ANALOG DEVICES

AN-240
APPLICATION NOTE

ONE TECHNOLOGY WAY • P.O. BOX 9106 • NORWOOD, MASSACHUSETTS 02062-9106 • 617/329-4700

Considerations for Selecting a DSP Processor ADSP-2101 vs. WE DSP16A

by Bruce Wolfeld

INTRODUCTION

Digital signal processing systems contain high-performance numerical processing units that capable of performing rapid computations. The numerical performance of a DSP system is measured by the processing unit's capabilities in the following areas:

- Fast and flexible arithmetic with extended dynamic range
- Efficient operand access
- Circular buffering capabilities
- Overhead required for program loops

As with any microprocessor design, you, the system designer, must consider other factors including:

- Interface to external devices
- Quantity and speed of external memory
- Instruction set
- Development tools available for system debug

This application note first discusses the numerical processing capabilities of the Analog Devices ADSP-2101 and the AT&T DSP16A. The second part discuses other factors that you should consider.

ARITHMETIC CAPABILITIES

One indication of good arithmetic architecture is the ability to perform a wide range of arithmetic computations. These computations should be flexibly handled so that the algorithm implementation preserves the order of the arithmetic operations and operands. If the arithmetic architecture is too special purpose, this is impossible. Rearranging an algorithm to fit the architecture requires extra programming, increases the possibility of error and may increase product development time.

Arithmetic Architecture

Figure 1 shows a block diagram of the arithmetic section of the ADSP-2101 while Figure 2 shows that of the DSP16A. Both of these devices utilize a modified Harvard architecture which can fetch data operands from both program memory and data memory. By utilizing on-chip memory, the ADSP-2101 can fetch an instruction from program memory and fetch/store two data operands from memory in a single cycle. The DSP16A requires one cycle to fetch an instruction from program ROM. Two cycles are required if the instruction and immediate data both reside in the program ROM.

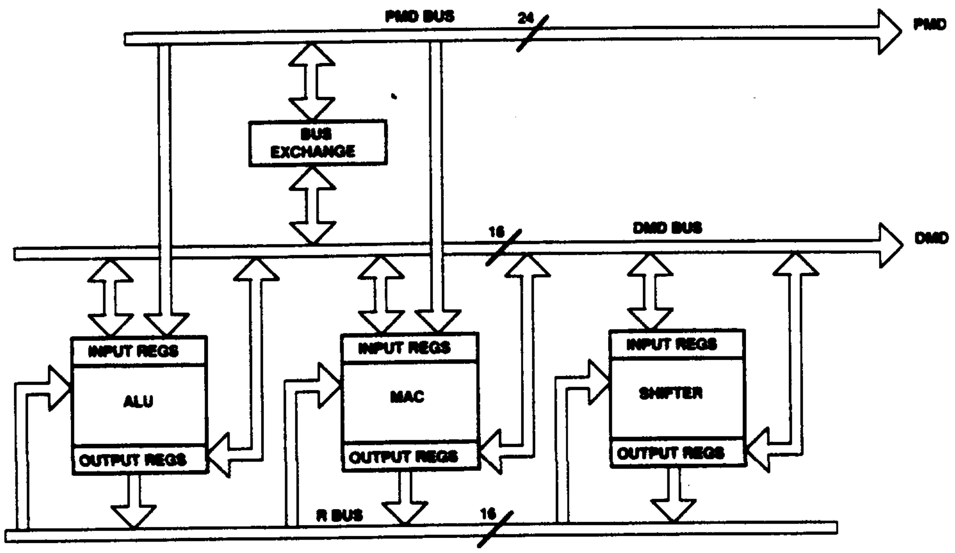

Figure 1. Block Diagram of Arithmetic Section of the ADSP-2101

The arithmetic section of the ADSP-2101 architecture was designed so DSP algorithms are easily coded and rapidly executed. Unlike many DSP processors, the ADSP-2101 uses an algebraic notation for a variety of multi-function instructions. These operations exploit the inherent parallelism of the ADSP-101 architecture by providing combinations of register moves, memory transfers, and conditional computation. Each program statement assembles into a single 24-bit opcode that can execute in a single cycle. There are no multicycle instructions in the ADSP-2101 instruction set.

The ADSP-2101 has three independent computational units: an arithmetic/logic unit (ALU), a multiplier/accumulator (MAC), and a barrel shifter. They are connected by the result bus (R bus) so that the output of any computational unit can be used directly as the input for itself or any other unit on the next instruction cycle. In addition, the ALU and MAC are directly connected to both the program and data memory busses. Operands for ALU and MAC operations can come from program memory, data memory or any of the result registers. The arithmetic units have features that allow for 32-bit or 64-bit computations.

The arithmetic section of the DSP16A contains a multiplier unit with a scaling shifter and a ALU/shifter unit. The multiplier has three input registers and one output register. While all three input registers can be loaded from data memory, only the x register can be loaded from program memory. The ALU/shifter can operate on data in the y registers, the multiplier output register, or the accumulator registers. The y registers and the accumulator registers are accessible only through data memory. If arithmetic results are to be stored in external memory, they must first be loaded into one of the accumulator registers.

ADSP-2101 ALU

The ADSP-2101 ALU has two X and two Y input registers: AX0, AX1, and AY0, AY1. ALU operations are performed on any X-Y register combination. These registers may be loaded from program memory, data memory, the counter or a variety of other processor registers. ALU results can be stored in either the ALU result (AR) or the ALU feedback (AF) register. AR and AF are available as the X and Y operands (respectively) in subsequent ALU operations. In addition, the result registers of the MAC and barrel shifter can be used as the X input to the ALU.

The ALU performs mathematical operations on 16-bit data operands. An internal carry bit is always updated during a mathematical operation. Addition with carry and subtraction with borrow instructions are provided for implementing 32-bit arithmetic.

Two operands can be fetched or stored in parallel with an ALU operation. By fetching operands while executing ALU operations, the processor can perform one ALU operation per cycle without speed penalties due to operand fetching. All ALU operations can execute in a single cycle.

