

Technical Summary

96-BIT GENERAL-PURPOSE FLOATING-POINT DIGITAL-SIGNAL PROCESSOR (DSP)

The DSP96001, a floating-point version of the fixed-point DSP56001, is the first member of Motorola's Family of single-chip HCMOS, low-power, general-purpose, IEEE floating-point DSPs. The DSP96001 features 512 words of full-speed on-chip program random-access memory (PRAM), two preprogrammed data ROMs, special on-chip bootstrap hardware to permit convenient loading of user programs into the PRAM, and on-chip debug circuitry to permit access to internal resources in support of circuit emulation (OnCE™). The DSP96001 is an off-the-shelf part (no user-programmable ROMs on chip), which is software compatible with the HYPERformance™ DSP56000 Family of fixed-point DSPs.

The central processing unit (CPU) consists of three 32-bit execution units operating in parallel: the data ALU, the address generation unit (AGU), and the program controller. The DSP96001 has MCU-style on-chip peripherals, program and data memory, and a memory expansion port, which facilitates interfacing to page mode and video RAMs. The MPU-style programming model and instruction set allows straightforward generation of efficient, compact code.

The 40-million floating-point operations per second (MFLOPS) peak performance of the DSP96001 makes it well-suited for real-time DSP applications requiring IEEE floating point. These applications include high-speed control, digital audio, numeric processing, image and speech processing, spectral analysis, instrumentation, medical, and navigation. The main features facilitating this throughput are as follows:

- **Speed** — At 13.33-million instructions per second, the DSP96001 can execute a 1024 point complex fast Fourier transform (FFT) in less than 2 milliseconds. The CPU can process over 2 million interrupts per second.
- **Precision** — The data arithmetic logic unit (ALU) provides full conformance with the IEEE 754-1985 Standard for Binary Floating-Point Arithmetic. All four rounding modes are supported: 1) round to nearest (even), 2) round to zero, 3) round to minus infinity, and 4) round to plus infinity. Infinities, nonnumbers, and denormalized numbers are handled according to the standard. Source and destination operands are held in a file of ten, 96-bit, extended-precision registers. The data ALU also supports integer arithmetic including a 32×32 multiplication with a full, nontruncated, 64-bit product.
- **Parallelism** — The data ALU, AGUs, and program controller operate in parallel within the CPU so that an instruction prefetch, up to three floating-point operations, two data moves, and two address pointer updates, using one of three types of arithmetic (linear, modulo, or reverse carry), can be executed in a single instruction cycle. This parallelism allows 40 MFLOP peak performance. Two, on-chip direct memory access (DMA) controller channels operate unobtrusively in parallel with the CPU to provide memory-to-memory or memory-to-peripheral transfers, also using one of three types of address update arithmetic (linear, modulo, or reverse carry).
- **Integration** — In addition to the three, independent execution units and two channels of DMA, the DSP96001 has six, on-chip memories, three, on-chip, MCU-style peripherals (serial communication interface (SCI), synchronous serial interface (SSI), and 32-bit host interface), a clock generator, and eight, 32-bit-wide buses (three address and five data), making the overall high-performance, low-power, compact system a good cost-performance value.
- **Invisible Pipeline** — The fetch-decode-execute instruction pipeline is essentially invisible to the programmer, thus allowing straightforward program development in either assembly language or a high-level language such as a full Kernighan & Ritchie C.

HYPERformance and OnCE are trademarks of Motorola Inc.

This document contains information on a new product. Specifications and information herein are subject to change without notice.



Features (Continued)

- Instruction Set** — The instruction mnemonics are MCU-like, making the transition from programming microprocessors to programming the DSP96001 as easy as possible. Regular register files for data and addresses enhance the efficiency of high-level language compilers. The orthogonal syntax supports controlling the parallel execution units. The no-overhead DO instruction and the repeat (REP) instruction make writing straightline code outdated. The program counter (PC) relative addressing supports writing position-independent code.
- DSP56000 Compatibility** — The DSP96001 has the same basic architecture as the DSP56000 Family. The instruction set is a super set of the DSP56000 Family mnemonics. This compatibility means the hardware and software development tools are nearly identical for all Motorola DSPs; therefore, investment in existing software is preserved. Applications can be easily developed on one processor and migrated across

the Motorola DSP product line to meet various cost and performance objectives.

SIGNAL DESCRIPTION

The DSP96001 is an 163-pin integrated circuit available in pin grid array packaging. Its input and output signals are organized into seven functional groups listed below and illustrated in Figure 1.

Port A Address, Data, and Control Buses
 Interrupt and Mode Control
 Power and Clock
 Host Interface or Peripheral I/O
 SCI or Peripheral I/O
 SS1 or Peripheral I/O
 On-Chip Debug

Table 1 includes descriptions and mnemonic symbols for each signal name.

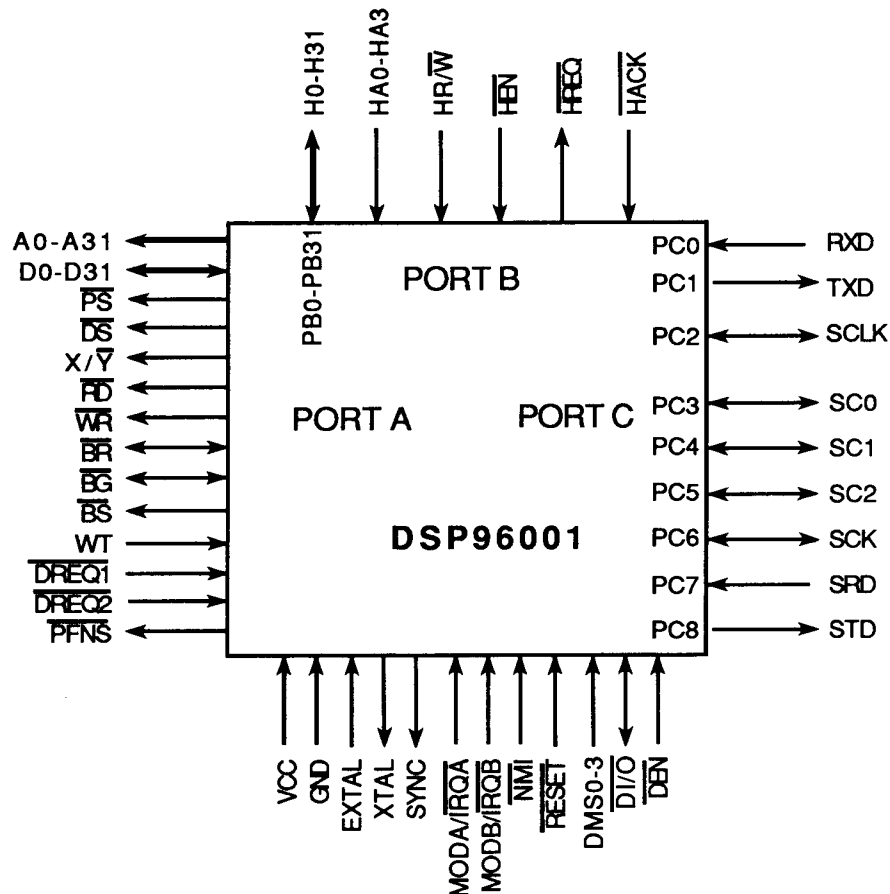


Figure 1. DSP96001 Functional Signal Groups

Table 1. Signal Summary (Sheet 1 of 3)

Signal Name	Mnemonic	Description
Port A Address, Data, and Control Buses		
Address Bus	A0-A31	These three-state output pins specify the address for external program and data memory accesses. To minimize power dissipation, A0-A31 do not change state when external memory spaces are not being accessed. A0-A31 are in the high-impedance state when \overline{BG} output is asserted or during processor reset.
Data Bus	D0-D31	These pins provide the bidirectional data bus for external program and data memory accesses. D0-D31 are in the high-impedance state when \overline{BG} output is asserted or during processor reset.
Program Memory Select	\overline{PS}	This three-state output is asserted only when external program memory is referenced. \overline{PS} is in the high-impedance state when \overline{BG} output is asserted or during processor reset. \overline{PS} has the same timing as A0-A31.
Data Memory Select	\overline{DS}	This three-state output is asserted only when external data memory is referenced. \overline{DS} is in the high-impedance state when \overline{BG} output is asserted or during processor reset. \overline{DS} has the same timing as A0-A31.
X/Y Select	X/\overline{Y}	This three-state output selects which external data memory space (X or Y) is referenced by data memory select \overline{DS} . X/\overline{Y} is in the high-impedance state when \overline{BG} output is asserted or during processor reset. X/\overline{Y} has the same timing as A0-A31.
Read Enable	\overline{RD}	This three-state output is asserted to read external memory from the data bus D0-D31. \overline{RD} is in the high-impedance state when \overline{BG} output is asserted or during processor reset. Therefore, \overline{RD} may require an external pullup in some systems.
Write Enable	\overline{WR}	This three-state output is asserted to write external memory to the data bus D0-D31. \overline{WR} is in the high-impedance state when \overline{BG} output is asserted or during processor reset. Therefore, \overline{WR} may require an external pullup in some systems.
Bus Request	\overline{BR}	This bidirectional pin allows another device, such as a processor or DMA controller, to become the master of the external data bus D0-D31 and external address bus A0-A31. When programmed as an input, asserting \overline{BR} will cause the DSP96001 to release the external data bus D0-D31, address bus A0-A31, and bus control pins \overline{PS} , \overline{DS} , X/\overline{Y} , \overline{RD} , and \overline{WR} (i.e., port A), by placing these pins in the high-impedance state after the execution of the current instruction has been completed. When programmed as an output, the DSP96001 can request control of the bus by asserting \overline{BR} .
Bus Grant	\overline{BG}	This three-state bidirectional pin is asserted to acknowledge an external bus request. When \overline{BR} is programmed as an input, then \overline{BG} is an output, which is asserted after port A has been released. When \overline{BR} is programmed as an output, \overline{BG} is an input, which, when asserted, signals the DSP96001 to take control of the bus.
Bus Strobe	\overline{BS}	This output is a strobe that indicates an active external bus cycle. \overline{BS} is asserted synchronously when either \overline{PS} or \overline{DS} is asserted, and is deasserted synchronously when either \overline{RD} or \overline{WR} is deasserted. Wait states will be inserted if WT is asserted when \overline{BS} is asserted, until WT is deasserted. \overline{BS} is in the high-impedance state when \overline{BG} is asserted or during processor reset.
Wait	WT	This input is used to asynchronously extend an external bus cycle an arbitrary number of wait states. The minimum number of wait states that can be inserted is two. WT can be asserted asynchronously with \overline{BS} assertion to insert wait states. WT must be deasserted synchronously with \overline{BS} for the number of wait states inserted to be deterministic.
DMA Request	$\overline{DREQ1}$ $\overline{DREQ2}$	These inputs, used by the external system to request service from the on-chip DMA controller, are ignored whenever the internal DMA channel is programmed to work without handshake.
Page Fault	\overline{PFNS}	This output, which facilitates interfacing with page-mode dynamic RAMs and video RAMs, has the same timing as \overline{BS} .

