ANALOG DEVICES

AN-231
APPLICATION NOTE

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

Considerations for Selecting a DSP Processor (ADSP-2111 vs. DSP56166)

by Noam Levine

INTRODUCTION

Digital signal processing applications require high performance computational hardware capable of performing intense mathematical operations in a short amount of time. The performance of a DSP system can be measured as to how well it performs in the following areas:

- Fast and flexible arithmetic
- Dynamic range on multiply/accumulate operations
- Efficient operand fetches (two operands in one instruction cycle)
- Circular buffering capabilities
- Zero overhead program looping and branching

In addition, the DSP processor should be capable of interfacing easily with external devices, be able to access large amounts of memory quickly, be easy to program, and have the support of powerful development tools to ease system debug.

This application note examines the performance of two leading DSP processors, the ADSP-2111 from Analog Devices and Motorola's DSP56000, as to how they meet these requirements.

ARITHMETIC CAPABILITIES

The basis of a good DSP processor is its ability to perform a wide variety of arithmetic and logical operations in a timely manner. The computations should be handled flexibly such that the implementation preserves the order of operations and operands. Too specific an architecture could require rearrangement of an algorithm to fit a given architecture, thereby increasing programming complexity, time, and the possibility of error.

ADSP-2111 Arithmetic Architecture Overview

Figure 1 shows a block diagram of the ADSP-2111 arithmetic section. The ADSP-2111's arithmetic section consists of three independent, parallel-connected computational blocks—an arithmetic logic unit (ALU), multiplier / accumulator (MAC) with 40-bit accumulator, and general purpose barrel shifter. Each computational unit has its own input and output registers. All arithmetic registers can also be used as general purpose data registers. A complete set of background registers is also provided, allowing a single-cycle context switch of all arithmetic registers. The three computational units can also pass information to each other via the Result Bus, which

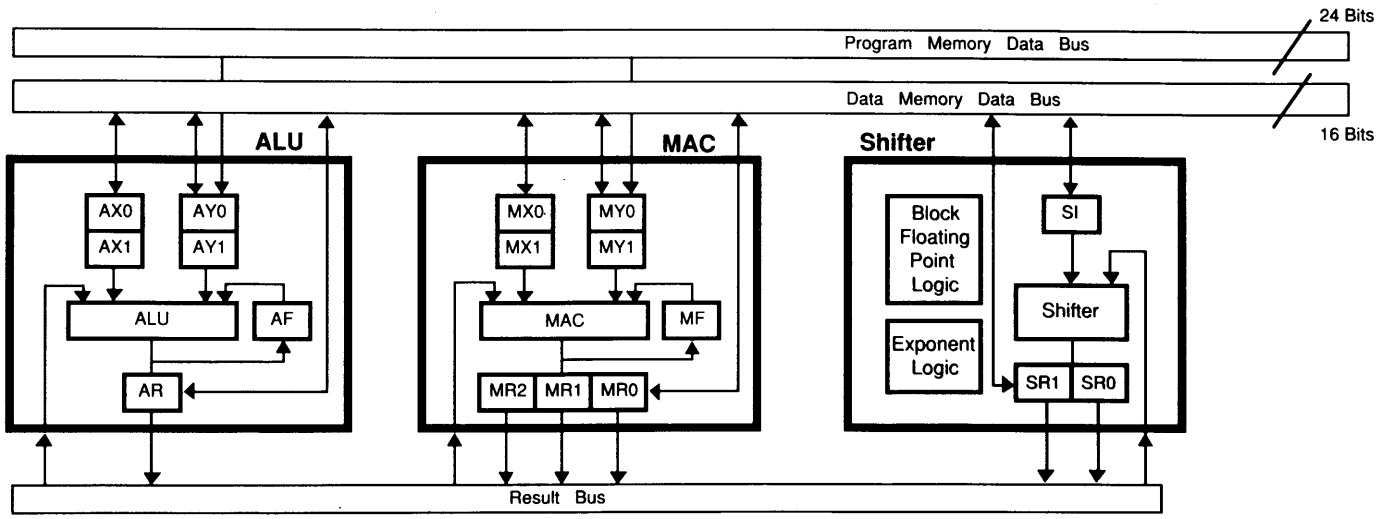

Figure 1. ADSP-2111 Computational Units

eliminates the need for extra data move instructions as any computational unit output can be used directly as an input to any computational unit. ALU and MAC inputs can come from both program and data memory, or any other data register in the processor.

DSP56000 Arithmetic Architecture Overview

Figure 2 shows a block diagram of the DSP56000 Data ALU. The Data ALU handles all of the computational operations within the processor. The Data ALU contains four 24-bit input registers, a multiply-accumulator/logic unit (MAC), two 48-bit accumulator registers, two 8-bit accumulator extension registers, an accumulator shifter (implemented in the data path prior to the MAC), and two data bus shifter/limiter circuits. Inputs for all Data ALU operations come from the four input registers. Computational results are stored in either the A or B accumulators.

Because all DSP56000 computational operations share common input and output registers, extra instructions may be required to handle temporary data storage for intermediate results should an algorithm require multiple interleaved MAC and Arithmetic operations. This would increase both the execution time and program and data memory size requirements of the algorithm. In addition, the lack of a general purpose shifter requires the use of multiple instruction cycles to execute data shifts greater than one bit.