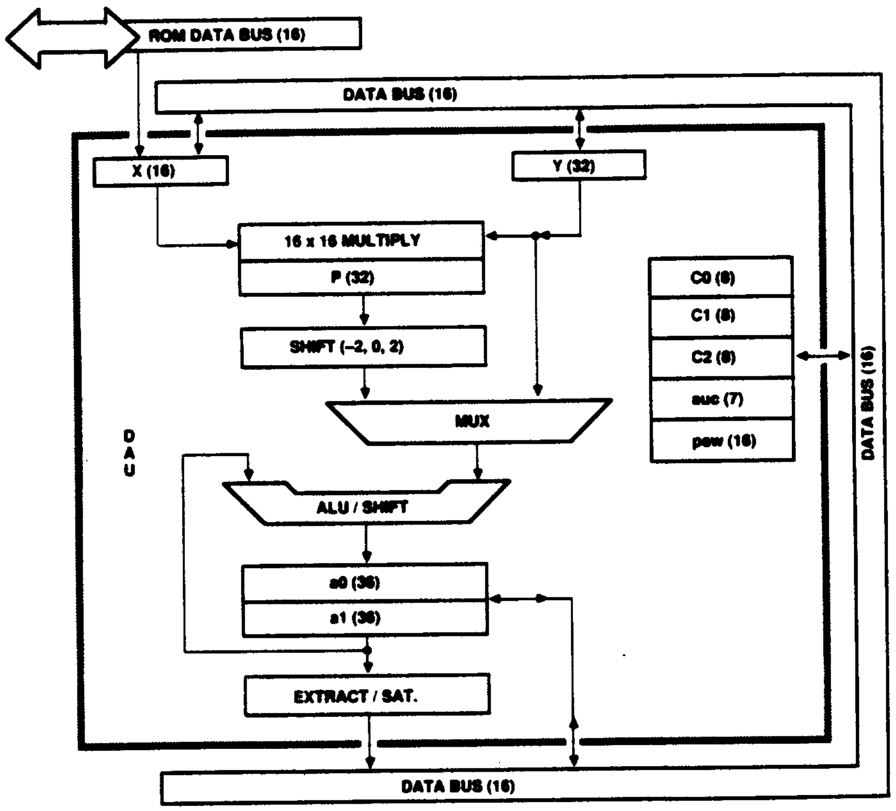

Figure 2. Block Diagram of Arithmetic Section of the DSP16A

Other features of the ADSP-2101 ALU include six status flags, division primitives and a complete set of background registers for fast context switching.

DSP16A ALU

DSP16A ALU operations require that one operand must come from an accumulator registers while the other must come from the multiplier output or from the data bus through yh or yl registers. Two-number addition is a two-step process. First, the accumulator must be loaded with the first data value. After the accumulator is loaded, a second number can then be added. Before the result can be used as an input value to the multiplier, it first must be stored back into data memory.

The ALU operates on 16- or 32-bit operands compared to 16-bit operands on the ADSP-2101. There is no carry bit or provisions for operating on larger operands. ALU result registers are 36 bits wide.

The number of cycles required to execute an ALU operation depends on the number of data transfer operations and if the instruction word is present in the cache. Generally, a single ALU operation executes in one cycle. An ALU operation with a data transfer may require two cycles.

ADSP-2101 MAC

As shown in Figure 1, the ADSP-2101 multiplier/accumulator (MAC) is separate from the ALU. All MAC operations occur in a single cycle. The unit performs both multiplications and MAC operations independent of the ALU. This is a key difference from the architecture of the DSP16A.

Like the ALU, the MAC has two X and two Y input registers, MX0, MX1 and MY0, MY1. Operations are performed on any X-Y pair of input registers. These registers can be loaded from program memory, data memory or other registers in the processor. The result of the operation appears in the multiplier result register (MR) or the multiplier feedback register (MF). As in the ALU, the feedback and result registers can also serve as the X and Y inputs to subsequent MAC operations. The barrel shifter result registers and the ALU result register can also be used directly as X inputs to the MAC.

The ADSP-2101 multiplier performs mathematical operations on 16-bit data. The data formats can be any combination of signed or unsigned values. This feature simplifies the implementation of 32-bit multiplication.

The multiplier result register (MR) is a 40-bit accumulate register. MR is divided into two 16-bit registers (MR0 and MR1) and one 8-bit extension register (MR2). DSP applications frequently deal with accumulator operations that require a large dynamic range. The extension register allows for 256 MAC overflows before a loss of data can occur.

All MAC operations can execute in a single cycle. Since two new operands can be loaded into the input registers in parallel with the computation, a new MAC operation can be executed every cycle.

Other features in the MAC include an overflow status bit, a 16-bit rounding option, and a complete bank of background registers for fast context switching.

DSP16A MAC

As shown in Figure 2, the DSP16A MAC consists of a 16x16 multiplier and the ALU described above. One multiply operand must come from the 16-bit x register. The second operand is chosen from one of the two 16-bit y registers. Because a multiply/accumulate operation requires both the ALU and the multiplier, two cycles are required to perform a single MAC operation.

There are only three registers available for MAC operands. The x register can be loaded from program or data memory. The y registers must be loaded from data memory. Two cycles are required to load the x register if the load instruction is not available in the cache. The 32-bit multiplier result is stored in the p register. To complete the MAC operation, the p value is added to an accumulator. A minimum of two cycles are required before a multiplier result can be used in another multiplication.

The DSP16A accumulate registers are 36 bits wide. A MAC operation can overflow 16 times before data is lost.

At least two clock cycles are required to execute a single MAC operation. Multiple MAC operations can be pipelined to reduce execution time. In addition, two new operands can be loaded in parallel with the MAC operation.

ADSP-2101 Barrel Shifter

The barrel shifter in the ADSP-2101 has an SI input register and can also accept inputs from any result register in the processor (e.g. MR1, AR) including its own result register, SR. Like the MAC result registers, the 32-bit SR is divided into two 16-bit registers, SR0 and SR1. The shifter also has an exponent register, SE, which is set automatically by the exponent adjust instruction and is used for normalization.

The shifter can place a 16-bit input value anywhere in a 32-bit field. The input can be shifted any number of bits from off-scale left to off-scale right. The shift can be sign-extended or zero-filled. Other functions such as exponent detection, normalization, denormalization, block floating-point exponent maintenance, and pattern merging can also be performed with the shifter. All shifter operations are performed in a single cycle regardless of the number of bits shifted.