— Continued —

Table 1. Signal Summary (Sheet 2 of 3)

Signal Name	Mnemonic	Description
Interrupt and Mode Control		
Mode Select A/External Interrupt Request A OR Mode Select B/External Interrupt Request B	MODA/ $\overline{\text{IRQA}}$ MODB/ $\overline{\text{IRQB}}$	These two inputs have dual functions: 1) to select the initial chip operating mode and 2) to receive an interrupt request from an external source. MODA and MODB are read and internally latched in the DSP when the RESET pin is deasserted. After the processor leaves the reset state, the MODA and MODB pins automatically change to external interrupt requests $\overline{\text{IRQA}}$ and $\overline{\text{IRQB}}$. After leaving the reset state, the chip operating mode can be changed by software. $\overline{\text{IRQA}}$ and $\overline{\text{IRQB}}$ may be programmed to be level sensitive or negative edge triggered. When edge triggered, triggering occurs at a voltage level and is not directly related to the fall time of the interrupt signal; however, the probability of noise on $\overline{\text{IRQA}}$ or $\overline{\text{IRQB}}$ generating multiple interrupts increases with increasing fall time of the interrupt signal. The stop state can be exited by asserting $\overline{\text{IRQA}}$.
Nonmaskable	$\overline{\text{NMI}}$	This edge-triggered input is used to request a nonmaskable interrupt. Triggering occurs at a voltage level and is not directly related to the fall time of the interrupt signal; however, the probability of noise on the $\overline{\text{NMI}}$ pin generating multiple interrupts increases with increasing fall time of the interrupt signal.
Reset	$\overline{\text{RESET}}$	This Schmitt trigger input is used to reset the DSP96001. When $\overline{\text{RESET}}$ is asserted, the DSP96001 is initialized and placed in the reset state. When $\overline{\text{RESET}}$ is deasserted, the initial chip operating mode is latched from the MODA and MODB pins. When coming out of reset, deassertion occurs at a voltage level and is not directly related to the rise time of the reset signal. However, the probability of noise on $\overline{\text{RESET}}$ generating multiple resets increases with increasing rise time of the reset signal.
Power and Clock		
Power Ground	VCC GND	There are five sets of power and ground pins: three pairs for internal logic; two power and five ground for port A address and control pins; two power and four ground for port A data pins; one power and two ground for the HOST port; and one power and two ground for peripherals.
External Clock/ Crystal Input	EXTAL	This input may be used to interface the internal crystal oscillator to an external crystal or an external clock. The maximum clock rate is 26.67 MHz.
Crystal Output	XTAL	This output connects the internal crystal oscillator output to an external crystal. If an external clock is used, XTAL should not be connected.
Synchronization Clock	SYNC	This output is used to synchronize bus timing and serial debug port functions with four-phase internal clock.
Host Interface		
Host Data Bus	H0-H31	This bidirectional data bus is used to transfer data between the host processor and the DSP96001. This bus is an input unless enabled by a host processor read. H0-H31 may be programmed as general-purpose, parallel I/O pins (PB0-PB31) when the host interface is not being used.
Host Address	HA0-HA3	These inputs, which provide the address selection for each host interface register, must be stable when $\overline{\text{HEN}}$ is asserted.
Host Read/Write	$\overline{\text{HR/W}}$	This input, which selects the direction of data transfer for each host processor access, must be stable when $\overline{\text{HEN}}$ is asserted.
Host Enable	$\overline{\text{HEN}}$	This input enables a data transfer on the host data bus. When $\overline{\text{HEN}}$ is asserted and $\overline{\text{HR/W}}$ is high, H0-H31 become outputs, and DSP96001 data may be read by the host processor. When $\overline{\text{HEN}}$ is asserted and $\overline{\text{HR/W}}$ is low, H0-H31 become inputs, and host data is latched inside the DSP when $\overline{\text{HEN}}$ is deasserted. Normally, a chip select signal, derived from host address decoding, and an enable clock are used to generate $\overline{\text{HEN}}$.
Host Request	$\overline{\text{HREQ}}$	This open-drain output is used by the DSP96001 host interface to request service from the host processor, DMA controller, or simple external controller.
Host Acknowledge	$\overline{\text{HACK}}$	This input has two functions: 1) to receive a host acknowledge handshake signal for DMA transfers and 2) to receive a host interrupt acknowledge signal compatible with MC68000 Family processors.

Table 1. Signal Summary (Sheet 3 of 3)

Signal Name	Mnemonic	Description
Serial Communications Interface		
Receive Data	RXD	This input receives byte-oriented serial data into the SCI receive shift register. Input data is sampled on the positive edge of the receive clock. RXD may be programmed as a general-purpose I/O pin (PC0) when the SCI RXD function is not being used.
Transmit Data	TXD	This output transmits serial data from the SCI transmit shift register. Data changes on the negative edge of the transmit clock. This output is stable on the positive edge of the transmit clock. TXD may be programmed as a general-purpose I/O pin (PC1) when the SCI TXD function is not being used.
SCI Serial Clock	SCLK	This bidirectional pin provides an input or output clock from which the transmit and/or receive baud rate is derived in the asynchronous mode and from which data is transferred in the synchronous mode. SCLK may be programmed as a general-purpose I/O pin (PC2) when the SCI SCLK function is not being used.
Synchronous Serial Interface		
Serial Control Zero	SC0	This bidirectional pin, which is used for control by the SSI, may be programmed as a general-purpose I/O pin (PC3) when the SSI SC0 function is not being used.
Serial Control One	SC1	This bidirectional pin is used for control by the SSI. SC1 may be programmed as a general-purpose I/O pin (PC4) when the SSI SC1 function is not being used.
Serial Control Two	SC2	This bidirectional pin, which is used for control by the SSI, may be programmed as a general-purpose I/O pin (PC5) when the SSI SC2 function is not being used.
SSI Serial Clock	SCK	This bidirectional pin provides the serial bit rate clock for the SSI when only one clock is used. SCK may be programmed as a general-purpose I/O pin (PC6) when the SSI is not being used.
SSI Receive Data	SRD	This input pin, which receives serial data into the SSI receive shift register, may be programmed as a general-purpose I/O pin (PC7) when the SSI SRD function is not being used.
SSI Transmit Data	STD	This output pin transmits serial data from the SSI transmit shift register. STD may be programmed as a general-purpose I/O pin (PC8) when the SSI STD function is not being used.
On-Chip Emulation (OnCE)		
Debug Mode Select	DMS0-DMS3	These inputs are used to select the debug function to be executed.
Debug Data	DI/O	This bidirectional pin transfers serial data to/from the on-chip debug controller.
Debug Enable	\overline{DEN}	This input is used to enable the on-chip debug controller.

BLOCK DIAGRAM DESCRIPTION

The DSP96001 architecture has been designed to maximize throughput in data-intensive real-time DSP applications requiring floating-point arithmetic. This objective has resulted in a dual-natured, expandable architecture with sophisticated on-chip peripherals and general-purpose I/O. Two independent, expandable data memory spaces, two address arithmetic units, and two on-chip DMA controllers make the architecture dual natured. The duality of the architecture makes it easier to write software for DSP applications. For example, data is naturally partitioned into X and Y coordinates for graphics and image-processing applications, into coefficient and data spaces for filtering and transformations, and into real and imaginary spaces for performing complex arithmetic.

The major components of the DSP96001 are as follows:

- Data Buses
- Address Buses

- Data ALU
- Address Generation Unit
- X Data Memory
- Y Data Memory
- Program Memory
- Program Controller
- Bootstrap ROM
- Input/Output
 - Expansion Port
 - Dual-Channel DMA Controller
 - General-Purpose I/O
 - Random-Number Generator
 - Host Interface
 - Serial Communication Interface
 - Synchronous Serial Interface
 - On-Chip Debugger

These components are depicted in Figure 2 and described in the following paragraphs.

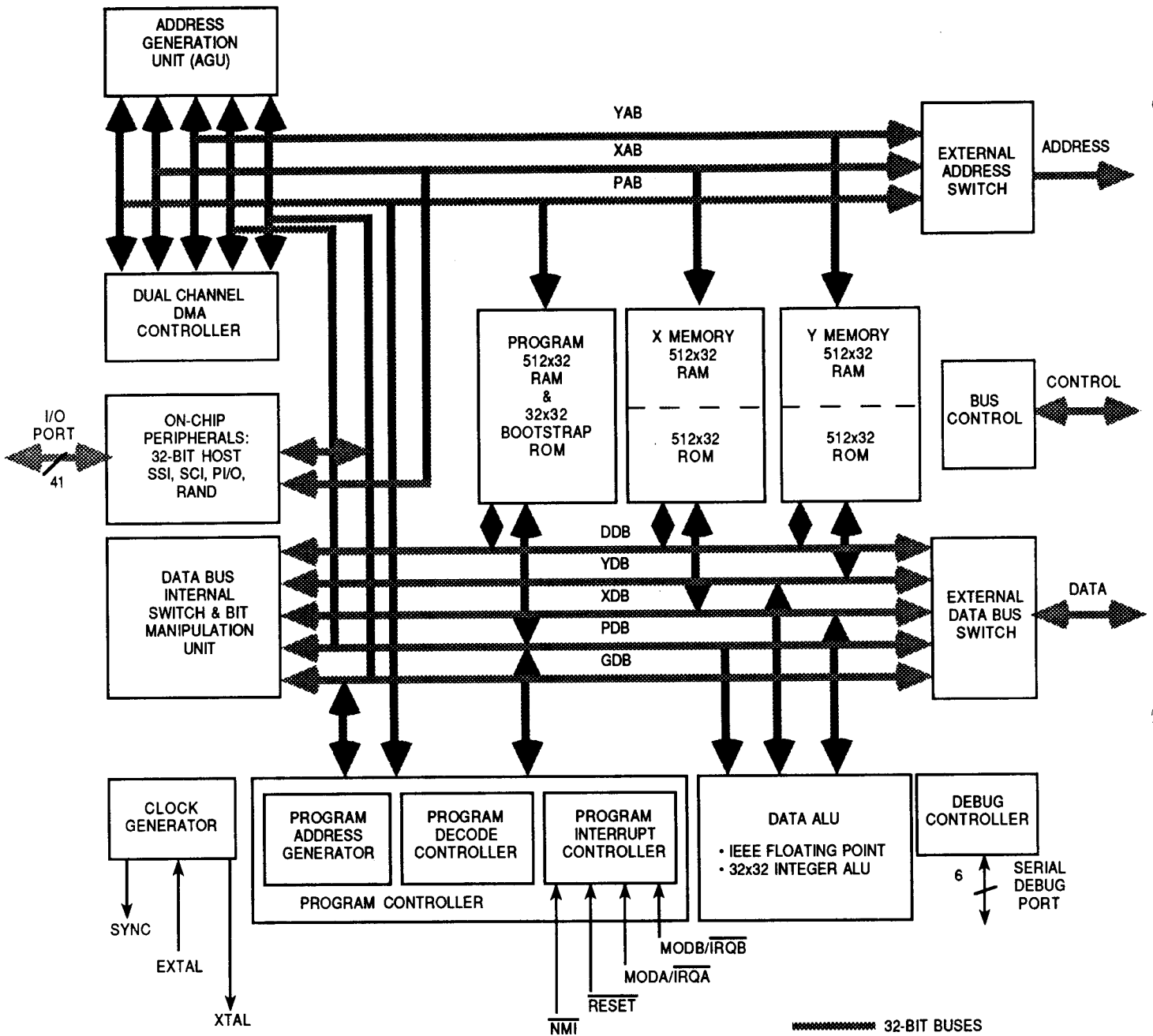


Figure 2. DSP96001 Block Diagram

DATA BUSES

Data movement on the chip occurs over five, bidirectional, 32-bit buses: the X bus (XDB), the Y bus (YDB), the program bus (PDB), the DMA bus (DDB), and the global bus (GDB). The X and Y buses may also be treated by certain instructions as one 64-bit data bus by concatenation of XDB and YDB. Data transfers between the data ALU and the X memory and Y memory occur over

XDB and YDB, respectively. XDB and YDB are kept local on the chip to maximize speed and minimize power dissipation. DMA transfers occur over the DDB; all other data transfers, such as I/O transfers to peripherals, occur over the GDB. Instruction word prefetches take place in parallel over the PDB. Transfers between buses are accomplished in the internal bus switch. In general, when transfers are made with registers that are not 32 bits, the unused bits read as zero and should be written with zero

for future compatibility. The bit manipulation unit, which resides in the internal bus switch, has access to all data buses and implements the bit manipulation instructions.

ADDRESS BUSES

Addresses are specified for internal X memory and Y memory on two, unidirectional, 32-bit buses — X address bus (XAB) and Y address bus (YAB). Program memory addresses are specified on the program address bus (PAB). External memory spaces are addressed via a single, 32-bit, unidirectional address bus driven by a three-input multiplexer that can select either XAB, YAB, or PAB. There is no speed penalty if only one external memory space is accessed in an instruction. If two or three external memory spaces are accessed in a single instruction, there will be a one- or two-instruction-cycle execution delay, respectively. A bus arbitrator prioritizes and schedules external access requests by the CPU and DMA controller. On-chip peripherals and the DMA controller are memory mapped in the internal X memory space. The XAB, YAB, and PAB are used twice in each instruction cycle to allow concurrent access to memory by the CPU and DMA controller.

DATA ALU

All the arithmetic (both fixed point and floating point) and logical operations are performed on data operands in the data ALU. Both signed and unsigned fixed-point arithmetic is implemented in a single cycle, which supports high-level language operations such as program flow constructs and arithmetic using integer data types. Single-cycle add, subtract, and multiply operations conform to the IEEE 754-1985 Standard for Binary Floating Point Arithmetic. Calculations are made to infinite precision and then rounded to single-precision (SP) or single-extended-precision (SEP) formats in hardware or double-precision (DP) and double-extended-precision (DEP) formats in software. Hardware is provided to support all four rounding modes: round to nearest (even), round to zero, round to plus infinity, and round to minus infinity. Plus and minus infinity, not-a-number (NaN), and denormalized numbers are also supported. When denormalized numbers are detected in the default mode, extra instruction cycles are inserted for normalization. In the flush-to-zero (FLUSH) mode, denormalized numbers are treated as zero and extra cycles are never required. A complete set of 32-bit logical operations is provided to support high-level language and controller applications.