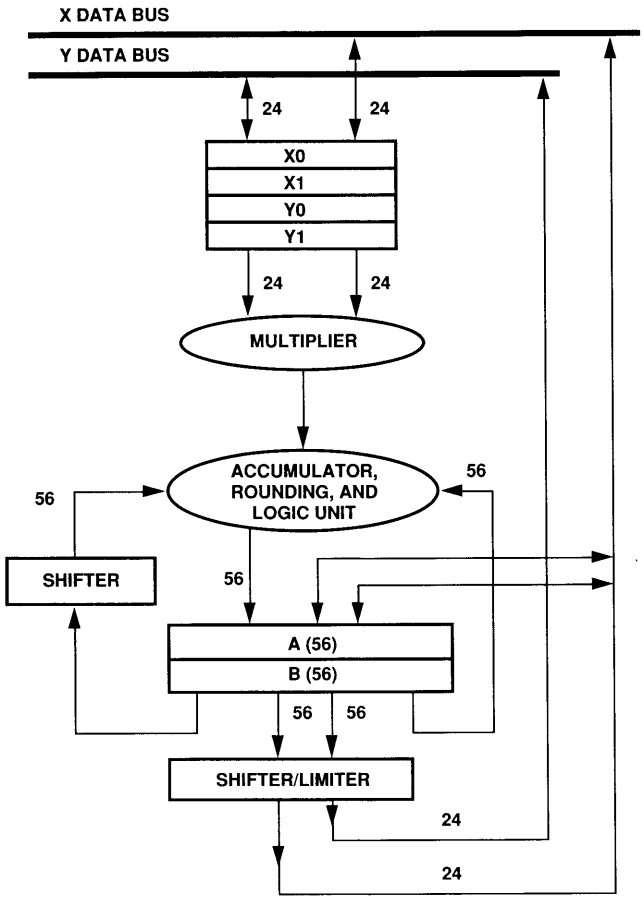

Figure 2. DSP56000 Data ALU

ADSP-2111 ALU

The ADSP-2111 ALU has two X and two Y input registers—AX0, AX1, and AY0, AY1. ALU operations are performed on any X-Y combination of these input registers. The input registers may be loaded from any combination of data and program memory, or from other data registers in the processor. ALU results appear in either the ALU result (AR) or ALU feedback (AF) register. AR and AF can also be used, respectively, as X and Y operands. In addition, MAC and shifter result registers can be used directly as X inputs to the ALU.

The ALU performs mathematical operations on 16-bit data operands. An internal carry bit is always updated during mathematical operations. Addition with carry and subtraction with borrow instructions are provided for implementing multiprecision arithmetic. The ADSP-2111 ALU supports all logic operations as well as a divide primitive and an absolute value function.

Two operands can be fetched or stored in parallel with an ALU operation. This allows for the processor to perform one ALU operation per cycle without speed penalties incurred by operand fetching. All ALU operations execute in a single cycle. ALU operations are coded in a register transfer, algebraic syntax; and all ALU operations can be performed conditionally with no cycle penalty.

Other features of the ADSP-2111 ALU include mode-selectable AR saturation to protect against overflows and six status flags to monitor the condition of the ALU result.

DSP56000 Arithmetic And Logical Operation

Arithmetic and logical operations on the DSP56000 are handled through the accumulator, rounding, and logic unit. For arithmetic instructions, the unit accepts up to three input operands and outputs one 56-bit result in the form *extension:most significant product:least significant product* (EXT:MSP:LSP). When a 56-bit result is to be stored as a 24-bit operand, the LSP can either be truncated or rounded into the MSP. In order to protect against overflows, accumulation registers A and B each have separate limiting circuitry. Arithmetic operations are coded using a mnemonic syntax. No provisions are made for conditional arithmetic, so cycle time is lost on conditional branching.

Logical operations are handled through the logic unit. This unit is 24 bits wide and operates on data in the MSP portion of the accumulator.

ADSP-2111 MAC

The ADSP-2111 multiplier/accumulator (MAC) is separate from the ALU. Like the ALU, it has two X and two Y input registers, MX0, MX1, and MY0, MY1, loadable from either data memory or program memory. Like the ALU, the programmer is free to choose any X-Y input pair. The unit performs both multiplications and multiply-accumulates independently of the ALU, on any X-Y assortment of MAC input registers. Results can be directed to either the MAC result (MR) register or the MAC feedback (MF) register. These registers can also be

used, respectively, as X and Y inputs. ALU and Barrel Shifter result registers can also be used as MAC X-inputs. All MAC operations occur in a single cycle.

The ADSP-2111 performs its mathematical operations on 16-bit data. The data formats of the operands can be any combination of signed or unsigned values, a feature which simplifies the implementation of multiprecision multiplication. The 32-bit multiplier output feeds a 40-bit dedicated add/subtract array, which in turn feeds the Multiplier Result (MR) register set.

The MR register is divided into two 16-bit registers (MR0 and MR1) and an 8-bit extension register (MR2). Because DSP applications frequently deal with values over a wide dynamic range, the extension register is used to allow for 255 full-scale multiply-accumulates before a loss of data can occur. All MAC operations can be performed conditionally with no cycle penalty.

Other MAC features include an overflow status bit, a 16-bit unbiased rounding option, single-cycle conditional MR saturation, as well as the ability to clear all 40 accumulator bits in a single cycle.

DSP56000 MAC Operation

Multiply / accumulate (MAC) operations are performed using the multiplier and accumulator portions of the DSP56000 arithmetic unit. Multiplier inputs can only come from the X or Y registers. The multiplier executes 24-bit x 24-bit parallel two's-complement fractional multiplies. The 48-bit product is right justified and added to the contents of either the A or B accumulator. An 8-bit adder is used as an extension accumulator for the MAC array. Its output is the EXT: portion of the EXP:MSP:LSP format result.

If a 24-bit result is required, the DSP56000 MAC can either truncate the LSP portion of the result or round the result into the MSP.

ADSP-2111 Shifter

Provided as a separate computational unit, the ADSP-2111 barrel shifter can place a 16-bit input value anywhere in a 32-bit output field, from off-scale left to off-scale right, in a single cycle.

The barrel shifter has one input register, SI, but can also accept any computational unit result register as input. The 32-bit Shifter Result (SR) register is divided into two 16-bit sections, SR0 and SR1. The shifter also has an exponent register, SE, which can be loaded either directly from another register or automatically set as a result of exponent detection operation.