DSP16A Shifters

The DSP16A contains two shifters. One shifter can shift the output of the multiplier two bits, left or right. This shifter is used for scaling multiplier results. The second shifter is contained in the ALU. Since the second shifter requires the same circuitry as the ALU, it has the same limitations as the ALU. The DSP16A can only shift operands located in the accumulators.

The ALU/shifter can perform a sign extended right-shift or a zero-fill left shift. The DSP16A has no provisions for block floating point, or pattern merging. Data can be shifted in a single cycle only by 1, 4, 8 or 16 bits. A 6-bit shift requires three instruction cycles.

Arithmetic Summary
ˉable 1 summarizes the comparison of the arithmetic apabilities of the ADSP-2101 and the DSP16A.

DSP Requirement	ADSP-2101	DSP16A
Single cycle ALU operations	√	√[1]
Single cycle multiplication	√	√
Single cycle MAC operations	√	No[2]
Single cycle shifting	0-32 bits	1,4, 8, or 16 bits
Accumulator overflow protection	8 bits	4 bits
Signed, unsigned or mixed-mode multiplications	√	No

1. May require 2 cycles if data transfer is required.
2. Multiple MAC operations can be pipelined to reduce execution time.

Table 1. Arithmetic Capabilities

ˀATA ADDRESSING
A digital signal processor's ability to perform fast arithmetic is wasted if the required data cannot be fetched at a similar speed. DSP algorithms require that data operands and coefficients be available at the same time. Likewise, circular buffers are a natural method for accessing tables and coefficients. The Harvard architecture allows coefficients and data to be available in both program or data memory. Simultaneous fetches of two operands is necessary to make efficient use of this architecture.

Figure 3 shows the data address generators of the ADSP-2101 while Figure 4 shows the data address generators of the DSP16A.

ADSP-2101 Addressing
There are two independent data address generators (DAGs) in the ADSP-2101. One typically supplies addresses for program memory data fetches while the other supplies addresses for data memory. The ability to simultaneously fetch two operands makes the ADSP-2101's Harvard architecture very efficient. The address generators are completely separate from the program sequencer.

Each DAG has four I registers which store pointers (addresses), four M registers for address modifiers, and four L registers which store circular buffer lengths for modulo addressing.

DAG1 can bit reverse addresses as they are output to the address bus. This zero-overhead bit-reversing is useful for implementing FFTs. The 14-bit I, M and L registers can also be used for general purpose data storage.

ADSP-2101 Indirect Addressing
With indirect addressing, the address in an I register drives either the data or program memory address bus. As shown in

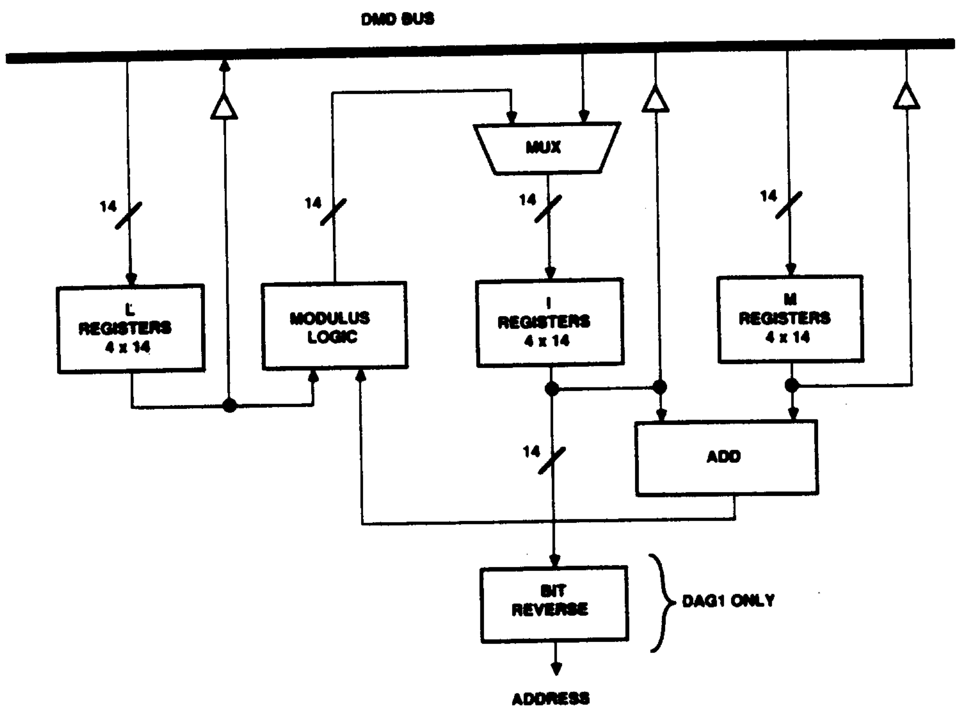

Figure 3. Block Diagram of ADSP-2101 Data Address Generators (DAGs)

