000.