The data ALU hardware consists of four blocks: a general-purpose register file, a floating-point multiplier, a floating-point adder/subtractor, and a 32-bit barrel shifter. The register file consists of 10 registers, each 96 bits long. These registers support extended-precision formats, DSP56000 compatibility, and 32- or 64-bit data transfers over XDB and YDB. ALU results are always stored in one of the general-purpose registers; floating-point ALU results are always 96 bits; integer (fixed-point) ALU results are either 32 or 64 bits. The multiplier supports SP or SEP floating-point formats with up to 32-bit mantissas and 11-bit exponents and 32×32 integer multiplication with a full 64-bit product. The IEEE floating-point adder/

subtractor supports SP, SEP in a single cycle and will generate the sum and difference of the same two operands in one cycle. This capability is particularly useful for calculating FFTs. The adder/subtractor also supports the fixed-point operations. The 32-bit barrel shifter provides the necessary shifting for normalizing floating-point numbers and scaling integers.

ADDRESS GENERATION UNIT

The AGU performs all of the address storage and effective address calculations necessary to address data operands in memory. The AGU implements three types of arithmetic (linear, modulo, and reverse carry) and operates in parallel with other chip resources to minimize address generation overhead. The AGU contains eight address registers (R0-R7), eight offset registers (N0-N7), and eight modifier registers (M0-M7). The Rn are 32-bit registers usually containing an address pointer. Each Rn register may be accessed for output to the XAB, YAB, and PAB address buses. The Nn and Mn registers are 32-bit registers, which are normally used to control updating of the Rn registers. Additional hardware provides the capability of doing PC relative and immediate offset addressing. Rn, Nn, and Mn registers can also be used for general-purpose storage.

AGU registers may be read or written via the global data bus as 32-bit operands. The AGU has two modulo arithmetic units, which can generate two, independent, 32-bit addresses for any two of the XAB, YAB, or PAB every instruction cycle. The AGU can directly address 4,294,967,296 locations on the XAB; 4,294,967,296 locations on the YAB; and 4,294,967,296 locations on the PAB — a total capability of 12,884,901,888, 32-bit data words.

MEMORIES

Three independent memory spaces of the DSP96001 (X, Y, and program) are shown in Figure 3. The memory spaces are configured by control bits MA, MB, and DE in the operating mode register. MA and MB control the program memory map and select the reset vector address. DE controls the X and Y memory maps, enabling the internal X and Y ROMs. The on-chip X and Y ROMs are disabled upon reset.

X Memory

On-chip X data RAM is a 32-bit-wide internal memory that occupies the lowest 512 locations in X memory space. The on-chip X ROM occupies locations 512 through 1023 in X memory space when enabled by setting DE = 1 in the operating mode register. The X ROM is factory programmed with a positive, two-quadrant cosine table useful for FFTs, discrete Fourier transforms (DFTs), and waveform generation. The on-chip peripheral registers occupy the top 128 locations of X memory. A special I/O short addressing mode provides efficient access to these locations. Addresses are received from the XAB, and data transfers occur on the XDB. Internal X memory may be accessed twice during a cycle — once by the CPU and once by the DMA controller. X memory may be expanded to four gigawords off-chip.

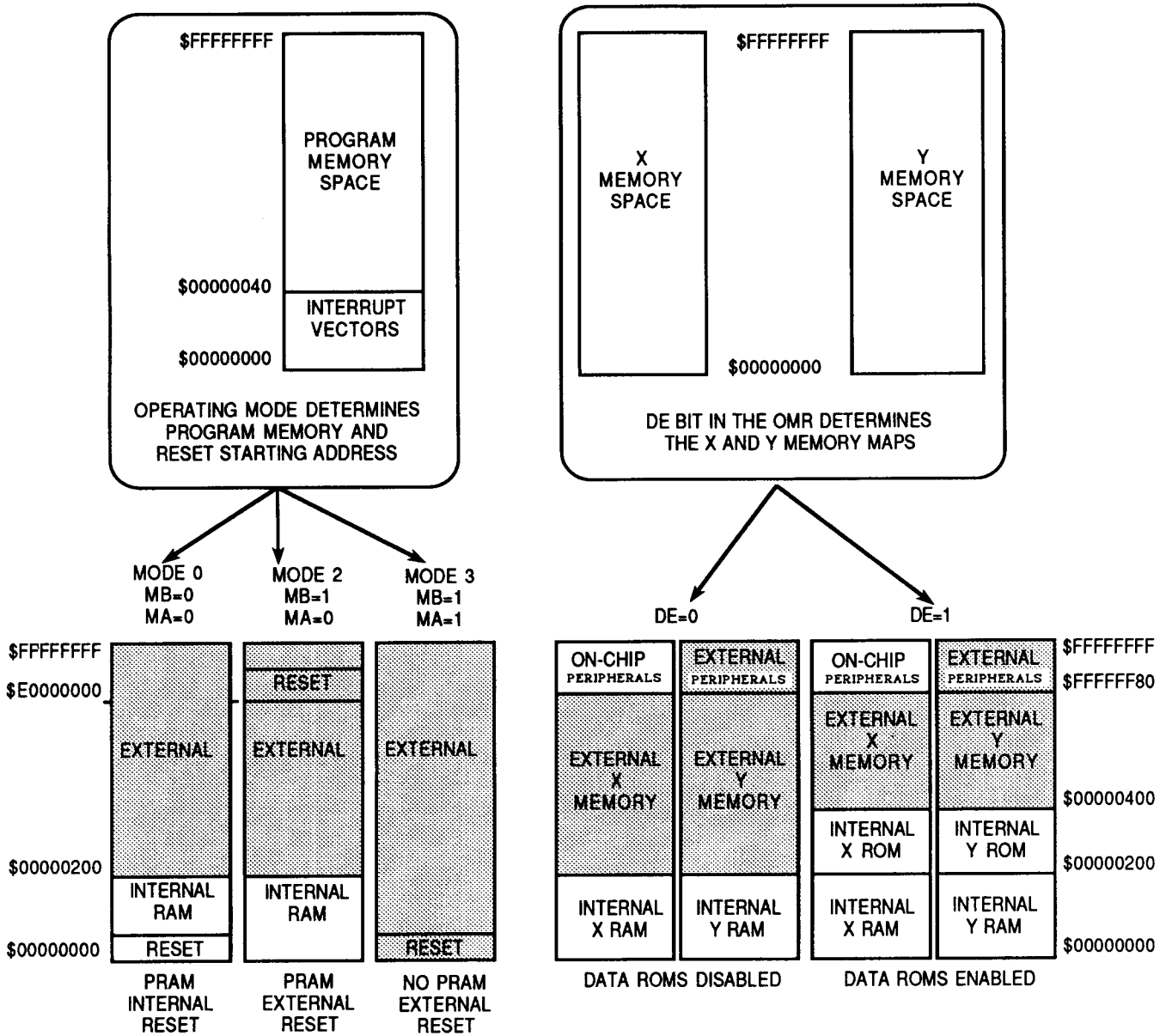


Figure 3. DSP96001 Memory Map

Y Memory

On-chip Y RAM is a 32-bit-wide internal memory that occupies the lowest 512 locations in Y memory space. The on-chip Y ROM occupies locations 512 through 1023 in Y memory space when enabled by setting DE = 1 in the operating mode register. The Y ROM is factory programmed with a positive, two-quadrant sine table useful for FFTs, DFTs, and waveform generation. External I/O devices should be mapped into the top 128 locations of Y memory. A special I/O short addressing mode provides efficient access to these locations. Addresses are received from the YAB, and data transfers occur on the YDB. Internal Y memory may be accessed twice during a cycle — once by the CPU and once by the DMA controller. Y memory may be expanded to four gigawords off-chip.

Program Memory

On-chip PRAM consists of a 512 × 32-bit, high-speed RAM, which is enabled by the MA and MB bits in the operating mode register. Addresses are received from the program control (usually the PC) over the PAB. Program memory may be written using MOVEM instructions or the DMA controller. The interrupt vector addresses are located in the bottom 64 locations of program memory. Program memory may be accessed twice during an instruction cycle — once by the CPU and once by the DMA controller. Program memory may be expanded to four gigawords off-chip.

PRAM has many advantages; it provides the user a means to develop code efficiently. The programs can be changed dynamically using the on-chip DMA to efficiently overlay DSP software algorithms, and interrupt

routines can be located on-chip for maximum efficiency, especially with fast interrupts. In this way, the on-chip PRAM operates as a fixed cache, thereby minimizing contention with accesses to external data memory spaces.

The bootstrap mode provides a convenient, low-cost method to load the DSP96001 PRAM with the user's program after poweron reset. The PRAM may be loaded from a single, inexpensive, external byte-wide EPROM or via the host interface from a host processor.

Bootstrap ROM

Bootstrap ROM is a 32×32-bit, factory-programmed ROM used only in the bootstrap mode, operating mode 1. The bootstrap mode may be selected with the external mode pins during processor reset or by writing the operating mode register at any time. The bootstrap ROM is not accessible by the user and is disabled in normal operating modes.

PROGRAM CONTROLLER

The program controller performs instruction prefetch, instruction decoding, hardware DO loop control, and exception processing. It consists of a program address controller (PAC), program decode controller (PDC), and the program interrupt controller (PIC).

Program Address Controller

The PAC provides the instruction prefetch address. Normal sequential addressing is modified by program flow control instructions, hardware DO loop processing, and interrupts.

Program Decode Controller

The PDC decodes instructions and provides control signals to the on-chip resources.

Program Interrupt Controller

The PIC prioritizes all pending interrupt requests and generates the interrupt vector addresses. Two interrupt types are provided: long interrupts and fast interrupts.

All interrupts start as long interrupts where the standard context switch is performed. There is no context switch performed for fast interrupts; instead, the two interrupt routine words are jammed into the instruction pipeline, and a context switch is not performed.

INPUT/OUTPUT

The I/O capability of the DSP96001 is extensive and advanced. The on-chip peripherals facilitate interfacing into a variety of system configurations, including multiple DSP96001 systems (with or without a host processor), global bus systems with bus arbitration, external memory, and many serial configurations, all with minimal glue logic. Each peripheral has its own control, status, and data registers and is treated as memory-mapped I/O by the DSP96001. Dedicated I/O interfaces have several interrupt vector addresses and control bits to enable/disable interrupts, which minimizes the overhead associated with servicing the device since each interrupt source can have its own service routine (Figure 4). The interrupt vectors can be programmed to one of three maskable priority levels.

Specifically, the I/O structure consists of an extremely flexible memory expansion port (port A), two channels of DMA, a random-number generator (RAND), 41 additional I/O pins, and two general-purpose interrupt pins, \overline{IRQA} and \overline{IRQB} . The 41 pins may be used as general-purpose I/O pins (port B and port C), or allocated to an on-chip peripheral under software control. Three dedicated interfaces are provided on the DSP96001: a 32-bit parallel host MPU/DMA interface (HOST), an SCI, and an SSI. Port B is a 32-bit I/O interface that may be used as general-purpose I/O pins or as host-interface pins. Port C is a 9-bit I/O interface that may be used as general-purpose I/O pins or as SCI and SSI pins.