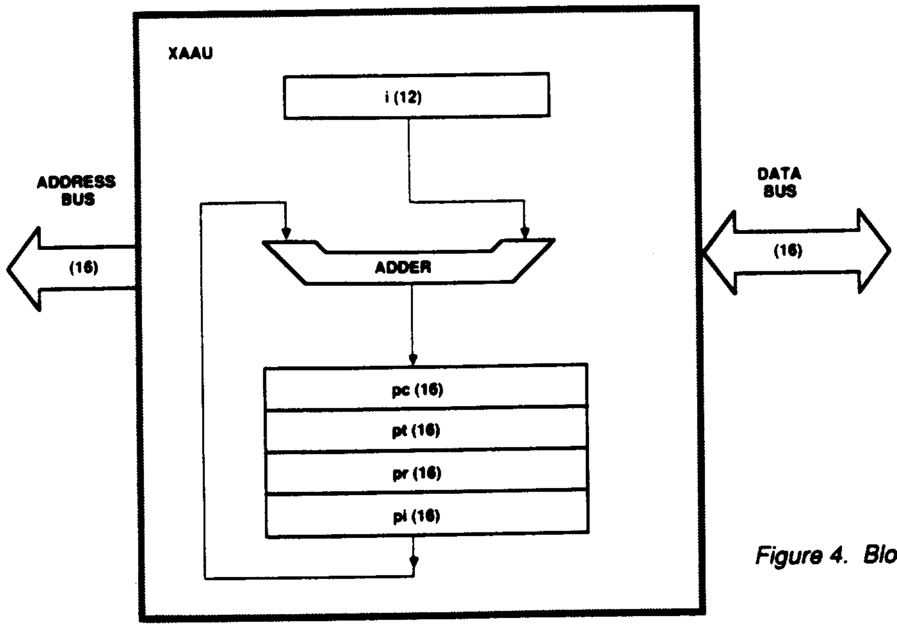

Figure 4. Block Diagrams of DSP16A Address Generators

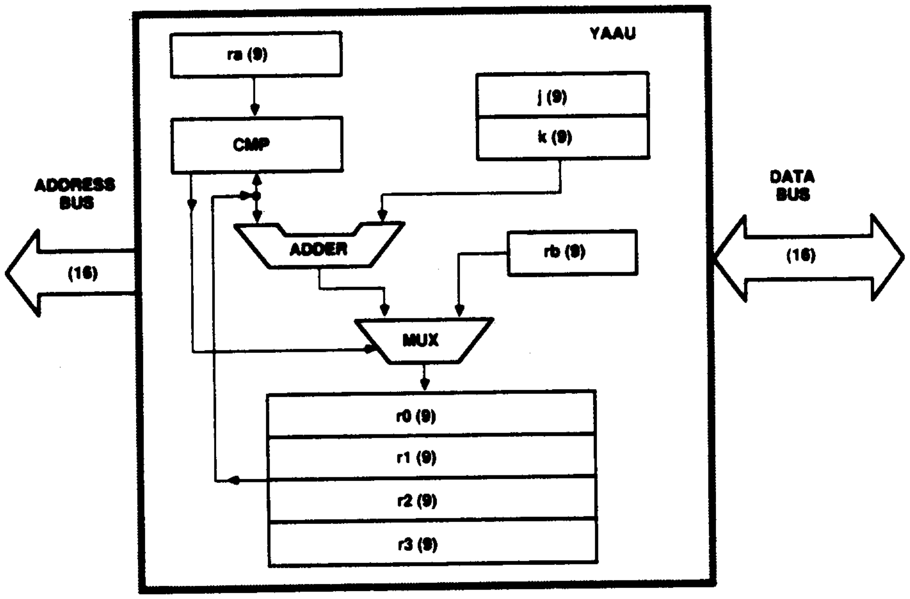

Figure 3, while memory is being accessed, The DAG updates the address simultaneously by adding to it the contents of any modify (M) register in the DAG. The specific pairing of the 14-bit I and M registers is up to the programmer. For example, I0 and M3 could be specified in the instruction as:

```
AX0=DM(I0,M3)  {Load AX0 from Data Memory}
```

The ability to mix I registers and M registers is especially useful for two-dimensional addressing or for supporting pointer increment and decrement operations without constantly loading different modify values. This instruction syntax shows explicitly what registers are used to generate the address and where the data is going; nothing is inferred.

The ADSP-2101 multifunction capability allows up to two data accesses and one arithmetic operation in a single instruction. This example instruction fetches two operands and performs an ALU operation:

```
AR=AX0+AY0(SS), AX0=DM(I2,M3), AY0=PM(M4,M5);
```

Loading the length of a circular buffer into the L register activates the modulus logic, guaranteeing that the address is kept inside the buffer in a modulo fashion. This structure is maintained automatically by the address generator hardware and does not have to be calculated explicitly by the programmer. Once initialized, circular buffers are used transparently and require no overhead instructions.

ADSP-2101 Direct Addressing

Due to the 24-bit width of the ADSP-2101 instruction, a full 14-bit address can be specified within a single-word instruction. This feature allows single cycle access to data located in data memory. There is no paging or memory segmentation on the ADSP-2101. The programmer can specify an immediate address or a predefined variable. Below are some examples of direct addressing read instructions.

```
MX0=DM(beta);

AY1=DM(0x0FE3);
```

DSP16A Addressing

Like the ADSP-2101, the DSP16A contains two address generators - the ROM address arithmetic unit (XAAU) and the RAM address arithmetic unit (YAAU). Like the ADSP-2101, the ability to fetch two operands allows efficient use of the DSP16A's Harvard architecture. Only the YAAU can implement circular buffers.

The XAAU generates addresses for program memory ROM. It acts as an instruction sequencer for the CPU and as a data address generator. It has four 16-bit static pointer registers that are used for the program counter (pc), the subroutine return address (pr), interrupt return address (pi) and table lookup (pt). The pt register is used for fetching data words from program ROM. This register is automatically post modified by +1 or by the value stored in the 12-bit i register. There is no provision for circular buffers in the XAAU.

The YAAU is dedicated to addressing data RAM. It contains four pointer registers, two address modify registers and provisions for one modulo or circular buffer. While the data address bus is 16 bits wide, the registers in the YAAU are only 9 bits wide. The DSP16A can indirectly access 64K of data memory, but only 2K bytes are available at any time.

There is no bit-reverse capability on the DSP16A.

DSP16A Indirect Addressing

The DSP16A uses indirect addressing in both the XAAU and the YAAU. The XAAU contains four 16-bit registers that can be post-modified after data is fetched from a ROM location. The XAAU lacks circular buffering capabilities. If filter coefficients are stored in program ROM, the program must reset the pt register after every filter iteration.

The YAAU contains four 9-bit pointer registers used to indirectly address data RAM. The pointers can be post-modified by -1, 0, +1, +2 or by a value stored in one of two post-modify registers.

The DSP16A has provisions for one circular buffer, limited to 2K words. Only +1 is allowed as a post-modify value. The ra and rb registers contain the last and first address of the circular buffer. If an address register is post-modified to a value equal to that in the ra register, then the value in rb is written back into the address register.

DSP16A Direct Addressing

The DSP16A can directly load a data value from memory. Below is an example of an instruction using direct addressing to read from data memory (user-defined variable, ibuf) to register zero:

```
r0=ibuf
```

Other DSP16A Addressing Modes

Most of the DSP16A internal registers can be immediately loaded from program memory. In most cases, it takes two instruction cycles and two ROM locations to implement an immediate load. Registers in the YAAU can be loaded in one instruction.