In addition to arithmetic (sign extension) and logical (zero fill) shift operations, the barrel shifter also performs exponent detection, normalization, denormalization, block-floating point exponent maintenance, as well as pattern merging. The exponent detection feature provides the user with an efficient means of performing operations using two-word floating-point data formats, utilizing the full dynamic range of a 16-bit mantissa and 8-bit exponent. The block floating point (block exponent detection) allows the normalization of an arbitrary-length frame of data to be normalized to a single exponent, thus simplifying floating point implementations.

All shifter instructions are executed in a single cycle, regardless of the number of bits shifted. All shifter operations can be conditional with no cycle penalty. As with the ALU and MAC, there is a complete set of background registers for the Shifter.

An example ADSP-2111 Shifter instruction is shown below.

```
SR = ASHIFT SI BY -17
```

This instruction performs an arithmetic (sign extended) shift on the contents of the Shifter Input register of 17 bits to the right [multiply by 2^(–17)], and places the result in the Shifter Result register. This instruction executes in a single cycle.

DSP56000 Shifter

The DSP56000 has no general purpose shifter. The accumulator shifter, implemented in the data path prior to the MAC accumulator, is used to shift accumulator input data one bit either to the left or right, pass data through, or force the accumulator to zero. The data shifter/limiters can shift, at most, one bit at a time. This means that shifting a value 32 bits would require 32 instruction cycles.

To implement the 17-bit arithmetic shift shown in the previous section, the following DSP56000 instructions would need to be executed.

```
MOVE    X0, A1      ; load accumulator from input
                    register
ASR     A           ; arithmetically shift
                    accumulator 1 bit to right
ASR     A
ASR     A
ASR     A
ASR     A
ASR     A
ASR     A
ASR     A
ASR     A
ASR     A
ASR     A
ASR     A
ASR     A
ASR     A
ASR     A
ASR     A
ASR     A
```

This operation takes the DSP56000 18 cycles to execute and uses 18 instruction words. To save code space, the shift instructions could be placed in a DO loop, requiring only 4 instruction words, but 21 instruction cycles to execute. Compare that to the ADSP-2111's single-cycle, single-instruction execution.

The DSP56000 has no exponent detection circuitry, so two-word floating-point mathematics is difficult to implement.

Computational Summary

Table 1 summarizes the comparison of computational capabilities between the ADSP-2111 and the DSP56000. While the DSP56000 provides for high dynamic range multiply accumulates, the lack of a general purpose shifter and the availability of only four input registers shared by the entire computational unit can make general purpose mathematic operations somewhat cumbersome. Algorithms may have to be altered if additions, MACs, and shifts are interleaved, as all functions share common input and output registers. There is also no provision made for background registers, so that during interrupt processing, time must be spent explicitly saving the contents of the data registers. In addition, arithmetic operations cannot be performed conditionally, thereby requiring additional cycles to handle conditional program branching.

The ADSP-2111 was designed to provide maximum flexibility for arithmetic operations. By having three separate computational units with their own input and output registers, and allowing any computational result to be used directly as a computational input, the processor can be used to implement a wide variety of algorithms without significant rearrangement of the algorithm. A general purpose shifter with built-in exponent detection logic further adds to this flexibility by allowing the programmer to implement arbitrary data scaling and two-word floating-point operations easily. A complete set of data background registers speeds up interrupt processing by allowing a complete single-cycle context switch of all computational data registers. Furthermore, all arithmetic operations can be conditional on any processor condition without cycle penalties.

DATA ADDRESSING

A digital signal processor's ability to perform fast arithmetic is wasted if the processor cannot fetch the required data fast enough to keep pace with the calculations. Many DSP algorithms require that both data operands and coefficients be made available simultaneously. Likewise, circular buffering (modulo addressing) is a natural method for accessing tables efficiently. The architecture of the DSP processor must be able to support these features easily and efficiently.

Figure 3 shows the data address generators of the ADSP-2111. The data address generators of the DSP56000 are shown in Figure 4, which can be found on page 6.

ADSP-2111 Addressing

The ADSP-2111 is based on a modified Harvard architecture. The Harvard architecture means that there are separate data and program memory spaces modified to allow the storage of both opcodes and data in program memory.

There are two independent data address generators (DAGs) in the ADSP-2111. One typically supplies addresses for data memory fetches while the other can supply addresses for data and program memory access. This configuration allows the ADSP-2111's modified Harvard architecture to fetch two data operands simultaneously—one data value from data memory and one data value from program memory. The DAGs are completely separate from the program sequencer, and only address data locations, not opcodes.

Each DAG has four index (I) registers which store memory pointers (addresses), four address modify (M) registers, and four length (L) registers which store circular buffer lengths for modulo addressing. The 14-bit I, M, and L registers can also be used for general purpose data storage.

DAG1 can bit reverse addresses as they are output to the output bus. This zero overhead bit reversing is useful in FFT implementations.

ADSP-2111 Indirect Addressing

With indirect addressing, the address in an I register drives either the data or program memory address bus. While the memory is being accessed, the address is simultaneously updated according to the contents of the specified M register. The specific pairings of I and M registers is up to the programmer. The ability to mix I and M registers is especially useful for two-dimensional addressing or for supporting

DSP Requirement	ADSP-2111	DSP56000
Computational input registers	4 ALU 4 MAC 1 Shifter	4 total
Background register set	✓	No
Single-cycle shifting	0-32 bits left or right	1 bit left or right
Conditional execution of any computation	✓	No