Expansion Port (Port A)

The DSP96001 expansion port is designed to synchronously interface over a common, 32-bit data bus with a wide variety of memory and memory-mapped peripheral devices, which include high-speed static RAMs, slower

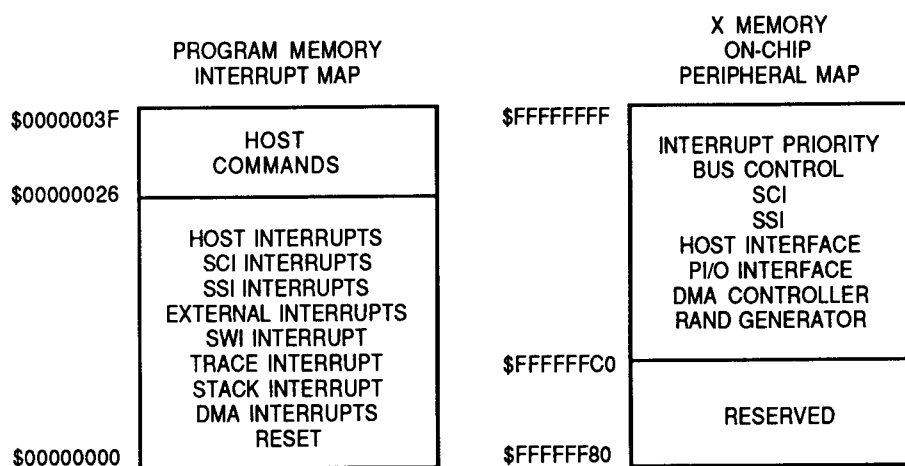


Figure 4. Interrupt and Peripheral Register Memory Maps

memory devices such as dynamic RAMs, and other DSPs and MPUs in a variety of bus configurations. This variety is possible because the expansion bus timing is programmable. The expansion bus timing is controlled by a bus control register (BCR), and the \overline{BS} and \overline{WT} pins. The BCR controls the timing of the bus interface signals, \overline{RD} and \overline{WR} , and the data lines. Each of four memory spaces, X, Y, program, and I/O, has its own 4-bit BCR, which can be programmed for up to 15 wait states (one wait state is equal to a clock period or equivalently one-half of an instruction cycle). In this way, external bus timing can be tailored to match the speed requirements of the different memory spaces. When the number of wait states is not deterministic (e.g., due to external bus arbitration) wait states can be inserted on request by an external arbitrator asserting the \overline{WT} pin. The \overline{PFNS} pin facilitates interfacing with specialized low-cost memories such as video RAMs. After the page size has been selected, \overline{PFNS} will be asserted when the address on the expansion bus crosses the page boundary. Typically, the assertion of \overline{PFNS} will signal external logic to insert wait states until the next base address in the video RAM is selected. Similarly, if the address generated is not sequential, the \overline{PFNS} pin will be asserted. The \overline{BR} pin can be programmed to be either an input or an output. As an input, the DSP96001 acts as a slave and relinquishes the bus after completing the current bus cycle when the \overline{BR} pin is asserted. When the \overline{BR} pin is programmed as an output, the DSP96001 can request the bus mastership. This flexibility facilitates using the DSP96001 in shared memory systems as well as multiprocessor systems.

Direct Memory Access

The dual-channel DMA controller performs all the address storage and effective address calculations necessary to address the DMA source and destination memory locations for two channels. Typically, one channel will be used to DMA through the expansion port; the other channel is used to DMA through the HOST. Generic memory-to-memory, memory-to-peripheral, and peripheral-to-memory transfers can be executed. The DMA controller operates in parallel with other chip resources to minimize overhead due to data or program transfers. Eight registers control each channel: one address, offset, and modifier register for both the source and destination memory locations, a transfer count register, and a channel control/status register. These 16 registers are memory mapped into the X memory space. The programmer's model for each DMA channel is similar to that of the AGU. In fact, the DMA controller unobtrusively shares the same modulo arithmetic units for calculating addresses, which permits the calculations to be done using either linear, modulo, or reverse-carry arithmetic.

General-Purpose I/O (Port B, Port C)

Each port B and C pin may be programmed as a general-purpose I/O pin or as a dedicated, on-chip peripheral pin under software control. A 9-bit port control register is associated with port C and allows each port pin to be programmed individually for one of these two functions. The port control register associated with port B contains

four bits that program each byte of port B. Also associated with each general-purpose port is a data direction register, which programs the direction of each pin, and a data register for data I/O. These registers are read/write, making the use of bit manipulation instructions extremely effective for accessing the ports.

Host Interface

The HOST is a 32-bit, full-duplex, parallel port, which may be connected directly to the data bus of a host processor. The host processor may be any of a number of industry-standard microcomputers or microprocessors, another DSP, or DMA hardware. The DSP96001 HOST has a 32-bit, bidirectional data bus H0-H31 (PB0-PB31) and eight, dedicated control lines ($\overline{HA0}$, $\overline{HA1}$, $\overline{HA2}$, $\overline{HA3}$, $\overline{HR/W}$, \overline{HEN} , \overline{HREQ} , and \overline{HACK}) to control data transfers. The HOST appears to the host processor as a memory-mapped peripheral occupying 16 bytes in the host-processor address space. Separate transmit and receive data registers are double buffered to allow the DSP96001 and host processor to transfer data efficiently at high speed. Host-processor communication with the HOST is accomplished using the standard data move instructions and addressing modes of the host processor. Handshake flags are provided for polled or interrupt-driven data transfers with the host processor. DMA hardware may be used with only the \overline{HREQ} and \overline{HACK} lines to transfer data without host-processor intervention.

One of the most innovative features of the HOST is the host-command feature. With this feature, the host processor can issue vectored interrupt requests to the DSP96001. The host may select any one of 32 DSP96001 interrupt routines to be executed by writing a vector address register in the HOST. This flexibility allows the host programmer to execute up to 32 functions preprogrammed in the DSP96001. For example, if the appropriate interrupt routines are implemented in the DSP96001, HOST interrupts allow the host processor to read or write DSP96001 registers, X, Y, or program memory locations, force exceptions for SSI, SCI, IRQA, and IRQB interrupt routines, and perform control and debugging operations to aid program development.

Serial Communications Interface

The SCI provides a full-duplex port for serial communication to other DSPs, microprocessors, or peripherals such as modems. The communication can be either direct or via RS232C-type lines. This interface uses three dedicated pins: transmit data (TXD), receive data (RXD), and SCI serial clock (SCLK). The SCI supports industry-standard asynchronous bit rates and protocols as well as high-speed (up to 3.33 Mbts/sec), synchronous data transmission. The asynchronous protocols include a multidrop mode for master/slave operation. The SCI consists of separate transmit and receive sections whose operations can be asynchronous with respect to each other. A programmable baud-rate generator is included to generate the transmit and/or receive clocks. An enable bit and interrupt vector have been included so that the baud-rate generator can function as a general-purpose timer when it is not being used by the SCI peripheral.

Synchronous Serial Interface

The SSI is an extremely flexible, full-duplex serial interface, which allows the DSP96001 to communicate with a variety of serial devices at high speed (up to 6.67 Mbits/sec). These devices include one or more industry-standard codecs, other DSPs, microprocessors, and peripherals. The following characteristics of the SSI can be independently defined by the user: the number of bits per word, the protocol or mode, the clock, and the transmit/receive synchronization. Three modes can be selected: normal, network, and on-demand. The normal mode is typically used to interface with devices on a regular or periodic basis. In this mode, the SSI functions with one data word of I/O per frame. The network mode provides time slots, a bit clock, and frame synchronization pulse. The SSI functions with from two to 32 words of I/O per frame in the network mode. This mode is typically used in star or ring time-division-multiplex networks with other DSP96001s and/or codecs. The on-demand mode is a data-driven mode; no timeslots are defined. This mode is intended to be used to interface with devices on a nonperiodic basis.

The clock can be programmed to be continuous or gated. Since the transmitter and receiver sections of the SSI are independent, they may be programmed to be synchronous (use a common clock) or asynchronous with respect to each other. The SSI supports a subset of the Motorola SPI interface in the on-demand mode. The SSI requires three to six pins, depending on the operating mode selected. For example, the six pins can be defined as transmit data, receive data, clock, frame sync, and two select pins so that four codecs can be addressed. A matrix of SSI operating modes versus typical applications is provided in Table 2.

Random-Number Generator

The RAND consists of three, 32-bit read/write registers: a random-number shift register, a bit mask register, and a control/status register. These registers are memory mapped in X memory space. This combination of registers allows the user to define a pseudorandom-number sequence polynomial and a seed. The hardware implements a linear feedback shift register. The output is a 32-bit integer that can be converted to a floating-point number if desired. Random numbers are useful for adding a

dither signal to data or for providing a uniformly distributed random variable for statistically based applications.

On-Chip Emulation

OnCE hardware provides a dedicated serial port for software debugging and production diagnostics. Hardware is provided on-chip to freeze instruction execution, single step, set breakpoints on X:, Y:, and P: addresses, and read data on the system data buses and programming model registers. A history buffer has also been added to read the last five instructions executed. This hardware eliminates the need for cumbersome multiline, high-speed cables that cannot meet the speed requirements for emulating today's high-speed processors. For RAM-based systems using the DSP96001, OnCE makes in-the-field diagnosis and repair a reality.

PROGRAMMING MODEL DESCRIPTION

The DSP96001 programming model is virtually identical to that of the DSP56001 except for the data ALU, which has been enhanced with additional, wider registers. Programming the DSP96001 is therefore a natural extension to programming the DSP56001; a new architecture does not have to be learned. The programmer can view the DSP96001 architecture as three execution units operating in parallel. The three execution units are the data ALU, the AGU, and program controller. The programming model, like that of conventional MPUs, eliminates the need to refer to the detailed chip architecture when programming the DSP96001, because the parallel execution units make the instruction execution pipeline virtually invisible. The programming model is shown in Figure 5 and is described in the following paragraphs.

DATA ALU

The data ALU appears to the programmer as a general-purpose register file consisting of ten, 96-bit, floating-point data registers, which may be alternatively accessed as 30, independent, 32-bit read/write registers. The 96-bit registers, D0-D9 (Dn), are developed by the concatenation of the high, middle, and low registers, Dn.h:Dn.m:Dn.l. The size of the 96-bit register supports

Table 2. SSI Operating Modes

Mode (Protocol)	Serial Clock	Relative TX-RX Timing	Typical Applications
Normal	Continuous	Asynchronous/Synchronous	Asynchronous/Synchronous Codec
Normal	Gated	Asynchronous	Periodic DSP-to-DSP
Normal	Gated	Synchronous	Periodic DSP-to-A/D and DSP-to-D/A
On-Demand	Continuous	Asynchronous	DPS-to-MCU
On-Demand	Continuous	Synchronous	P-to-S and S-to-P Conversion
On-Demand	Gated	Asynchronous	DSP-to-DSP
On-Demand	Gated	Synchronous	DSP-to-SPI Peripherals
Network	Continuous	Asynchronous/Synchronous	TDM Codecs/DSP Networks

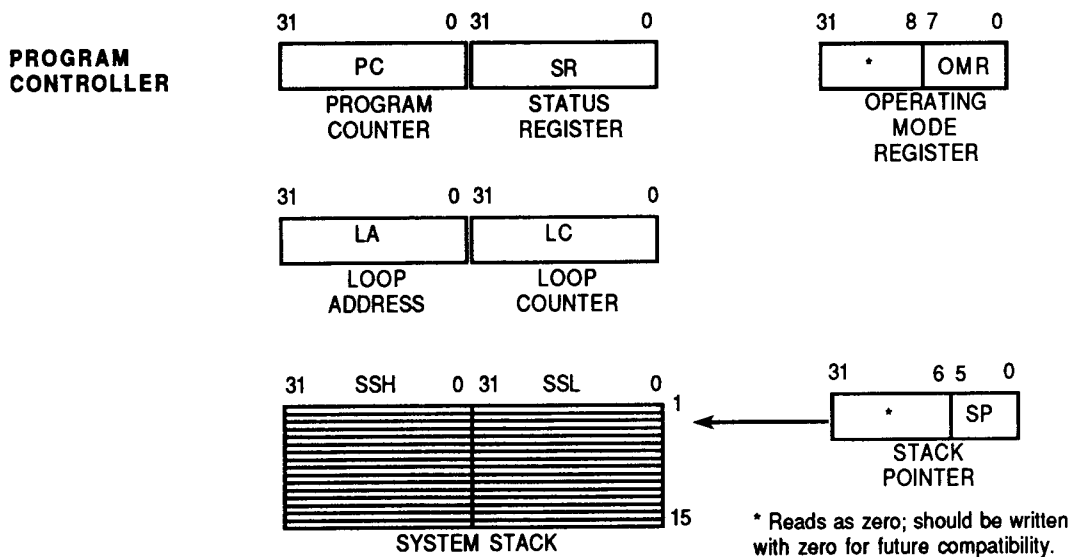
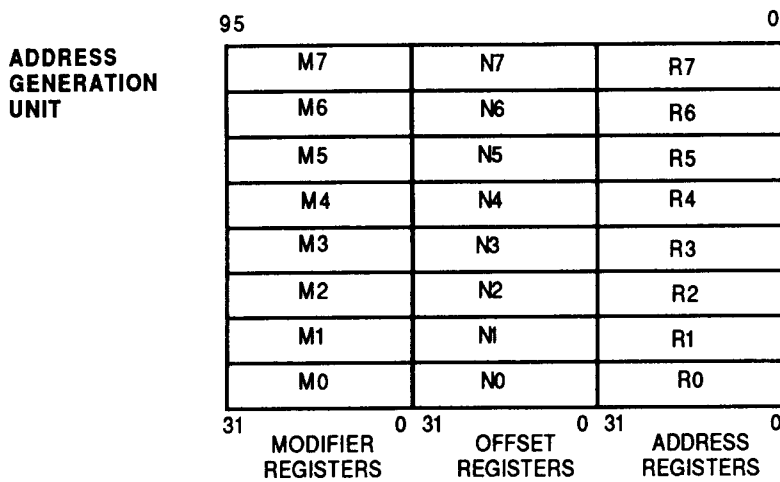
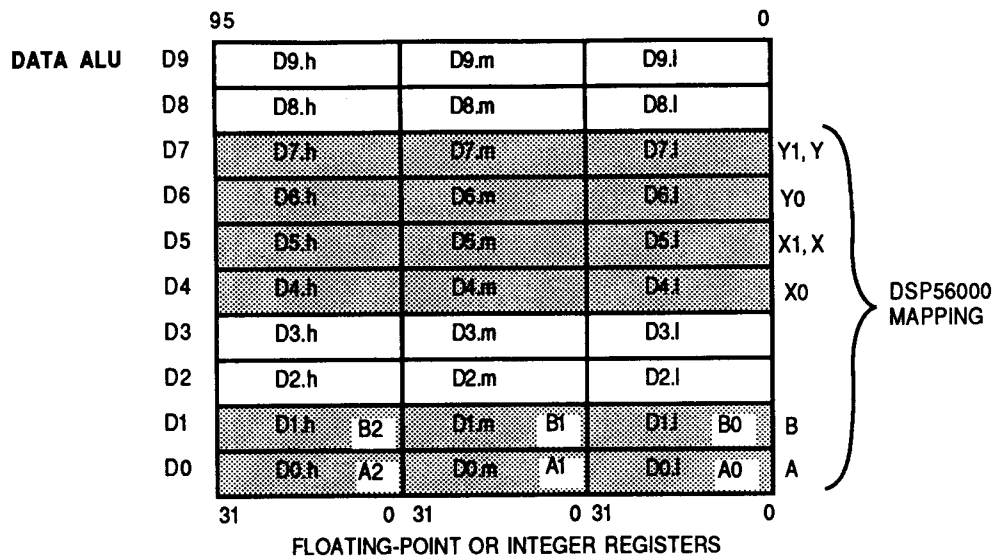