The DSP16A has a compound addressing mode. The programmer can specify a swap between an internal register and a data RAM location addressed by a YAAU register. The YAAU register can be post-modified during this operation. Compound addressing requires two instruction cycles to execute.

Address Generation Summary

Sustained intensive arithmetic operations demand maximum performance from the data addressing section of a processor architecture. Table 2 summarizes the differences between the ADSP-2101 and the DSP16A.

DSP Requirement	**ADSP-2101**	**DSP16A**
Single cycle fetch of two operands	√	√
Generate new program memory and data memory addresses each cycle	√	√[1]
Bit reverse data memory addresses for FFT	√	**No**
Transparent, no overhead circular buffers	1-8	1
Maximum circular buffer size	16K words	2K words

1. The YAAU can directly access only 2K-bytes of data memory.

Table 2. Data Addressing Capabilities

PROGRAM SEQUENCING

Efficient architectures for signal processing require fast arithmetic and matching speed in data addressing and fetching capabilities. To fully deliver the performance required for real-world signal processing, a DSP machine must execute its program with little or no overhead spent on maintaining the proper program flow.

Efficiency in program sequencing has many different aspects. The comparison in this application note focuses on three areas:

- Execution of Loops
- Execution of branches and conditional instructions
- Processing of interrupts and subroutine calls

Loops are fundamental to the implementation of DSP algorithms. Many operations, such as the sum-of-products, are repetitive. If a program is efficiently expressed in looped form then coding is quite straightforward. In addition, modifying the program (for example, increasing the number of filter taps) is easy.

Branching on conditions is a natural way to code any program which must respond to its environment. Efficient response to interrupt requests is also important.

Figure 5 shows the architecture of the ADSP-2101 program sequencer. The program control section of the DSP16A is shown in Figure 6.

ADSP-2101 Program Sequencer

The program sequencer on the ADSP-2101 contains logic that selects a program memory address source and routes the address to the program memory address bus (PMA). This address selection occurs automatically in response to the