Table 1. Summary of Computational Capabilities

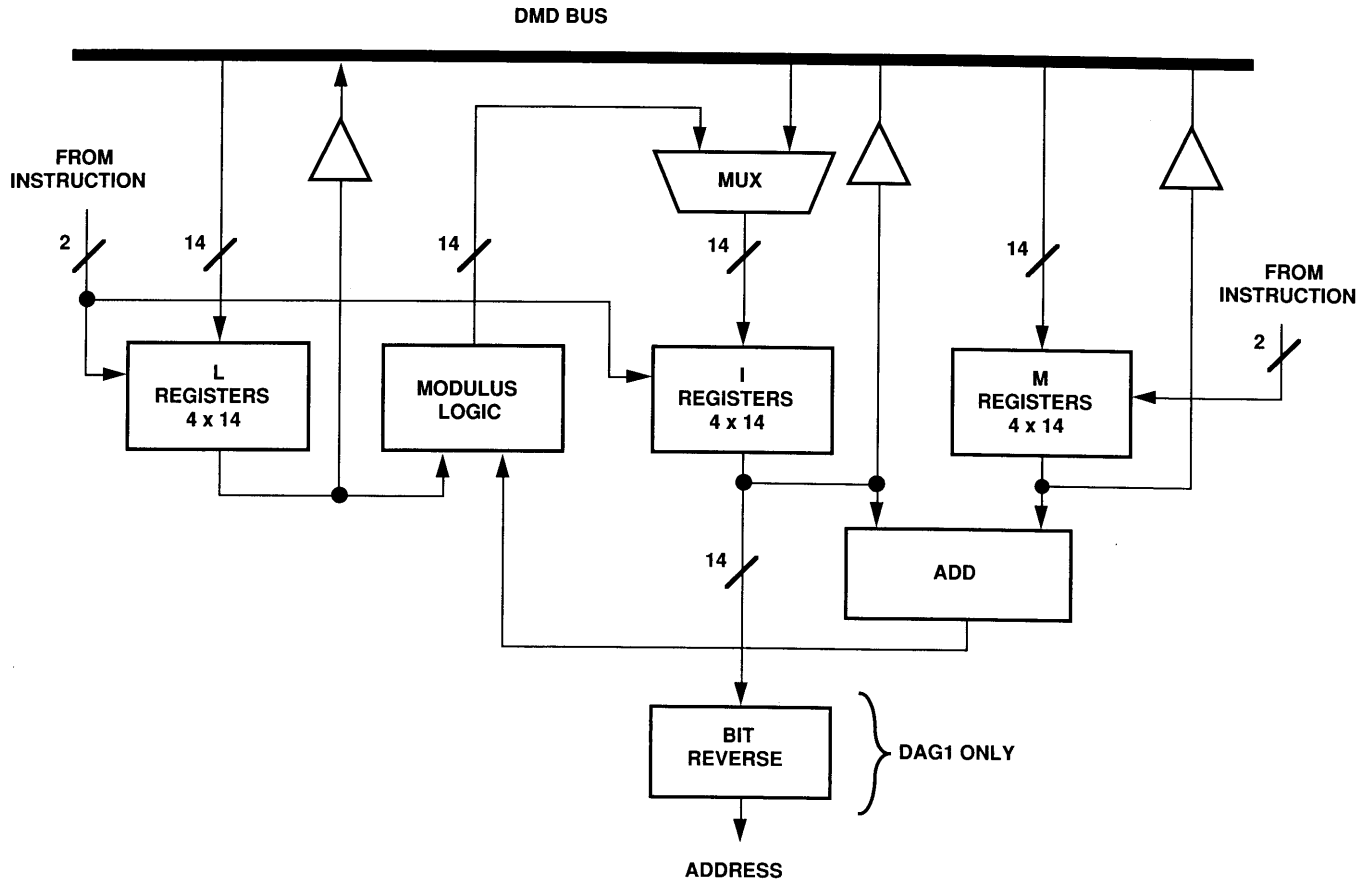

Figure 3. ADSP-2111 Data Address Generator

pointer increment and decrement without continuously loading new modify values. The instruction syntax explicitly shows which register is being used to generate the address and where the data is going; nothing has to be inferred. For example, the assembly language statement

```
DM(I1,M3) = AX0
```

means that the value in register AX0 is loaded into data memory at the address contained in I1. I1 is then updated according to the value contained in register M3. M registers can take values from – 8192 to 8191.

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 operate transparently and require no overhead instructions.

ADSP-2111 Direct Addressing

With the 24-bit width of the ADSP-2111 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. The programmer can specify an immediate address or a predefined label variable. Below are some examples of direct addressing memory read instructions.

```
AY1=DM(0x0FE3); {Absolute address specified}

MX0=DM(beta); {The label "beta" is a
              predefined memory buffer label}
```

DSP56000 Addressing

The Address Generation Unit (AGU) of the DSP56000 performs effective address calculations necessary to address data values stored in memory. The AGU is split into identical halves, each containing an address arithmetic logic unit and four sets of three registers—address (R), offset (N), and modifier (M).

The two AGUs can generate two addresses every instruction cycle; one for any two of the XAB, YAB, or PAB memory spaces. Facilities for modulo addressing and address bit reversal are also provided.

DSP56000 Indirect Addressing

Indirect addressing in the DSP56000 is handled through the use of R:N:M register triplets. Register use is restricted in that the registers must be used as triplets, e.g., only N2 and M2 can be used to update R2. No arbitrary R:N pairs can be used. Indirect addressing operations are broken down into a number of modes—no update, postincrement by 1, postdecrement by 1, postincrement by offset N, postdecrement by offset N, indexed by offset N and predecrement by 1.