Figure 5. User Programming Model

extended-precision and mixed-precision arithmetic and allows extra accuracy for intermediate calculations and transcendental functions. For example, when the Dn registers are the destination of floating-point instructions, a conversion is automatically and invisibly done to represent the data in double-extended-precision format. When the Dn registers are the destination of nonfloating-point instructions, the Dn.l registers are written if the result is 32 bits, and Dn.m:Dn.l registers are written if the result is 64 bits. The remaining registers are not affected. Compatibility with the DSP56001 data ALU registers is achieved as shown in Figure 5.

Registers D0-D7 are considered as general-purpose registers. These registers can be used as source or destinations in general or as source and destination operands in the same instruction: i.e., source for the current instruction and destination for loading the source operands for the next instruction. They may also be read back to the appropriate data bus to implement memory delay operations and save/restore operations for interrupt service routines.

D8 and D9 are special-purpose registers that can be read or written over XDB or YDB. These registers can be used to hold source operands for data ALU operations, act as data pipeline registers, or as registers for holding temporary variables or constants that are frequently used for implementing high-level languages. D8 and D9 are not used to hold the results of data ALU operations.

ADDRESS GENERATION UNIT

The programmer's model for the AGU consists of three sets of 32-bit read/write registers called address registers, offset registers, and modifier registers. They provide all the registers necessary to generate address-register indirect effective addresses.

Address-Register Files (R0-R3 and R4-R7)

The eight address registers, R0-R7, are 32 bits wide and may contain addresses or general-purpose data. The 32-bit address in a selected address register is used in the calculation of the effective address of an operand. When supporting concurrent, parallel X and Y memory moves, the address registers must be viewed as two separate files, R0-R3 and R4-R7, one file for each bus. The content of an Rn may point to data directly and/or may be pre- or post-updated according to the selected addressing mode. Modifier registers are always used to specify the type of arithmetic to be used if the associated Rn is updated. Offset registers are used if the update by offset addressing mode is specified. The address-register modification is performed by one of the two modulo arithmetic units.

Offset-Register Files (N0-N3 and N4-N7)

The eight offset registers, N0-N7, are 32 bits wide and may contain offset values used to increment and decrement address registers in indexed, address-register update calculations, or they may be used for 32-bit, general-purpose storage. For example, the contents of an offset register may be used to step through a table at some rate (e.g., five locations per step for waveform generation),

or may specify the offset into a table or the base of a table for indexed addressing. Each Rn has its own Nn associated with it.

Modifier-Register Files (M0-M3 and M4-M7)

The eight modifier registers, M0-M7, are 32 bits wide. The content of Mn defines the type of address arithmetic to be performed for address update calculations. The AGUs support linear, modulo, and reverse-carry arithmetic types for all address-register indirect addressing modes. For the case of modulo arithmetic, the content of Mn also specifies the modulus. Each Rn has its own Mn associated with it. Each modifier register is set to \$FFFFFFF on processor reset, which specifies linear arithmetic as the default type for address-register update calculations.

PROGRAM CONTROLLER

The program controller has the standard program flow resources such as the PC, status register (SR), operating mode register (OMR), stack pointer (SP), and a 15-level by 64-bit system stack memory. The DSP96001 program controller also features a loop address (LA) register and a loop counter (LC) register, which are dedicated to supporting the hardware DO loop and repeat instructions. With the exception of the PC, all registers are read/write to facilitate system debug.

Program Counter

The 32-bit PC register contains the address of the next location to be fetched from program memory space. This special-purpose address register is pushed on the stack when program looping is initiated, during long interrupts, or when a jump-to-subroutine instruction is performed. The PC can address 4,294,967,296 locations in program memory space.

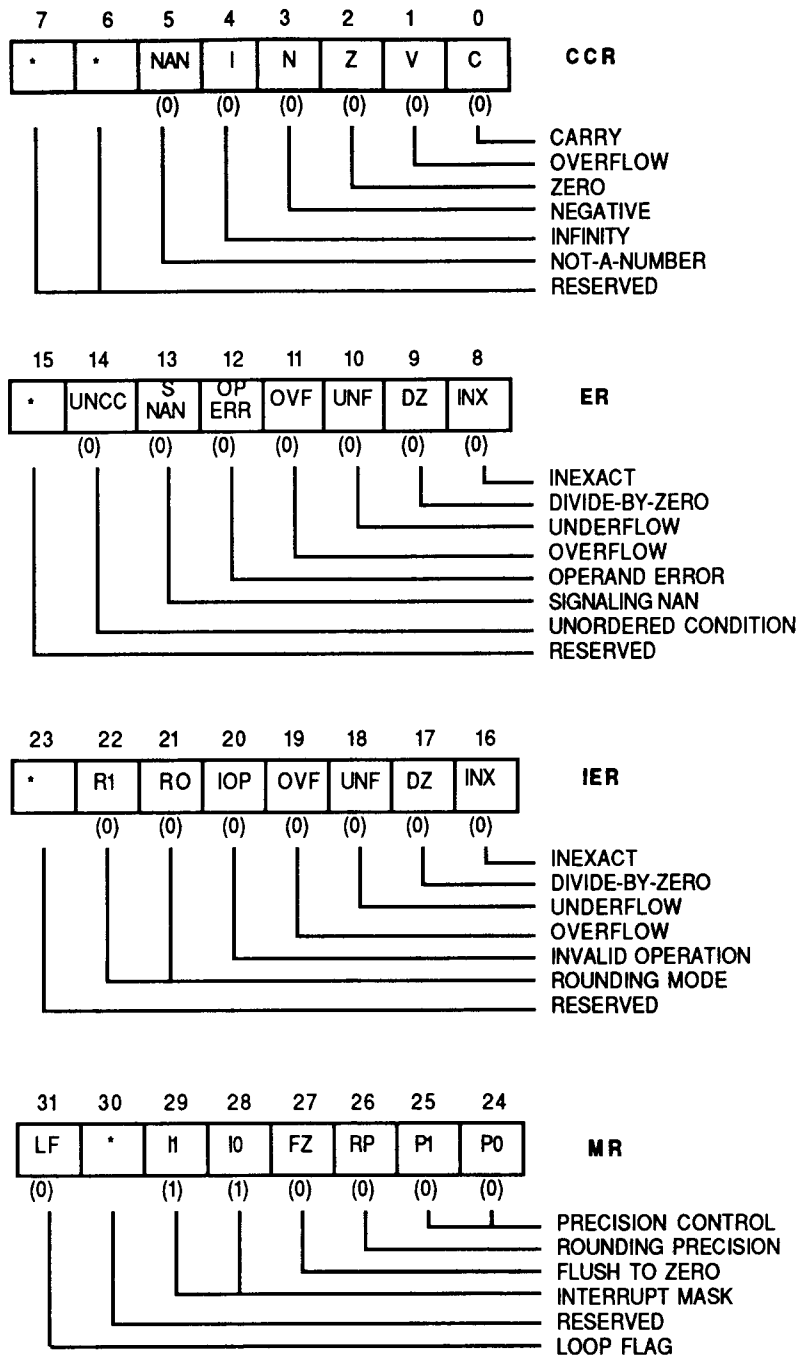
Status Register

The SR is a 32-bit register consisting of an 8-bit condition code register (CCR), an 8-bit exception register (ER), an 8-bit IEEE exception register (IER), and an 8-bit mode register (MR). SR is pushed on the stack when program looping is initialized, during long interrupts, or when a jump-to-subroutine instruction is performed (see Figure 6).

The contents of CCR reflect the status produced by each data ALU instruction. The CCR bits are affected by data ALU operations and by instructions that directly reference the CCR register — namely, ORI and ANDI. The CCR bits are not affected by transfers over the XDB and YDB.

The contents of ER reflect the exceptions produced as a result of the execution of the last instruction. The ER bits are affected by processor reset, by instructions which directly reference the ER register (namely, ORI and ANDI), and by the data ALU floating-point operations.

The IER contains the five IEEE exception flags and rounding mode control. The IER bits are affected by processor reset, by instructions directly referencing the IER register (namely, ORI and ANDI), and by the data ALU floating-point operations. The IEEE exception bits are sticky in that they remain set until cleared by the user.



NOTE: Number in parentheses indicates status after poweron reset.

Figure 6. Status Register Format

MR is a special-purpose control register that defines the current system state of the processor. The MR bits are affected by processor reset, exception processing, the DO, ENDDO, RTI, ILLEGAL, and SWI instructions, and by instructions directly referencing the MR register — namely, ORI and ANDI.

Loop Counter

The LC is a special-purpose 32-bit counter used to specify the number of times a hardware program loop is to be repeated. This register is stacked by a DO instruction and unstacked by end-of-loop processing or by execution

of an ENDDO instruction. The LC is decremented each time the loop is executed and may be read under program control, which allows the number of times a loop has been executed to be monitored during execution. The LC is also used in the REP instruction.

Loop Address Register

The content of the 32-bit LA register indicates the location of the last instruction word in a program loop. This register is stacked by a DO instruction and unstacked by end-of-loop processing or by execution of an ENDDO instruction.

System Stack

The SS is a separate internal memory, which stores the PC and SR for subroutine calls and long interrupts. The SS will also store the LC and LA registers in addition to the PC and SR registers for program looping. The SS is in stack memory space, and its address is always inherent and implied by the current instruction. The SS memory is 64 bits wide and 15 locations deep. The SS is divided into two separately addressable banks: system stack high (SSH) and system stack low (SSL), each 32 bits wide. SSH stores the PC or LA contents; SSL stores the SR or LC contents. Being able to address SSL and SSH facilitates setting up software stacks in memory.

When a subroutine call or long interrupt occurs, the contents of the PC and SR are pushed on the location in the SS. When a return from subroutine occurs, the contents of the top location in the SS are pulled to the PC, but not to the SR. When a return from interrupt occurs, the contents of the top location in the SS are pulled to both the PC and SR.

The interrupt subsystem of the DSP96001 is vector based and prioritized. Interrupt vectors point to two consecutive locations in program memory. If one of the two words fetched by the interrupt controller is a jump-to-subroutine instruction, a long interrupt routine is formed, and a context switch is performed using the stack. If neither interrupt instruction word causes a change of control flow,

then the two interrupt instruction words fetched constitute a fast interrupt routine; the two instruction words are executed in line without a context switch being performed. The fast interrupt routine provides exception processing with no context switching overhead. This mechanism is commonly used to move data between memory and an I/O device. Up to 2.2 million interrupts per second can be processed using this mechanism.