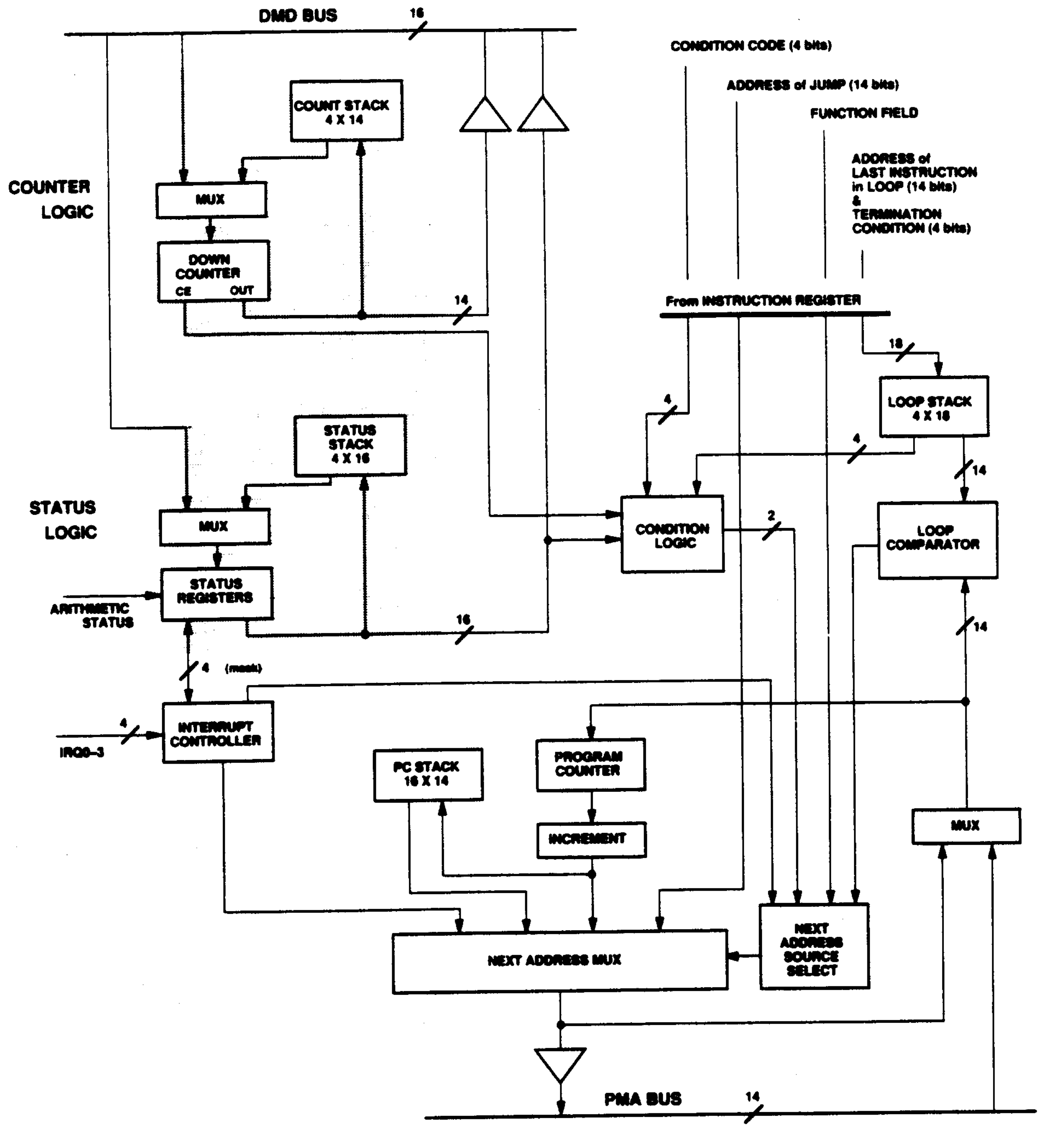

Figure 5. Block Diagram of ADSP-2101 Program Sequencer

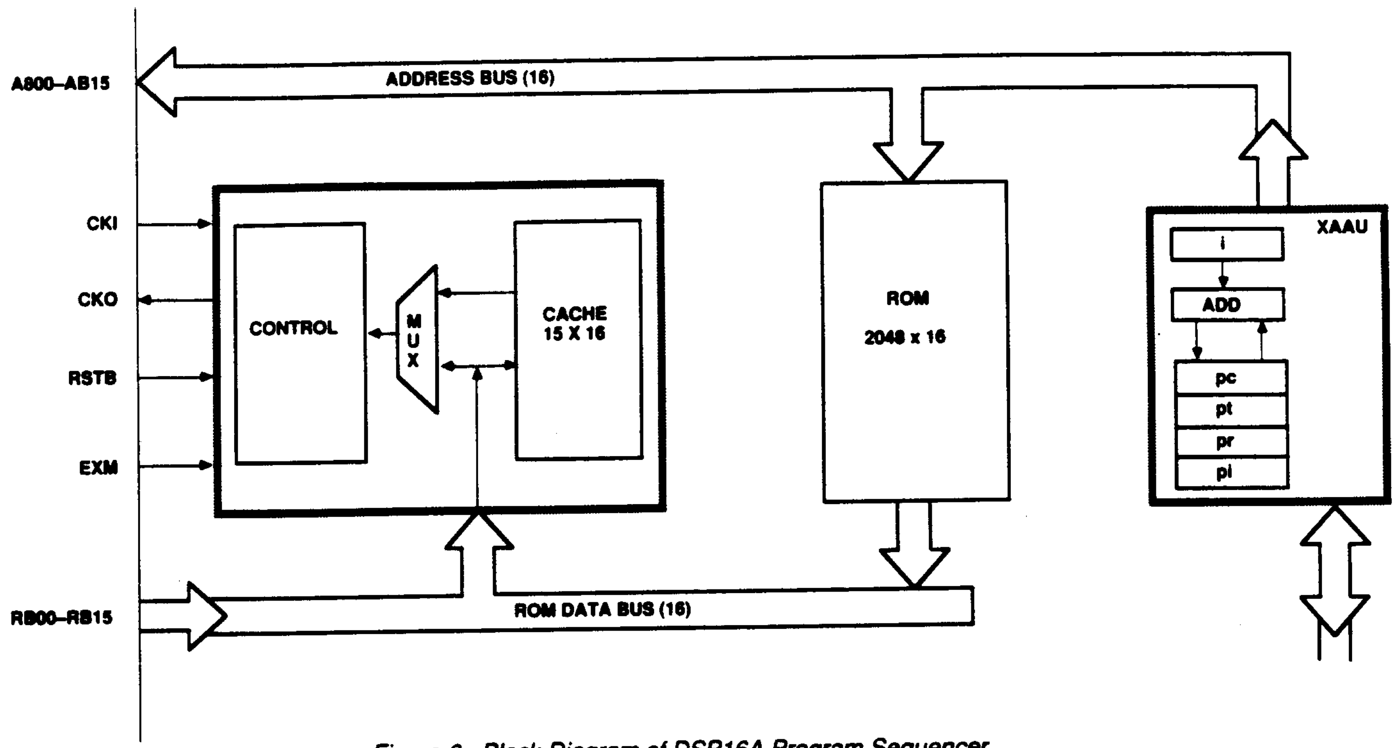

Figure 6. Block Diagram of DSP16A Program Sequencer

current instruction. The address placed on the address bus can come from

- The program counter (for sequential addressing),
- A 14-bit address in the instruction word itself, for direct jumps and subroutine calls,
- The PC stack, for returns from subroutines and interrupts,
- The interrupt logic, to automatically vector to the interrupt routine in response to assertion of any external interrupt.

All instructions can execute in a single cycle; this applies equally to jumps, calls, and interrupts. No pipelined instructions are required. Since the ADSP-2101 can fetch an instruction and access both on-chip program and on-chip data memory in one cycle, no cache is required.

When an interrupt occurs, the complete status of the processor (stack status, mode status, arithmetic status and interrupt mask) is automatically pushed onto the status stack as part of the interrupt vector process.

ADSP-2101 Looping Capabilities

The ADSP-2101 program sequencer can support zero-overhead loops. Using the loop stack and loop comparator, the processor can decide if the loop should terminate and determine the address of the next instruction (either the top of the loop or the instruction following the loop). These actions are performed in parallel with instruction execution, are transparent to the user and require no overhead cycles. Program loops can be nested up to four deep.

A DO UNTIL loop may be as large as program memory size permits. A loop may terminate when the 14-bit counter expires or when a specified arithmetic condition occurs. The example below shows a three instruction loop that is repeated 100 times:

```
CNTR = 100;
DO foo UNTIL CE;
        first loop instruction
        second loop instruction
foo:    third loop instruction
first instruction outside of loop
```

The first instruction loads the counter with 100. The DO UNTIL instruction contains the termination condition (counter expired) and the address *foo*, the last instruction in the loop. The execution of the DO UNTIL instruction causes the address of the first instruction of the loop to be pushed on the PC stack and the address of the last instruction of the loop to be pushed onto the loop stack.

As instruction addresses are output to the program memory address bus, the loop comparator determines if the instruction is the last instruction in the loop. If it is, the program sequencer checks the status and condition logic to see if the termination condition is satisfied. The program sequencer then takes either the address from the PC stack (to go back to the top of the loop) or simply increments the PC (to go to the first instruction outside the loop).

The looping mechanism of the ADSP-2101 is automatic and transparent to the user. Once the DO UNTIL instruction is specified, program flow, all stack updates, and counter updates are handled by the sequencer logic. No program overhead or extra cycles are required. After the last loop instruction is executed, the next instruction will be the first loop instruction or the first instruction following the loop.

The ADSP-2101 has up to six interrupts. Three of these interrupts are external to the processor. The interrupts are prioritized, maskable and can be edge or level sensitive. When an interrupt is asserted, the ADSP-2101 aborts execution of the current instruction and vectors to the appropriate interrupt service routine. Because the interrupts are prioritized, concurrent interrupts can be handled without external hardware.

The PC stack is 16 words deep. Up to seven nested interrupts are allowed. The size of the loop stack and the count stack allow four-deep nested loops.