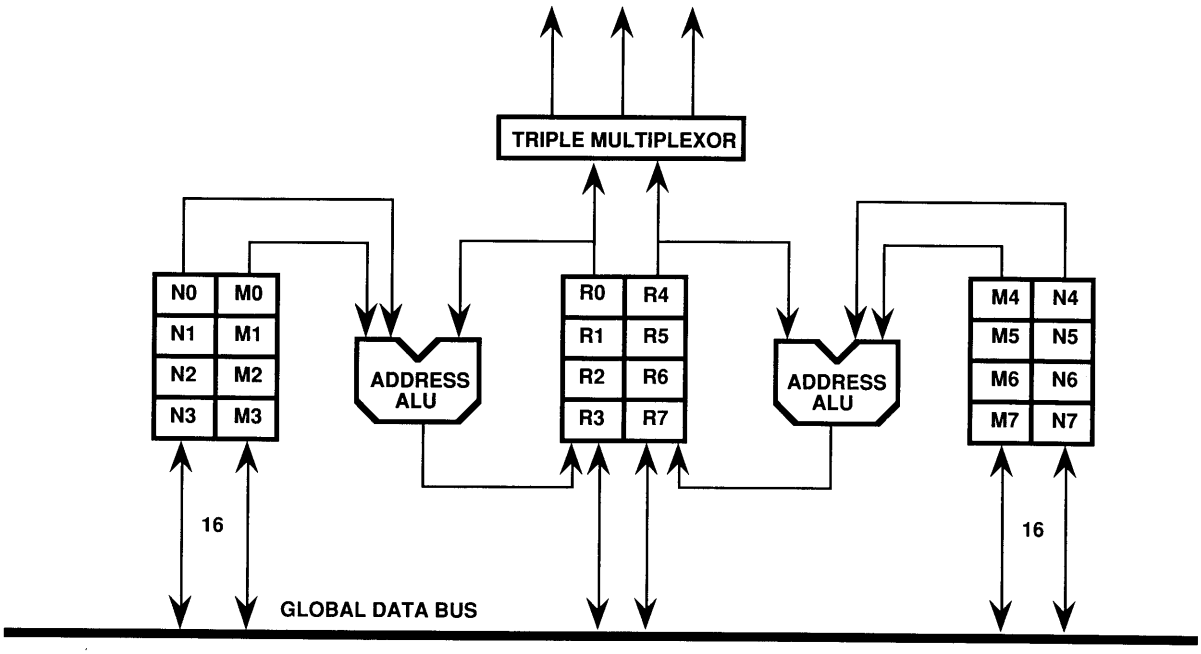

Figure 4. DSP56000 Address Generation Unit

Modulo addressing occurs when a value other than 0xFFFF is loaded into an M register. Address modification is then performed modulo M, with the address pointer remaining in the specified buffer space. Address bit reversing or reverse carry modification can be achieved by setting the appropriate M register to 0x0000.

If any addressing register (R,M,N) is modified, the new value **cannot** be used on the next cycle. A one-cycle delay is introduced due to instruction pipelining. In the ADSP-2111, any value written to any register is available for use on the very next cycle.

DSP56000 Direct Addressing

Direct addressing of memory locations in the DSP56000 can take several forms, some with cycle-time penalties. Moving data from a specific address in memory (for example, MOVE Y:$5432,B0, known as absolute addressing) takes two instruction cycles to execute. If memory between the addresses $0000 to $003F is to be accessed (absolute short addressing), then no penalty is incurred. The speed of memory access is therefore gated by the location of the data. In the ADSP-2111, any location in memory is accessible in a single cycle

Address Generation Summary

Table 2 summarizes the differences between the addressing capabilities of the two processors. Both the ADSP-2111 and DSP56000 have two independent addressing components. Both processors are able to access two operands in a single cycle and both processors support modulo (circular) buffering. The key differences between the two processors is that, depending on addressing mode, the DSP56000 may need multiple cycles to access data, and due to instruction pipelining, any addressing register that is modified cannot be used on the next cycle, thereby reducing the flexibility of the processor for memory access. The ADSP-2111 has single-cycle data access anywhere in its memory space, and any addressing changes are available on the very next cycle.

PROGRAM SEQUENCING

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

Determining the efficiency of program sequencing encompasses many areas of processor control. This comparison 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 sum-of-products, are very repetitive. If a program can be expressed efficiently in looped form, then coding is quite straightforward. Modifying a looped program to change the number of taps in a filter requires very little work.

Quick conditional branching is another natural way to code programs which must respond to its environment. Efficient interrupt servicing is also very important to fast program execution.

Figure 5, located on page 8, shows the architecture of the ADSP-2111 program sequencer. The program control section of the DSP56000 is shown in Figure 6 which can be found on page 9.

DSP Requirement	ADSP-2111	DSP56000
Internal RAM	2K words PM 1K words DM	256 words X 256 words Y
Single-cycle data fetch anywhere in memory	✓	No
Single-cycle fetch of two operands	✓	✓ *
Modify two addresses by two different modify values on every cycle	✓	✓ **

* Location dependent, some memory locations require multiple cycles for access.
** Address and modify registers must be used as a set. Re-indexing an address register requires resetting its corresponding modify register which requires two processor cycles.

Table 2. Summary of Data Addressing Capabilities

ADSP-2111 Program Sequencing

The program sequencer of the ADSP-2111 contains logic that selects a program memory source address and routes the address to the program memory address (PMA) bus. This address selection occurs automatically, in response to the current instruction. The address placed on the address bus can come from either

- 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, or
- the interrupt logic, for automatic vectoring to the appropriate interrupt service routine.

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.

All ADSP-2111 instructions execute in a single cycle. This applies equally to jumps and calls anywhere in the instruction space as well as interrupts. There is no instruction pipelining in the ADSP-2111, so program flow is easy to understand.

ADSP-2111 Looping Capabilities

The ADSP-2111 uses the structure

DO **endlabel** *UNTIL* **condition**

to implement zero overhead software loops. **endlabel** is the label assigned to the last instruction of the loop, and the **condition** can be any arithmetic or counter condition. A "forever" condition is also available to implement infinite loops.