The SS is also used to implement no-overhead, nested hardware DO loops. When the DO instruction is executed, the address of the first instruction in the loop and the contents of SR, LA, and LC prior to the start of the loop are saved on the stack, which allows nesting of DO loops.

Stack Pointer

The SP register is a 32-bit register that indicates the location of the top of the SS and the status of the stack (underflow, empty, full, and overflow conditions). The SP is referenced implicitly by some instructions (DO, REP, JSR, RTI, etc.) or directly by the MOVEC instruction. The SP register format is shown in Figure 7.

Operating Mode Register

The OMR is a 32-bit register that defines the current operating mode of the processor: i.e., the memory maps for program and data memories as well as the startup procedure. The OMR bits are only affected by processor reset and by instructions directly referencing the OMR. During processor reset, the chip operating mode bits, MA and MB, will be loaded from the external mode select pins, MODA and MODB. The data ROM enable bit is cleared, disabling the X and Y on-chip lookup table ROMs. The external clock bit is set if an external clock oscillator is used. When the external clock bit is set, exit from the stop state takes only 25 clock cycles. Figure 3 shows the effect of the OMR on the DSP96001 memory maps. The operating mode register format is shown in Figure 8. Table 3 summarizes the DSP96001 operating modes. Tables 4 and 5 show the program and data memory spaces.

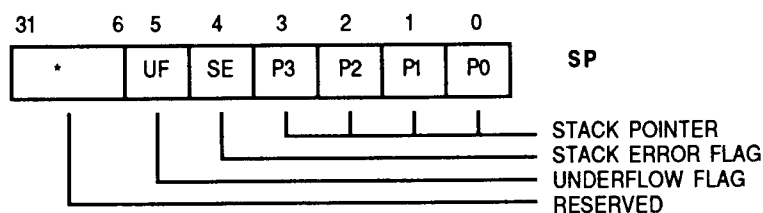


Figure 7. Stack Pointer Format

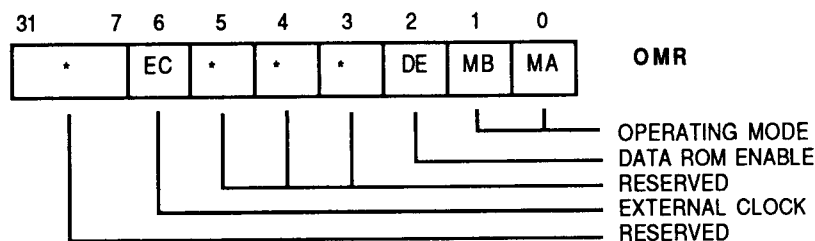


Figure 8. Operating Mode Register Format

Table 3. Operating Mode Summary

Operating Mode	MB	MA	Description
0	0	0	PRAM enabled; reset at \$00000000 (internal).
1	0	1	Special bootstrap mode, after PRAM loading mode 2 is automatically selected.
2	1	0	PRAM enabled; reset at \$E0000000 (external).
3	1	1	PRAM disabled; reset at \$00000000 (external).

Table 4. Program Memory Space

Operating Mode	MB	MA	Description
0 and 2	X	0	Internal RAM: \$00000000-\$000001FF External: \$00000200-\$FFFFFFF
3	1	1	External: \$00000000-\$FFFFFFF

Table 5. Program Memory Space

Data ROM Enable	X Memory Space Map	Y Memory Space Map
0	Internal RAM: \$00000000-\$000001FF External: \$00000200-\$FFFFFF7F On-chip peripherals: \$FFFFFF80-\$FFFFFFF	Internal RAM: \$00000000-\$000001FF External: \$00000200-\$FFFFFFF
1	Internal RAM: \$00000000-\$000001FF Internal ROM: \$00000200-\$000003FF External: \$00000400-\$FFFFFF7F On-chip peripherals: \$FFFFFF80-\$FFFFFFF	Internal RAM: \$00000000-\$000001FF Internal ROM: \$00000200-\$000003FF External: \$00000400-\$FFFFFFF

INSTRUCTION SET SUMMARY

The DSP96001 instruction set has been designed to be as orthogonal as possible to allow flexible, independent, concurrent control of the data ALU, AGU, and program controller execution units during each instruction cycle. This instruction-set design maximizes throughput, minimizes program storage requirements, and enhances the efficiency of high-level language compilers. The instruction set is divided into the following groups:

- Floating-Point Arithmetic
- Fixed-Point Arithmetic

- Logical Bit Manipulation
- Loop
- Move
- Program Control

INSTRUCTION FORMAT

Instructions are one or two words in length. The instruction and its length are specified by the first word of the instruction. The second word may contain an absolute address or immediate data. When an instruction is more than one word in length, an additional instruction execution cycle may be required. The assembly language source code for a floating-point FMPY//FADD one-word instruction is shown below. The source code is organized into six fields:

Opcode
FMPY

Operands
D4, D5, D0

Opcode
FADD

Operands
D0, D1

Parallel Data Move
X: (R0) + ,D4

Parallel Data Move
Y: (R4) + ,D5

The opcode fields typically indicate the data ALU operation to be performed; it may also specify a move, address generation, or program control operation. At least one opcode field must always be included in the source code. The operand fields specify the operands to be used by the opcode immediately to the left. Up to two parallel data moves are defined in many instructions. The memory space qualifiers, X:, Y:, P:, and L: (long memory space), indicate which memory space is being referenced. Parallel move operations are important because they allow new operands to be prefetched for use in the next instruction and allow results from the previous instruction to be saved.

The DSP96001 allows parallel processing by the data ALU, AGU, and program controller. For example, in the previous instruction word, the DSP96001 will perform the designated ALU operations (data ALU), the data transfers specified with address register updates (AGU), and will also decode the next instruction and fetch an instruction word from program memory (program controller), all in one instruction cycle. In addition, the program controller may be processing an active hardware DO loop.

Floating-Point Arithmetic Instructions

All floating-point arithmetic instructions operate on the 96-bit data ALU registers, which means the on-chip data buses are free for parallel move operations. If the floating-point instruction specifies one parallel move, either floating-point or fixed-point data types can be moved. If a floating-point instruction specifies two parallel moves,

only floating-point data can be moved. Floating-point instructions affect the condition codes in the IER and ER registers with the exception of the transfer conditionally instruction, FTcc, which does not affect any condition codes. Floating-point instructions execute in a single cycle in the default mode except when denormalized numbers are detected. Floating-point instructions always execute in a single cycle in the flush-to-zero mode with the possible exception of the FMAC instruction. The FMAC instruction takes two cycles if the contents of the destination are used in the following instruction, regardless of the mode.

Multi-operand floating-point instructions are as follows:

FADD	Float Add
FADDR	Float Add and Round
FADDSUB	Float Add, Float Subtract
FADDSUBR	Float Add and Round, Float Subtract and Round
FCMP	Float Compare
FCMPM	Float Compare Magnitude
FCOPYS	Float Copy Sign
FDIVI	Float Divide Iteration*
FGETEXP	Float Extract Exponent
FGETMAN	Float Extract Mantissa
FINT	Float Extract Integer with Default Round
FLOOR	Float Extract Integer with Round to Minus Infinity
FMAC	Float Multiply with Accumulation
FMACR	Float Multiply with Accumulation and Round
FMPY	Float Multiply
FMPYR	Float Multiply and Round
FMPY//FADD	Float Multiply, Float Add
FMPY//FADDR	Float Multiply, Float Add and Round
FMPY//FADDSUB	Float Multiply, Float Add, Float Subtract
FMPY//FADDSUBR	Float Multiply, Float Add and Round, Float Subtract and Round
FMPY//FSUB	Float Multiply, Float Subtract
FMPY//FSUBR	Float Multiply, Float Subtract and Round
FSCALE	Float Scale
FSCALER	Float Scale and Round
FSUB	Float Subtract
FSUBR	Float Subtract and Round
FTFR	Float Transfer Data ALU Register
FTcc	Float Transfer Data ALU Register Conditionally*

*These instructions do not allow parallel moves.

Single-operand floating-point instructions are as follows:

FABS	Float Absolute Value
FCLR	Float Clear
FLOAT	Integer to Float Convert
FLOATU	Unsigned Integer to Float Convert
FNEG	Float Negate
FSGL	Float Convert to Single Precision
FTST	Float Test
INT	Float to Integer Convert
INTRZ	Float to Integer Convert with Round to Zero

The IEEE 754 standard divide, remainder, square root, and convert binary to/from decimal operations are provided by Motorola in the form of IEEE conformant software.

Fixed-Point Arithmetic Instructions

These arithmetic instructions perform all of the fixed-point arithmetic operations within the data ALU. With the exception of the TFR and Tcc instructions (which do not affect any condition codes), fixed-point instructions may affect the condition codes in the CCR. Fixed-point arithmetic instructions are register based so that the data ALU operation indicated by the instruction does not use the X bus, Y bus, or G bus. Parallel data movement over these on-chip buses is allowed during most data ALU operations. The data format for two parallel moves is fixed point; if only one parallel move is specified, either floating-point or fixed-point data can be moved as specified by the destination register. D0.m,l-D7.m,l are the destination register alternatives for these instructions. Fixed-point arithmetic instructions execute in one instruction cycle.

Multi-operand fixed-point instructions are as follows:

ADC	Add with Carry
ADD	Add
ASL	Arithmetic Shift Left (Multiple)
ASR	Arithmetic Shift Right (Multiple)
CMP	Compare
CMPA	Compare Address*
CMPM	Compare Magnitude
MPY	Signed Multiply
MPYU	Unsigned Multiply*
SUB	Subtract
SBC	Subtract with Carry
Tcc	Transfer Conditionally*
TFR	Transfer Data ALU Register

*These instructions do not allow parallel moves.

The CMPM affects the condition code bits according to the results of the subtraction of the absolute values of two operands. This instruction, together with Tcc, is useful in determining maximum and minimum values in blocks of data.

Single-operand fixed-point instructions are as follows:

ABS	Absolute Value
ASL	Arithmetic Shift Left
ASR	Arithmetic Shift Right
CLR	Clear
DEC	Decrement
INC	Increment
NEG	Negate
TST	Test
TSTA	Test Address*

*These instructions do not allow parallel moves.

Logical Instructions

The logical instructions use only the resources internal to the data ALU to perform all logical operations affecting the CCR bits. Logical instructions are data ALU register based. Optional data transfers may be specified with most logical instructions, which allows for parallel data movement over the on-chip buses during a data ALU logical operation. New data can be prefetched for use in the following instructions, and results calculated in previous instructions can be stored. For logical instructions, the data format for parallel moves is fixed point if two moves

are specified; if a single parallel move is specified, either fixed-point or floating-point data may be moved, depending on the destination address specified. These instructions execute in one instruction cycle. The destination is one of D0.I-D7.I, except for ANDI or ORI, whose destination is one of the status registers.

AND	Logical AND
ANDI	AND Immediate with Control Register*
EOR	Logical Exclusive OR
LSL	Logical Shift Left
LSR	Logical Shift Right
NOT	Logical Complement
OR	Logical Inclusive OR
ORI	OR Immediate with Control Register*
ROL	Rotate Left
ROR	Rotate Right

*These instructions do not allow parallel moves.

Bit Manipulation Instructions

The bit manipulation instructions test the state of any single bit in a memory location or program model register and then optionally sets, clears, or inverts the bit. The carry bit of the CCR will contain the result of the bit test. Parallel moves are not allowed with any of these instructions.

BCLR	Bit Test and Clear
BSET	Bit Test and Set
BCHG	Bit Test and Change
BTST	Bit Test on Memory

Loop Instructions

The DO and ENDDO instructions make writing straight-line code practically unnecessary. The DO instruction sets up a hardware loop by initiating a program loop, setting up looping parameters, and then cleans up the SS when terminating a loop. Initialization includes saving registers LA and LC on the SS so that program loops can be nested. The address of the first instruction in a program loop is also saved on the stack to allow no-overhead looping. Single-instruction DO loops can be implemented. DO loops are interruptible. An indirect address can be used to specify a dynamic loop count in the DO instruction, which facilitates parameter passing. The ENDDO instruction is used to prematurely terminate the current DO loop and to clean up the stack. To maximize throughput, the REP instruction repeats the next instruction without re-fetching the instruction. Because the instruction repeated is not re-fetched, a REP operation is not interruptible. An interruptible repeat instruction can be implemented using a single-instruction DO loop. These instructions do not allow parallel moves.