Conditional execution of instructions is available on the ADSP-2101. All ALU, MAC and shifter instructions can be executed conditionally. The only exceptions are ALU division and immediate shifts. All program jumps, subroutine calls and returns can also be conditionally executed. All of these instructions execute in a single cycle.

DSP16A Program Sequencer
Program sequencing in the DSP16A is controlled by the XAAU, a 16 word cache and other control logic. The XAAU contains the program counter, an interrupt return register and a subroutine return register. The program counter can address up to 64K of program memory ROM.

The DSP16A can implement one level of program looping with the internal cache. The loop is limited to 15 bytes of instruction and cannot execute more than 127 times. An advantage of the internal cache is that instructions that require a ROM memory read execute more efficiently. However, interrupt requests are not recognized if the DSP16A is executing instructions from the cache.

The example below shows a two instruction loop that executes 100 times:

```
do 100 {
          first loop instruction
          second loop instruction
       }
```

There are two classes of instructions that can be executed conditionally: special-function instructions and branch instructions. Special-function instructions consist of data shifts, accumulator increments and some data transfer operations. These instructions can be conditionally executed based on the status of the multiplier/ALU status flags or a pseudo-random sequence bit. Special-function instructions execute in one cycle.

Branch instructions include goto, subroutine calls and return instructions. These instructions require two cycles to execute. If the instruction is conditional, three cycles are required. Since there is only one subroutine return register, extra instructions are required to implement nested subroutine calls.

There are four internal and one external interrupts on the DSP16A. The internal interrupts are asserted by the parallel and serial interface ports. In addition, there is one software interrupt available. All interrupts are maskable. When an interrupt is recognized, the DSP16A completes the current instruction and vectors to the interrupt service routine. Note that the processor supports only one service routine. The service routine must determine which interrupt was processed. The DSP16A does not recognize interrupts when executing branch instructions or instructions in the cache. Interrupts are not prioritized. Providing for concurrent interrupts requires extra hardware and/or programming.

Program Sequencer Summary
Efficient looping capabilities are very import for DSP algorithms due to their repetitive nature. Table 3 summarizes the program sequencing capabilities of the ADSP-2101 and the DSP16A.

DSP Requirement	ADSP-2101	DSP16A
Zero-overhead looping	√[1]	√[2]
Conditional arithmetic instructions	√	√[3]
Single-cycle branching	√	No
Zero-overhead nested interrupts	Up to 7 deep	No

1. Four levels of nested loops.
2. One level of looping; up to 15 instructions long; can execute only 127 times.
3. Shifts and accumulator increments only.

Table 3. Program Sequencing Capabilities

PROGRAMMING EXAMPLE
A good illustration of the differences between the ADSP-2101 and the DSP16A is the implementation of the stochastic gradient algorithm for updating taps in an adaptive filter.

$$C_{n+1}=b*e*Y_n$$

A pseudo-code description of this algorithm is shown for a 256 tap filter.

```
MOD=beta*e;
Loop n=0 to 255;
   $C_{n+1}=C_n$ MOD*$Y_n$;
EndLoop;
```

When implementing this algorithm you should consider the following:

How many cycles are required for loop overhead?

How many cycles are required for loading and storing operands?

How many extra cycles are required to move intermediate results?

Listing 1 shows an ADSP-2101 code segment for updating the taps in a 256-tap adaptive filter. The coefficients are stored in program data memory and the sample data is stored in data memory. Efficient operand manipulation is achieved by storing the MOD term in the multiplier feedback register. In the loop, data is fetched in parallel with instruction execution. No cycles are wasted fetching or storing data.

Listing 2 shows a DSP16A code segment for updating the taps in a 256-tap adaptive filter. Although data operands can be fetched from program memory ROM, is not possible to write updated values back to program ROM. Therefore, both coefficients and data must be stored in data memory. Since the multiplier output register is not available as an input for another multiplication, two extra instruction cycles are required to move the MOD term back into the x register. The program loop contains two instructions. Three clock cycles are required to execute this loop.

Interrupt requests are ignored when the DSP16A is executing from a cache loop. The code segment shown below periodically stops the loop so pending interrupt requests can be serviced.

The total number of clock cycles for each algorithm are:

	N Taps	*256 Taps*	*Time*
DSP16A	8N + 25	2073	69 µs (33 ns cycle)
ADSP-2101	3N + 15	783	63 µs (80 ns cycle)

CONNECTING THE PROCESSOR TO OTHER DEVICES

Although digital signal processors that have efficient architectures, fast arithmetic capabilities and efficient data addressing are well suited for high performance systems, a DSP system designer must also consider how the microprocessor communicates with other components in the system. These devices can include other DSP processor, a host processor, or peripherals such as CODECs, A/D converters and D/A converters. The configuration of off-chip memory and the size of on-chip memory can significantly affect the cost of a DSP system.

Connecting the ADSP-2101 to Other Devices

The ADSP-2101 has two bidirectional double-buffered serial ports. These ports can be used to communicate with serial devices such as CODECs, serial A/D converters and serial ports on other ADSP-2101s. The ADSP-2101 can communicate with parallel devices by memory-mapping them into external data memory space.

The serial ports (SPORTs) are synchronous and use framing signals to control data flow. Each SPORT can generate its own serial clock internally or use an external clock. The framing sync signals may be generated internally or by an external device. Word lengths may vary from three to sixteen bits. The serial ports can have a zero chip interface to other serial devices and can perform single cycle µ-law and A-law companding in hardware.

SPORT0 has a multichannel capability which allows the receiving or transmitting of arbitrary data words from a 24-word or 32-word bitstream. This is useful for implementing a T1 interface or a networking scheme for multiple processors.

SPORT1 may be optionally configured as two external interrupts, $\overline{\text{IRQ0}}$ and $\overline{\text{IRQ1}}$, and the Flag In and Flag Out signals instead of as a serial port.

The SPORTs also have an autobuffering capability. This feature allows a block of data to be loaded into memory while the ADSP-2101 is executing program code. When all data is loaded, the SPORT interrupts the processor.

The ADSP-2101 can interface to parallel devices such as parallel DACs and UARTs through external data memory. Software-controlled wait states in the ADSP-2101 allow for a simple interface to these devices. The only additional circuitry that may be required is address decode logic.