When the *DO ... UNTIL ...* instruction is encountered, the address of the first instruction of the loop is pushed onto the program counter's PC stack, and the address of the last instruction (endlabel) of the loop is pushed onto the program counter's loop stack. Because the upper and lower bounds of the loop are stored internally to the program counter, the loop is able to execute without any additional overhead. DO loops can be nested four deep.

The *DO ... UNTIL ...* instruction operates in a single instruction cycle. For example, a counted loop of three instructions, e.g.,

```
     CNTR = 10;            {initialize counter}
    DO endlabel UNTIL CE; {do until counter expires}
         instruction 1;
         instruction 2;
endlabel: instruction 3;
```

will execute in 32 total instruction cycles; one cycle each for loading the counter and executing the *DO ... UNTIL ...* instruction, and 3 x 10 = 30 instruction cycles for loop execution. Note that program comments are contained between the brackets { }.

DSP56000 Program Sequencing

The Program Controller of the DSP56000 performs address generation (instruction prefetch), instruction decoding, DO loop control, and exception (interrupt) processing. The program controller implements a three-level pipelined architecture in which concurrent instruction, fetch, decode, and execution occur. Although the pipelining operation remains hidden from the user, for the most part it does impose a myriad of restrictions on the programmer. Most DSP56000 operations can execute in a single cycle. There are, however, several critical operations that require multiple instructions to execute, and thus can cause critical delays in program execution. Conditional jumps require up to three instruction cycles to execute. Returns from interrupts and subroutines

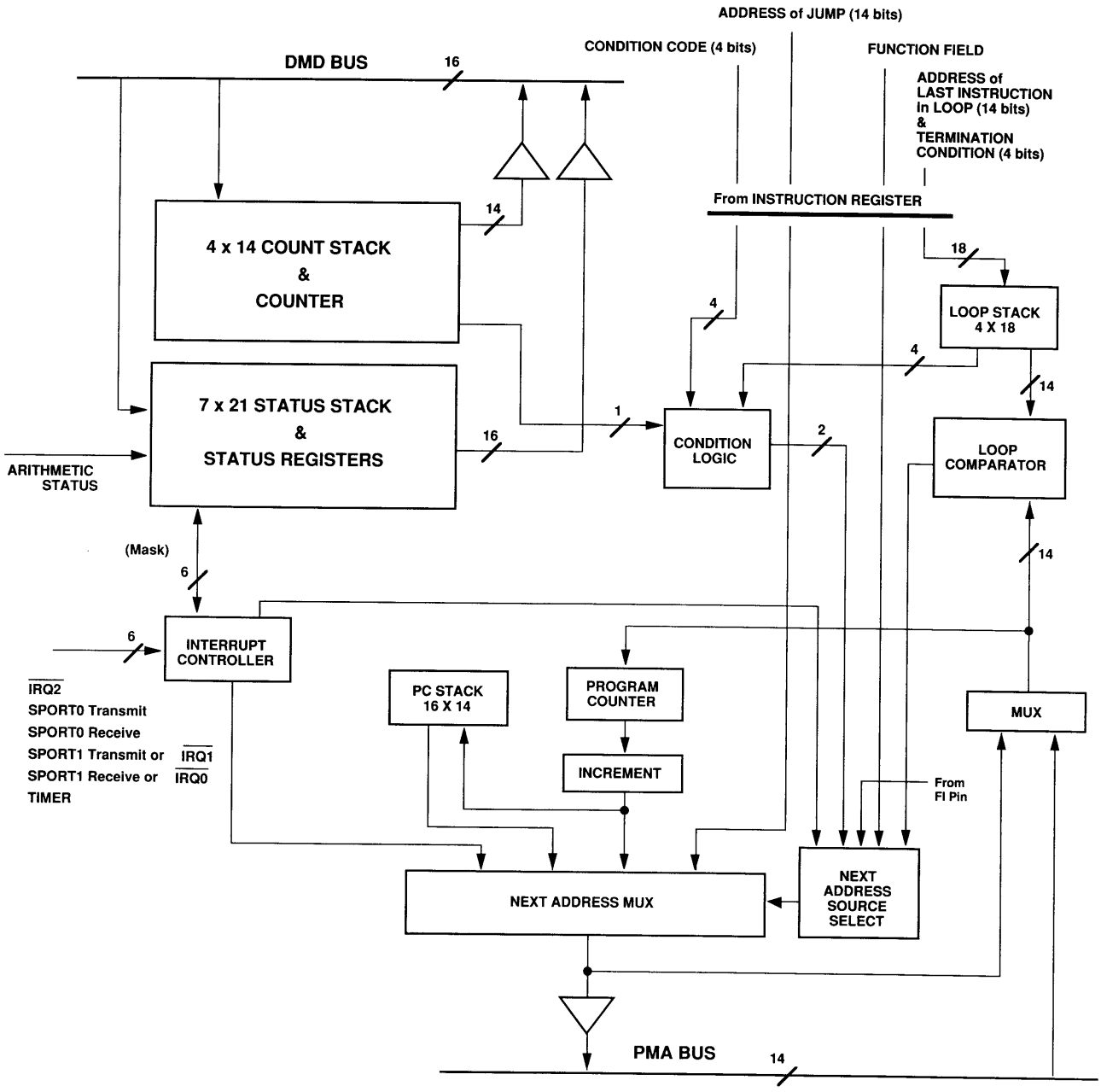

Figure 5. ADSP-2111 Program Sequencer

require two cycles. Because of these instruction latencies, restrictions are placed on which instructions can precede returns, *DO* and *ENDDO* instructions, MOVE, and conditional jump instructions. These multi-cycle instructions also consume extra program memory storage.

DSP56000 Looping Capabilities

The DSP56000 has looping capabilities similar to that of the ADSP-2111. Execution of the *DO* instruction requires three instruction cycles. During the first cycle, the current contents of the loop address and loop counter registers are pushed onto the system stack, and the *DO* instruction's source operand is loaded into the loop counter register. The loop counter register contains the remaining number of times the *DO* loop will be executed. During the second cycle, the current contents of the program counter and status register are pushed on the stack, and the *DO* instruction's address operand is loaded into the loop address register. During the third cycle, the loop flag is set. If the stack is managed to prevent overflow, loops can be stacked indefinitely.