DO	Start Hardware Loop
ENDO	End Current DO Loop
REP	Repeat Next Instruction

Move Instructions

The MOVE instructions perform data movement over XDB, YDB, GDB, and PDB. MOVE instructions do not affect the status registers. The FMOVE and MOVE instructions provide all the floating-point and fixed-point,

respectively, parallel move operations and can be considered to be data ALU No Ops with parallel moves.

FMOVE	Float Move
LEA	Load Effective Address
MOVE	Move Data
MOVEC	Move Control Register
MOVEI	Move Immediate
MOVEM	Move Program Memory
MOVEP	Move Peripheral Data

Program Control Instructions

The program control instructions include jumps, conditional jumps, branches, conditional branches, and other instructions affecting the PC and SS. Branch instructions allow PC relative offsets needed for writing position-independent code. Program control instructions may affect the CCR bits as specified in the instruction. Optional parallel data transfers over the on-chip buses are not allowed during the execution of program control instructions. Execution of the STOP and WAIT instructions place the DSP96001 in low-power states. All processor activity is suspended, and the oscillator is gated off after a STOP instruction has been executed. When the WAIT instruction is executed, internal processing is halted; however, the on-chip peripherals and oscillator remain active. The processor waits for an interrupt or processor reset to exit the stop or wait states. The following are the program control instructions.

BBSET	Branch if Bit Set
BBCLR	Branch if Bit Clear
BRA	Branch Always
BRcc	Branch Conditionally
BScc	Branch to Subroutine Conditionally
BSR	Branch to Subroutine
BSCLR	Branch to Subroutine if Bit Clear
BSSET	Branch to Subroutine if Bit Set
FBcc	Float Branch Conditionally
FBScc	Float Branch to Subroutine Conditionally
ILLEGAL	Illegal Instruction Interrupt
Jcc	Jump Conditionally
JCLR	Jump if Bit Clear
JMP	Jump
JScc	Jump to Subroutine Conditionally
JSCLR	Jump to Subroutine if Bit Clear
JSET	Jump if Bit Set
JSR	Jump to Subroutine
JSSET	Jump to Subroutine if Bit Set
NOP	No Operation
RESET	Reset On-Chip Peripheral Devices
RTI	Return from Interrupt
RTS	Return from Subroutine
STOP	Stop Processing (Low Power Standby)
SWI	Software Interrupt
WAIT	Wait for Interrupt (Low Power Standby)

ADDRESSING MODES

The addressing modes are grouped into three categories: register direct, register indirect, and special. These addressing modes are summarized in Table 6. All address calculations are performed in the AGU to minimize execution time and loop overhead. Addressing modes specify whether the operand is in a register, memory, or encoded in the instruction (as immediate data).

Table 6. Address Modes Summary

Addressing Mode	Modifier MMMM	Memory/Registers Referenced								
		S	C	D	A	P	X	Y	L	XY
Register Direct										
Data or Control Register	No	X	X	X						
Address Register	No				X					
Address Modifier Register	No				X					
Address Offset Register	No				X					
Address Register Indirect										
No Update	Yes					X	X	X	X	X
Postincrement by 1	Yes					X	X	X	X	X
Postdecrement by 1	Yes					X	X	X	X	X
Postincrement by Offset Nn	Yes					X	X	X	X	X
Postdecrement by Offset Nn	Yes					X	X	X	X	X
Indexed by Offset Mn	Yes					X	X	X	X	
Indexed by Displacement	Yes						X	X	X	
Predecrement by 1	Yes					X	X	X	X	
Special										
Immediate Data	No					X				
Absolute Address	No					X	X	X	X	
Immediate Short Data	No					X				
Short Jump Address	No					X				
PC Relative	No					X				
I/O Short Address	No						X	X		
Implicit	No	X	X			X				

Where:

- MMMM = Address Modifier
- S = Stack Reference
- C = Program Controller Register Reference
- D = Data ALU Register Reference
- A = Address ALU Register Reference
- P = Program Memory Reference
- X = X Memory Reference
- Y = Y Memory Reference
- L = L Memory Reference
- XY = XY Memory Reference

The register direct addressing mode can be subclassified according to the specific register addressed. The data registers include D0-D9. The control registers include SR, OMR, SP, SSH, SSL, LA, LC, CCR, IER, ER, and MR.

Address register indirect modes use an address register (Rn) to point to locations in memory. Except in the indexed-by-offset or displacement modes, the content of Rn forms the effective address for address indirect addressing and can be predecremented or postincremented. For indexed modes, the effective address is Rn + Nn if an offset is used, or Rn + \$xxxxxxx if a displacement is used and the content of Rn is not changed. In all cases, the update is done according to one of three types of arithmetic selected by the corresponding Mn register. If a mode using an offset is specified, an offset register (Nn) is also used for the update. The Nn and Mn registers are assigned to the Rn with the same n. Thus, the assigned register sets are the triplets — M0;N0;R0, M1;N1;R1, M2;N2;R2, M3;N3;R3, M4;N4;R4, M5;N5;R5, M6;N6;R6, and M7;N7;R7. This structure is unique and extremely powerful in general, and particularly powerful in setting up DSP-oriented data structures. All address register indirect modes use at least one triplet, and the

X/Y memory reference uses two triplets, one for X memory space and one for Y memory space.

The special addressing modes include immediate and absolute modes, PC relative for P: space, and implied references to the PC, SS, and program memory. PC relative modes are indexed by absolute and by Rn.

Although the indexed-by-displacement indirect addressing mode is only directly available for X:, Y:, and XY: memory spaces, it can be used indirectly for P: space by using the LEA instruction to load a pointer. Similarly, PC relative addressing can be used to address operands in X:, Y:, and XY: spaces by using the LEA instruction to load a pointer.

Address Modifiers (Mn)

The address modifiers allow the AGU to support linear, modulo, and reverse-carry address arithmetic for all address register indirect modes. These special address arithmetic types allow the creation of data structures in memory for FIFOs (queues), delay lines, circular buffers, stacks, and bit-reversed FFT buffers. Data is manipulated by updating address registers rather than moving large blocks of data. The content of the address modifier register, Mn, defines the type of address arithmetic to be

performed for addressing mode calculations. For the case of modulo arithmetic, the content of Mn also specifies the modulus. The three types of arithmetic are discussed below.

Linear Arithmetic (Mn = \$FFFFFFF)

The address calculation is performed using normal 32-bit, twos complement linear arithmetic. A 32-bit offset, Nn, may be used in the address calculations. The range of values may be considered as signed (Nn from -2,147,483,648 to +2,147,483,647) or unsigned (Nn from 0 to +4,294,967,296).

Modulo Arithmetic (Mn = modulus - 1)

The address calculation is performed modulo M, where M ranges from 2 to +16,777,216 (i.e., 24 bits). Modulo M arithmetic causes the address register value to remain within an address range of size M defined by a lower and upper address boundary. The value M - 1 is stored in the modifier register, Mn. The lower boundary (base address) value must have zeros in the k LSBs, where $2^k \geq M$, and therefore must be a multiple of 2^k . The minimum value of 2^k , which meets this criteria, is called the block size. The upper boundary is the lower boundary plus the modulus size minus one (base address plus Mn). For example, to create a circular buffer of 21 locations, M is 21 and the lower address boundary must have its five least significant bits equal to zero ($2^k \geq 21$, thus $k \geq 5$). The Mn register is loaded with the value M - 1 (i.e., 20). The lower boundary may be chosen as 0, 32, 64, 96, 128, 160, etc. since these addresses all have at least five least significant zeros. The upper boundary of the buffer is then the lower boundary plus 20. The address pointer is not required to start at the lower address boundary nor to end on the upper address boundary; however, it must initially point anywhere within the defined modulo address range. Neither the lower nor the upper boundary of the modulo region is stored; only the size of the modulo region is stored in Mn. Assuming the (Rn) + indirect addressing mode, if the address register pointer increments past the upper boundary of the buffer (base address plus M - 1), it will wrap around to the lower boundary (base address). Alternatively, assuming the (Rn) - indirect addressing mode, if the address decrements past the lower boundary (base address), it will wrap around to the upper boundary (base address plus M - 1).

If the (Rn) + Nn addressing mode is used in the address calculations, the 32-bit value |Nn| must be less than or equal to M for proper modulo addressing because a single modulo wraparound is detected. If |Nn| > M, the result is data dependent and unpredictable, except for the special case where $Nn = L * 2^k$, a multiple of the block size, 2^k , where L is a positive integer. The offset Nn must be a positive twos complement integer. For this case, the pointer Rn will be incremented using linear arithmetic to the same relative address L blocks forward in memory. Similarly, for the (Rn) - Nn addressing mode, the pointer

Rn will be decremented, using linear arithmetic, L blocks in memory. For the normal case where $|Nn| \leq M$, the modulo arithmetic unit will automatically wrap the address pointer around by the required amount. This type of address modification is useful in creating circular buffers for FIFOs (queues), delay lines, and sample buffers up to 16,777,216 words long; it is also useful for decimation, interpolation, and waveform generation. The special case of $(Rn) \pm Nn$ with $Nn = L * 2^k$ is useful for performing the same algorithm on multiple buffers — for example, implementing a bank of parallel filters.

Reverse-Carry Arithmetic (Mn = \$00000000)

This arithmetic is a special case of modulo addressing in that a buffer is established with boundary conditions. Within the buffer, the address update arithmetic is performed by propagating the carry in the reverse direction — that is, from the most significant bit to the least significant bit. This method is equivalent to bit-reversing (i.e., redefining the most significant bit as the least significant bit and the next most significant bit as bit 1, etc.) the content of Rn and the offset value, Nn, adding normally, and then bit-reversing the result. If the (Rn) + Nn addressing mode is used with this address modifier type, and if Nn contains the value 2^{k-1} , then postincrementing by +Nn is equivalent to bit-reversing the k LSBs of Rn, incrementing Rn by 1, and bit-reversing the k LSBs of Rn. This address arithmetic is useful for performing 2^k point FFTs, which allows bit-reversed addressing for FFTs having up to 4,294,967,296 points.

As an example, consider a 1,024 point complex FFT ($k = 10$) with real data stored in X memory and imaginary data stored in Y memory. Then Nn would contain the value 512 and postincrementing by +Nn would generate the address sequence 0, 512, 256, 768, 128, 640, . . . This sequence is the scrambled FFT data order for sequential frequency points from 0 to 2 pi. The lower boundary (base address) of the reverse carry buffer must have at least k zeros. Therefore, the reverse-carry modifier works when the base address of the FFT data buffer is a multiple of 2^k , such as 0, 1024, 2048, 3072, in our example. The use of addressing modes other than postincrement by +Nn is possible, but it may not provide a useful result.

APPLICATION DEVELOPMENT TOOLS

The application development tools include a macro cross assembler, a linker/librarian, a C compiler, an application development system, and the DSP electronic bulletin board.

SOFTWARE

Development software support products run on any of the following platforms: IBM™ PC, Macintosh™ II, VAX™, SUN-3™ workstations. The software, written in C, consists of an assembler, linker, and simulator, which are

IBM is a trademark of International Business Machines
Macintosh is a trademark of Apple Computer, Inc.
VAX is a trademark of Digital Equipment Corporation
SUN-3 is a trademark of Sun Microsystems, Inc.

marketed as an integrated product, labeled DSP9600CLASx, and a full Kernighan and Richie C compiler, labeled DSP96KCCx. The ordering information is as follows:

Host Platform	Operating System	Order Number
IBM PC	DOS 2.x, 3.x	DSP9600CLASA DSP96KCCA
Macintosh II	MAC OS 4.1	DSP9600CLASB DSP96KCCB
SUN-3	BSD 4.2	DSP9600CLASC DSP96KCCC
VAX	VMS 4.5	DSP9600CLASD DSP96KCCD
VAX	BSD 4.2	DSP9600CLASE DSP96KCC E

Macro Cross Assembler

The full-featured macro cross assembler translates one or more source fields containing DSP instruction mnemonics, operands, and assembler directives into relocatable object modules that are relocated and linked by the DSP96000 linker operating in the relocatable mode. In the absolute mode, the assembler will generate absolute load files. The assembler recognizes the full instruction set and all addressing modes of the DSP96000 Family, which includes support for separate X and Y memory spaces and data transfer operations in parallel with the data ALU operations.