Connecting the DSP16A to Other Devices

The DSP16A has two ports for communicating with the other devices. The SIO is a serial communications port. The PIO is designed for parallel communications. Either the SIO or the PIO is required to load internal RAM from off-chip devices, such as external memory or a host processor.

The SIO is a synchronous port. The framing signals are ILD and OLD and must be synchronized to the input and output clocks respectively (ICK and OCK). Framing and clock signals can be generated internally or externally. The internal clock is programmable to one of four different clock speeds. Word lengths are limited to 8 or 16 bits. A time division multiplex (TDM) communications scheme can be used to communicate with up to seven other DSP16As.

The PIO interfaces to AT&T CODECs without any additional circuitry. A bit-reversing capability on the PIO allows for an easier software implementation of companding operations.

The DSP16A PIO is a parallel port that can be programmed to transmit or receive 16-bit data words or simultaneously transmit and receive 8-bit data words. The PIO can operate in either the active or the passive modes. In the passive mode, an external device provides the I/O strobe signals. In active mode, the strobe signals are provided by the DSP16A. The width of the strobe signals is programmable allowing connections to slow peripherals.

Memory Configuration on the ADSP-2101

The ADSP-2101 contains 2K words of on-chip program memory and 1K words of on-chip data memory. In a single cycle, the ADSP-2101 can access on-chip program RAM twice and on-chip data RAM once. All instructions execute in a single cycle when the processor is executing from on-chip memory.

The ADSP-2101 can address 16K words of program memory and up to 16K words of data memory. An additional 16K words of data memory are available with the PMDA pin. The processor can be configured with all memory off-chip or with a

```
        {                     DAG Initialization                      }
        {                                                             }
        {I2->Oldest input data value in delay line}
        {L2=Filter Length}
        {I6->Beginning of filter coefficient table}
        {L6=Filter Length}
        {M1,M5=1}
        {M6=2}
        {M3,M7=-1}

.ENTRY affir;
strt: CNTR=< Filter Length >
        MX0=< error >
        MY1=beta;
        MF=MX0*MY1(RND), MX0=DM(I2,M1);        {MF=error*beta}
        MR=MX0*MF(RND), AY0=PM(I6,M5);

        DO adapt UNTIL CE;
              AR=MR1+AY0, MX0=DM(I2,M1), AY0=PM(I6,M7);
adapt:  PM(I6,M6)=AR, MR=MX0*MF(RND);

        MODIFY(I2,M3);                         {Point to oldest data}
        MODIFY(I6,M7);                         {Point to start of table}
```

Listing 1. ADSP-2101 Code Segment for Stochastic Gradient Algorithm

```
      j=0
      k=1
      r0=beta                                             /*Load beta and e*/
      r1=e
                                                x=*r0
                                                y=*r1
                  p=x*y                                   /*Calculate MOD*/
      a0=p
      x=a0                                                /*Store MOD in x*/
      r0=<address of coefficients Cn>                     /*Init pointers*/
      r1=<address of data An>
                                                a0=*r0
                                                y=*r1++
                  p=x*y
  DO 2 {                                                  /*Program loop */
      a0=a0+p                             y=*r1++
                  p=x*y                   *r0jk:a0
      }
  REDO 127                                                /*Service Interrupts*/
```

Listing 2. DSP16A Code Segment for LMS Stochastic Gradient Algorithm

portion of memory on-chip. For off-chip memory, the program and data memory lines are multiplexed into a single off-chip address bus and a single off-chip data bus. Extra cycles may be required to execute instructions when more than one piece data is located off-chip.

The boot capability of the ADSP-2101 allows loading internal program memory directly from an inexpensive boot EPROM. On reset, up to 2K words of instructions can automatically be loaded into on-chip program memory. Up to seven more 2K instruction blocks can be paged into the processor on demand. If desired, the boot code can contain a short program for downloading data and program code through the serial ports.

Software programmable wait states on the ADSP-2101 makes it compatible with slow external RAM or ROM. Using slower memories reduces overall system cost at the expense of reduced execution speeds. Regardless of the speed of external memory, the ADSP-2101 always executes at full speed when accessing on-chip program and data memories.

Memory Configuration on the DSP16A
The DSP16A contains 2K words of on-chip data memory and 4K words of on-chip program memory ROM. In one cycle, the DSP16A can access data RAM and instructions in the program memory ROM. A second cycle is required to fetch data from program ROM.

The DSP16A can address up to 64K of program ROM. Like the ADSP-2101, the part can be configured to use off-chip memory only or a combination of on-chip and off-chip program memory. Since there is no off-chip data bus, data memory cannot be expanded off-chip. The data memory address space can be expanded through the PIO port. However, this requires external circuitry and additional software.

Because the DSP16A is designed to execute ROM code, there are no read/write strobes associated with the program memory address and data lines. This makes it difficult to use RAM for program memory. Extra hardware is required to download code from a host processor.

The internal ROM on the DSP16A must be programmed by the factory. The extra costs associated with programming internal ROM make it impractical for low-volume applications. It is also expensive to fix bugs or upgrade code once the device is programmed. In applications with external program memories, the high clock speeds of currently available DSP16As require extremely fast external EPROMs. The 33 ns version requires EPROMs with 15 ns access times. A system that uses fast versions of the DSP16A will be very expensive to design and manufacture. There are no programmable wait states on the DSP16A.

SUMMARY
Digital signal processing is a specialized branching of processor design and application. Table 4 summarizes the fundamental requirements of DSP.

The DSP processors available today vary drastically in their ability to meet the requirements described above. Some processors are optimized for very specific applications while others are flexible and allow for future growth. Analyzing the requirements of your DSP system and matching them to the capabilities of a DSP architecture assures efficient operation.

Due to space limits, this application note does not cover many topics in detail. Consult the *ADSP-2101 User's Manual* and the *ADSP-2101 Cross-Software Manual* for a greater depth of information on this processor.

DSP Requirement	**ADSP-2101**	**DSP16A**
Fast arithmetic	√	√
Extended dynamic range on multiplication/accumulation	√[1]	√[2]
Hardware circular buffering (both on- and off-chip)	√	**No**
Zero overhead looping & branching	√	**No**

1. Accumulator may overflow 256 times.
2. Accumulator may overflow 16 times.

Table 4. Fundamental Requirements of DSP