In DSP56000 code, the previous looping example would look like the following.

```
        DO #10, ENDLOOP
          instruction1
          instruction2
          instruction3
ENDLOOP
```

Assuming that the three instructions each take one instruction cycle, this loop would take three instruction cycles to implement the *DO* and 3 x 10 = 30 instruction cycles for the loop—a total of 33. Care must be taken not to violate any

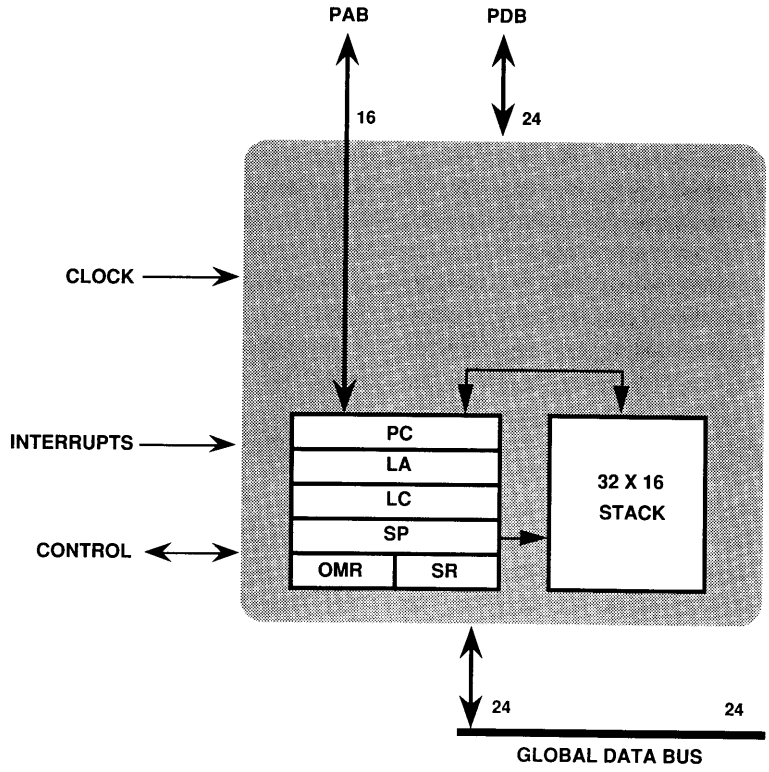

Figure 6. DSP56000 Program Controller

pipelining restrictions prior to the *DO* instruction. An *ENDDO* instruction is provided for early loop termination. Instruction processing would then resume at the first instruction following the loop.

Program Sequencing Summary

Table 3 highlights the key differences in program sequencing capability between the two processors. Due to the instruction pipeline delay, many restrictions are placed on the DSP56000 programmer in terms of the order in which certain instructions may be used. This could invariably lead to rearrangement of algorithms or cycle time wasted waiting for the pipeline to clear.

The ADSP-2111 has **no restrictions** on the order in which instructions can be executed. All ADSP-2111 instructions operate in a single cycle.

DATA COMMUNICATION

The computational efficiency of a processor is negated if the processor is unable to communicate easily and effectively with the outside world. The greatest performance enhancement occurs when the DSP is able to communicate with a variety of external devices, both serially and in parallel. Both the ADSP-2111 and the DSP56000 have provisions for serial and parallel data transfer.

ADSP-2111 Serial Ports

The ADSP-2111 provides two full duplex synchronous serial ports (SPORTs). Each SPORT has an independently programmable serial clock, programmable receive frame sync and transmit frame sync. These control signals can either be internally generated, or received from external devices. Word lengths for each SPORT are independent and user

DSP Requirement	ADSP-2111	DSP56000
All instructions execute in one processor cycle	✓	No
Single-cycle conditional jumps and subroutine calls	✓	No
Single-cycle returns from subroutines and interrupts	✓	No

Table 3. Summary of Program Sequencing Capabilities

programmable from 3 to 16 bits. Built-in hardware is also provided for automatic μ-law and A-law companding of data. The SPORTs support data transfer rates up to the full processor speed of 13 Mbits/second.

For additional functionality, SPORT0 can be set in multichannel mode. Block length can be set to either 24 or 32 words, providing an easy interface with time division multiplexed (TDM) systems. SPORT1 has the option of being configured either as a SPORT or may optionally be configured as two additional external interrupt pins and the Flag Out (FO) and Flag In (FI) pins.

The address generators of the ADSP-2111 can be used in conjunction with the serial ports to provide an automatic data buffering capability. Using the circular buffering capability of the address generator, data can either be received into or transmitted out of a buffer in memory automatically. Interrupts are generated when the buffer pointer wraps around at the end of the buffer, instead of after each word as in normal operation. Autobuffers of any arbitrary length are supported.

DSP56000 Serial Communication

Port C of the DSP56000 contains the processor's Serial Communication Interface (SCI) and Synchronous Serial Interface (SSI). The SCI is a full-duplex 8-bit data port. This interface uses three dedicated pins for transmit data, receive data, and serial clock. The SCI supports asynchronous bit rates and protocols or synchronous data transmission up to 2.5 Mbits/second.

The SSI is a full-duplex serial interface with programmable word length, protocol, clock, and transmit/receive synchronization. There are three modes of operation for the SSI—normal, on-demand or network. Normal mode is used for periodic data transfer, on-demand. No provision for automatic data companding is provided necessitating the use of a memory-based lookup table and additional companding software.