Because the DSP96001 floating-point data ALU is substantially different from the DSP56001 fixed-point data ALU, several assembler options are provided to allow source-code compatibility across both families. These options provide the ability to assemble and execute the same source code in fixed point on the DSP56001 and in floating point on the DSP96001. During normal assembly, the DSP96000 assembler accepts DSP96000 instruction mnemonics and register names. The emulation option, `-E<outfil>`, assembles DSP56000 source code to DSP96000 source code, which emulates the DSP56000 on a bit-for-bit basis. For example, a DSP56000 instruction ASL will be assembled so that bit 23 is actually shifted into the carry bit instead of bit 31, as on the DSP96000. In this mode, the DSP96000 assembler will accept the same DSP56000 source code as the DSP56000 assembler but will not accept any DSP96000 source code. This option is useful when the target machine is the fixed-point DSP56000 but it is desired to first develop the algorithm using floating-point arithmetic. The `<outfil>` option will generate a DSP96000 source-code file generated by the emulation option. The floating-point option `F<outfil>`, assembles DSP56000 source code to DSP96000 source code in order to run the routine in floating point. In this option, the target machine is the DSP96000, and the DSP96000 assembler accepts both DSP56000 and DSP96000 source code. In this case, the DSP56000 instruction ASL will be assembled to the DSP96000 instruction ASL where bit 31 is shifted into the carry bit. The `<outfil>` option will generate a DSP96000 source-code file before the assembly. This source-code file may be

edited and reassembled to aid program porting. The DSP96000 assembler options are summarized in Table 7.

Table 7. DSP96000 Assembler Options

Option	Accepts DSP56000 Source	Accepts DSP96000 Source	Target Word Size
None	No	Yes	32
-E	Yes	No	24
-F	Yes	Yes	32

This assembler program offers the usual complement of features found in modern assemblers, such as conditional assembly, file inclusion, nested macros with support for macro libraries (via the MACLIB directive), and modular programming constructs ordinarily found only in higher level languages.

The unique architecture and parallel operation of the DSP96000 Family demands special-purpose facilities and programming aids that this assembler readily provides. These aids include built-in functions for common transcendental math computations such as sine, cosine, log, and square root functions; arbitrary expressions and modulo operations; and directives to define modulo and bit-reversed data buffers. Moreover, the assembler incorporates extensive error checking and reporting to indicate programming violations peculiar to the DSP environment or stemming from the advanced features of the DSP96001. These offenses include errors for improper nesting of hardware DO loops, improper address boundaries for modulo and bit-reversed buffers, and violations of instruction pipeline restrictions.

The assembler program also generates source-code listings including numbered source lines, optional titles and subtitles, optional instruction cycle counts, symbol table and cross-reference listings, and memory-use reports.

Features of the macro cross assembler program are as follows:

- Produces relocatable modules compatible with the linker program (LNK96000) in the relocatable mode
- Produces absolute load files compatible with the simulator program (SIM96000) in the absolute mode
- Supports porting DSP56000 code to the DSP96000
- Supports the full instruction set, memory spaces, and parallel data transfer fields of the DSP96000 Family
- Modular programming features including local labels, sections, and external definition/reference directives
- Nested macro processing capability with support for macro libraries
- Complex expression evaluation including boolean operators
- Built-in functions for data conversion, string comparison, and common transcendental math operations

- Directives to define modulo and bit-reversed buffers
- Extensive error checking and reporting

Linker/Librarian

The linker relocates and links relocatable object modules from the macro cross assembler to create an absolute load file, which can be loaded directly into the DSP96000 simulator or converted to Motorola S-record format for PROM burning.

The librarian utility will merge separate, relocatable object modules into a single file, which allows common or frequently used modules to be grouped for convenient linking and storing.

Simulator

The simulator program is a software tool for developing programs and algorithms for the DSP96000 Family. This program exactly emulates the functions of the part, including all on-chip peripheral operations, the entire internal and external memory spaces, all memory and register updates associated with program code execution, and all exception processing activity. This emulation enables the simulator program to provide the user with an accurate measurement of code execution time, which is so critical in DSP applications.

The simulator program executes DSP object code, which has been generated using either the linker or the simulator's internal, single-line assembler. The object code is loaded into the simulated DSP memory map. Instruction execution can proceed until a user-defined breakpoint is encountered, or in single-step mode, stopping after each instruction has been executed. During program debug, the user may display and change any of the registers or memory locations.

The simulator package includes three, linkable, object-code libraries of simulator functions that were used to create the simulator. The libraries allow the user to build his own customized simulator and to integrate it with his unique C language system simulation. Source code for some of the functions, such as the terminal I/O functions and external memory accesses, is provided to allow the user to simulate the particular application.

Features of the simulator program are as follows:

- Simulates single or multiple DSP96001s
- Single-stepping through object programs
- Conditional or unconditional breakpoints
- Program patching using a resident, single-line assembler/disassembler
- Instruction and cycle timing counters
- Session and/or command logging for later reference
- ASCII input/output files for peripherals
- Help file and help line display of simulator commands
- Loading and saving of files to/from simulator memory
- Macro command definition and execution

- Display enable/disable of registers and memory
- Hexadecimal/decimal/binary calculator
- Linkable object-code libraries
 - Object library entry points
 - External memory
 - Screen management
- Nondisplay simulator

C Compiler

The DSP96KCCx is a full Kernighan and Ritchie implementation of the C programming language and supports development of DSP96000 Family applications.

The product will consist of the following software: 1) C compiler program (CC96000), 2) macro cross assembler program (ASM96000), and 3) linker/librarian program (LNK96000/LIB9600).

In summary, the C compiler supports:

- Full IEEE floating point
- Structures and unions
- In-line assembly language code compatibility
- Full function preprocessor
 - Macro definition/expansion
 - File inclusion
 - Conditional compilation
- Low compiler overhead
- Full error detection and reporting

HARDWARE: APPLICATION DEVELOPMENT SYSTEM

The DSP96001-based application development system (ADS) is a development tool for designing, debugging, and evaluating DSP96001 target system equipment. Internal and external DSP96001 operations can be monitored through the OnCE serial interface to the on-chip debug controller. The ADS is fully compatible with the DSP96000CLASx design-in software package and may act as an accelerator for testing simulated DSP96001 algorithms.

An IBM PC, Macintosh II, or Sun-3, using a parallel interface card, can act as the host platform between the user and the DSP96000ADS hardware. The ADS components are an application development module (ADM) board, a simple control/interface card, two cables, and a user interface program (see Figure 9).

Jumper options allow changing clock inputs, DSP96001 operating mode on reset, reconfiguration of RAM partitioning between program, X, or Y memory spaces, and address relocation of RAM and/or ROM.

The hardware features of the DSP96001ADS include:

- Full-speed operation at 26.67 MHz
- Multiple ADS support with programmable ADS addressing
- Configurable RAM for DSP96001 code development
- Standalone operation of ADM after initial development
- Interface card supports DSP56000 or DSP96000 Family ADMs

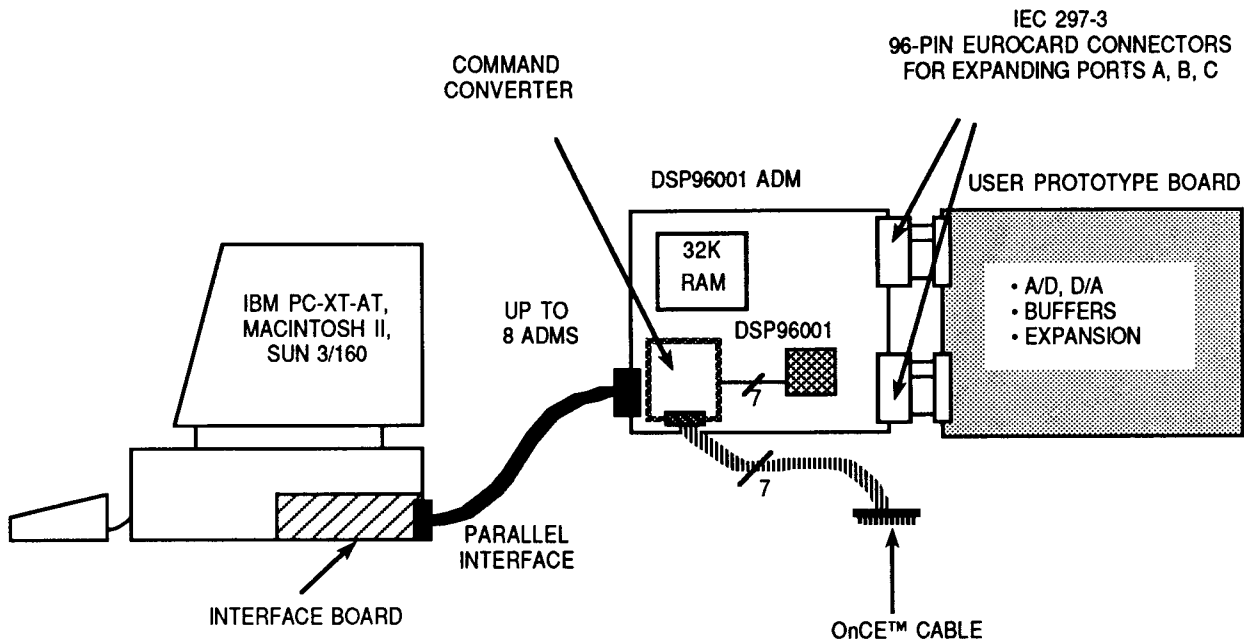


Figure 9. Application Development System Components

The features of the DSP96001ADS user interface program are as follows:

- Single/multiple stepping through DSP96001 object programs
- Up to 99 conditional or unconditional breakpoints
- Program patching using a single-line assembler/disassembler
- Session and/or command logging for later reference
- Loading and saving of files to/from ADM memory
- Macro command definition and execution
- Display enable/disable of registers and memory
- Debug commands that support multiple (up to 8) ADMs
- Hexadecimal/decimal/binary calculator
- System commands from within ADS user interface program
- Multiple input/output file access from DSP96001 object programs
- Fully compatible with the DSP9600CLASA design-in software package

The order number is DSP96001ADS.

DSP ELECTRONIC BULLETIN BOARD — DR. BuB

Dr. BuB is the name of the electronic bulletin board dedicated to DSP at Motorola. This bulletin board offers the following information on Motorola's DSP products:

- Current documentation
 - New products
 - Improvements to existing products

- Application notes
 - New
 - Updates to existing notes
- Question and answer forum
- Confidential mail service

How to Make an Appointment:

You can access Dr. BuB from anywhere in the world. To make an appointment with Dr. BuB the following equipment is needed:

- 1200-baud modem (Bell 212A or V.22)
- Terminal or personal computer
- Telephone line

This minimum configuration will enable the user to read Motorola DSP information and post questions and comments. However, a file transfer program such as XMODEM, YMODEM, or KERMIT will allow downloading of documentation. (XMODEM programs are generally bundled together with the purchase of personal computer modems.)


After obtaining the hardware and ensuring that it is operating properly, use the following procedure to log on:

1. Dial (512) 440-DSP1 (440-3771) to access Bell 212A modems, (512) 440-DSP3 (440-3773) to access 2400 baud modems, or (512) 440-DSP2 (440-3772) to access V.22 modems. Be sure to set the character format to 7 data, even parity, 1 stop bit.
2. Once the connection has been established, the computer will respond with "Dr. BuB login:". Type *guest* in lowercase, followed by a carriage return. If *GUEST* (uppercase) is entered, extraneous characters may appear on the screen.

3. Do not be confused if "password:" appears on the screen. Simply hit a carriage return and wait for "Dr. BuB login:" to reappear.
4. Finally, identify your terminal type. Terminal type is one of two things: 1) dumb, if you have no

terminal emulation software, or 2) the type of terminal being emulated in software on your personal computer. A brief list of the most common terminals emulated is displayed to assist in wording your entry.

5. Make the appropriate menu selections.

Motorola reserves the right to make changes without further notice to any products herein to improve reliability, function or design. Motorola does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights nor the rights of others. Motorola and  are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Employment Opportunity/Affirmative Action Employer.

Literature Distribution Centers:

USA: Motorola Literature Distribution; P.O. Box 20912; Phoenix, Arizona 85036.

EUROPE: Motorola Ltd.; European Literature Center; 88 Tanners Drive, Blakelands Milton Keynes, MK145BP, England.

ASIA PACIFIC: Motorola Semiconductors H.K. Ltd.; P.O. Box 80300; Cheung Sha Wan Post Office; Kowloon Hong Kong.



MOTOROLA

A23002-3 PRINTED IN USA 12/88 IMPERIAL LITRO C61295 30,000 YCAVAA

24

013366 2 - 4

DSP96001
BR574/D