ADSP-2111 Host Interface Port

The ADSP-2111 Host Interface Port (HIP) is a parallel I/O port which allows the ADSP-2111 to act as a memory-mapped peripheral to a host computer. The host addresses the ADSP-2111 HIP as a section of 8-bit **or** 16-bit words of dual-ported memory. To the ADSP-2111, the HIP is a group of six memory-mapped, bi-directional data registers and two status registers. The ADSP-2111 HIP is software configurable for a variety of address, data and read/write strobe formats, allowing easy interface to most industry-standard microprocessors. ADSP-2111 HIP read or write operations can occur within a single 60 ns cycle, allowing several ADSP-2111s to operate at full speed in multiprocessor systems. The HIP is completely asynchronous with the rest of the ADSP-2111 and can be read or written to by the host with the DSP operating at full speed. ADSP-2111 HIP operation can be either interrupt driven or used through software polled operation. The HIP can also be used to load programs from a host processor into internal program memory RAM.

DSP56000 Host Interface

Port B of the DSP56000 can be configured to provide an asynchronous, double-buffered, 8-bit only Host Interface (HI). From the host processor's viewpoint, the DSP56000 HI consists of eight, 8-bit wide, consecutive memory locations for control and data transfer. Three of these locations are for data received from the DSP, three for data transmitted to the DSP. The DSP sees one 24-bit, write-only, host data transmit register; one 24-bit, read-only, host data receive register; a host control register; and a read-only host status register. The DSP56000 HI can be used with either software polled, interrupt driven, or direct memory access protocols.

Table 4 summarizes the data communication capabilities of the ADSP-2111 and the DSP56000.

MEMORY RESOURCES

The availability of on-chip memory simplifies DSP system design by reducing the overall chip count in a system.

The ADSP-2111 provides users with 2K words internal Program Memory RAM and 1K words Data Memory RAM. The modified Harvard architecture allows for data as well as code to be stored in Program Memory. Any portion of the internal program memory is available as mask-programmed ROM.

The DSP56000 has only 256 words X memory RAM and 256 words Y memory RAM internal to the device. 3.75K words of program memory ROM is also provided on chip. The DSP56001 contains the same internal X and Y memory as the DSP56000, with the addition of 512 words Program Memory RAM and optional bootstrap ROM.

SYSTEM CONSIDERATIONS

A clean external architecture, with the ability to connect easily and inexpensively to system resources, eases DSP system design and enhances processor performance.

For the ADSP-2111, the system designer has the choice of using either a clock oscillator or simple microprocessor-grade crystal. The clock or crystal used runs at a 1 x instruction cycle rate, or 16 MHz. The boot memory is typically an inexpensive, single byte-wide EPROM. The boot circuitry of the ADSP-2111 supports up to eight different 2K-word pages of boot memory. If Host Booting is disabled, the ADSP-2111 will automatically load its internal Program Memory from Page 0 of the Boot PROM. The other boot pages are accessible under software control. This feature allows the ADSP-2111 to perform up to eight completely distinct programs without any outside intervention, thereby increasing the flexibility of any ADSP-2111 design. Program modules and variables can also be defined such that data and code can be shared between boot pages.

The DSP56000 system designer also has the choice of using a clock oscillator or crystal, but the DSP56000 requires a 2 x instruction cycle rate clock (27 MHz for a 13.5 MHz instruction rate), necessitating extra care due to the presence of higher frequency signals in the system. For automatic booting into internal RAM, the DSP56001 has available, as an option,

DSP Requirement	ADSP-2111	DSP56000
Host port data registers	6 bi-directional	1 read-only 1 write-only
Host port configurable for 8- or 16-bit operation	✓	No
Serial ports	2 independent synchronous ports	1 synchronous 1 asynchronous
Built-in data companding hardware	✓	No

Table 4. Summary of Data Communication Capabilities

bootstrap code installed in internal ROM which will copy the contents of a byte-wide PROM into internal memory. The DSP56000 has no such provision. Even with the bootstrap circuitry, no boot paging scheme is supported, thereby relegating the part to a single type of functionality. Because the DSP56000 is an internal-ROM based part, quick changes or field upgrades are difficult.

DEVELOPMENT ENVIRONMENT

A complete set of easy-to-use development software and hardware is essential for system development with any DSP processor.

The ADSP-2111 is supported by a complete set of software and hardware development tools. The software development environment consists of the System Builder, Assembler, Linker, PROM Splitter, and full-featured Simulator. A C-compiler is also available. On the hardware side, Analog Devices offers the ADSP-2111 EZ-LAB™, a stand-alone hardware development board, as well as two grades of in-circuit emulators, each capable of full-speed processor emulation.

Software for the DSP56000 consists of the Macro Cross Assembler, Linker/Librarian and Simulator. The hardware development system consists of the Application Development System (ADS). While the ADS can offer emulator-like single stepping, it has no provision for full processor emulation in an arbitrary customer board.

SUMMARY

The DSPs examined in this application note take different approaches to meeting the key DSP processor requirements. The DSP56000 places its emphasis on high dynamic range MAC operations at the expense of general purpose processing flexibility. The ADSP-2111's parallel-connected arithmetic architecture, single-cycle instruction execution and flexible host port allow it to conform easily to meet algorithm requirements instead of algorithms being required to meet processor limitations.

While we're on the subject of comparison....

If the level of on-chip integration in the ADSP-2111 is not a system requirement, but high performance and low cost are priorities, then the ADSP-2105 should merit consideration.

The ADSP-2105 is a 10 MIPS processor with 1K words internal Program Memory RAM, 512 words internal Data Memory RAM and one serial port. The device is code-compatible with the ADSP-2111, as well as with all other ADSP-2100 family members (ADSP-2100A, ADSP-2101). The ADSP-2105, along with all ADSP-2100 family members, shares the same internal core architecture (arithmetic, addressing and program sequencing units) and features such as zero overhead looping and branching, circular buffering, and single-cycle instruction execution.