TEXAS INSTRUMENTS

TMS320C31 Embedded Control

Technical Brief

PRELIMINARY

PRELIMINARY

TEXAS INSTRUMENTS

TMS320C31 Embedded Control Technical Brief

TEXAS
INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments Incorporated (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to current specifications in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Please be aware that TI products are not intended for use in life-support appliances, devices, or systems. Use of TI product in such applications requires the written approval of the appropriate TI officer. Certain applications using semiconductor devices may involve potential risks of personal injury, property damage, or loss of life. In order to minimize these risks, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards. Inclusion of TI products in such applications is understood to be fully at the risk of the customer using TI devices or systems.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Read This First

How to Use This Manual

This document contains the following chapters:

Chapter 1   **Introduction**
A general description of the TMS320C31, its key features, benefits, embedded-controller requirements, compatible devices, and development support.

Chapter 2   **TMS320C31 Architectural Overview**
Functional block diagram. TMS320C31 architecture description, hardware components, and device operation. Instruction set summary.

Chapter 3   **TMS320C31 Features/Performance Comparison**
Comparison of TMS320C31 benchmark performance and feature values versus those of other embedded controllers.

Chapter 4   **Application Examples**
Four application examples showing how the TMS320C30 and TMS320C31 have been used for system-control functions in several application areas.

Chapter 5   **Development Support**
Discussion of code-generation, debug, and system integration development flow. Summarizes features of Texas Instruments simulation and emulation development tools and describes available technical documentation and technical assistance.

Chapter 6   **TMS320C31 Third-Party Support**
Alphabetical listing of third-party manufacturers and suppliers who provide development support products for the TMS320C31 and description of their products.

Appendix A   **TMS320 DSP Family**
Description of DSP market, TI's role in the DSP industry, TMS320 product roadmap, and the five generations of TMS320 devices.

Appendix B   **Part Ordering Information**
Listings of the hardware and software available from Texas Instruments to support the TMS320C31 device.

Style and Symbol Conventions

This document uses the following conventions.

❏ Program listings, program examples, interactive displays, filenames, and symbol names are shown in a `special typeface` similar to a typewriter's. Examples use a **`bold version`** of the special typeface for emphasis; interactive displays use a **`bold version`** of the special typeface to distinguish commands that you enter from items that the system displays (such as prompts, command output, error messages, etc.).

Here is a sample program listing:

```
0011  0005  0001          .field    1, 2
0012  0005  0003          .field    3, 4
0013  0005  0006          .field    6, 3
0014  0006                .even
```

Here is an example of a system prompt and a command that you might enter:

`C:` **`csr –a /user/ti/simuboard/utilities`**

Square brackets are also used as part of the pathname specification for VMS pathnames; in this case, the brackets are actually part of the pathname (they are not optional).

❏ In syntax descriptions, the instruction, command, or directive is in a **bold typeface** font and parameters are in an *italic typeface*. Portions of a syntax that are in **bold** should be entered as shown; portions of a syntax that are in *italics* describe the type of information that should be entered. Here is an example of a directive syntax:

.asect "*section name***",** *address*

.asect is the directive. This directive has two parameters, indicated by *section name* and *address*. When you use .asect, the first parameter must be an actual section name, enclosed in double quotes; the second parameter must be an address.

❏ Two vertical bars (||) identify a parallel instruction. An instruction that is preceded by two vertical bars will be executed in parallel with the previous instruction in the assembly language source file. Here is an example of a parallel instruction:

```
   MPYI3   R7, R4, R0
|| ADDI3   *AR3,*AR5—(1),R3
```

Since the ADDI3 is preceded with two vertical bars, the two lines of assembly language are considered a single instruction where both an integer multiply and integer add are performed.

❏ An at character (@) preceding a label or expression in an instruction indicates that direct addressing is being performed and that the label or expression following the at character is used to form the data address. Here is an example:

ADDI   @0BCDEh, R7

In this instruction, the data address is formed by concatenating 0BCDEh with the current value of the data page pointer. The contents of this location is added to R7 and stored in R7.

❏ Braces ({ and }) indicate a list. The symbol **|** (read as *or*) separates items within the list. Here's an example of a list:

{ * | *+ | *− }

This provides three choices: *, *+, or *−.

Unless the list is enclosed in square brackets, you must choose one item from the list.

❏ Some directives can have a varying number of parameters. For example, the .byte directive can have up to 100 parameters. The syntax for this directive is:

.byte   *value$_1$ [, ... , value$_n$]*

This syntax shows that .byte must have at least one value parameter, but you have the option of supplying additional value parameters, separated by commas.

Related Documentation From Texas Instruments

TMS320C3x User's Guide (literature number SPRU031) describes the 'C3x ('C30 and 'C31) 32-bit floating-point microprocessors, developed for digital signal processing as well as embedded-control applications. Covered are its architecture, internal register structure, instruction set, pipeline, specifications, and operation of its DMA and its two serial ports. Software and hardware applications are included.

TMS320 Floating-Point DSP Optimizing C Compiler User's Guide (literature number SPRU034) describes the TMS320 floating-point C compiler. This C compiler accepts ANSI standard C source code and produces TMS320 assembly language source code for the 'C3x and 'C4x generations of devices.

TMS320 Floating-Point DSP Assembly Language Tools User's Guide (literature number SPRU035) describes the assembly language tools (assembler, linker, and other tools used to develop assembly language code), assembler directives, macros, common object file format, and symbolic debugging directives for the 'C3x and 'C4x generations of devices.

TMS320C3x C Source Debugger User's Guide (literature number SPRU053) tells you how to invoke the 'C3x emulator, evaluation module, and simulator versions of the C source debugger interface. This book discusses various aspects of the debugger interface, including window management, command entry, code execution, data management, and breakpoints, and includes a tutorial that introduces basic debugger functionality.

TMS320C30 Hewlett-Packard 64776 Analysis Subsystem User's Guide (literature number SPRU071) describes the analysis subsystem, which supplements the 'C30 emulator capabilities by providing realtime breakpoint, trace, and timing features. The analysis subsystem can be used only with the 'C30 emulator.

Trademarks

SPARC and S-bus are trademarks of Sun Microsystems, Inc.

Spirit 30 is a trademark of Sonitech International Inc.

SPOX is a trademark of Spectron Microsystems, Inc.

Tiger 30 is a trademark of DSP Research, Inc.

VPRO-4 is a trademark of Voice Processing Corporation.

If You Need Assistance. . .

If you want to. . .	Do this. . .
Request more information about Texas Instruments Digital Signal Processing (DSP) products	Call the CRC[†]: **(800) 336–5236**
	Or write to: Texas Instruments Incorporated Market Communications Manager, MS 736 P.O. Box 1443 Houston, Texas 77251–1443
Order Texas Instruments documentation	Call the CRC[†]: **(800) 336–5236**
Ask questions about product operation or report suspected problems	Call the DSP hotline: **(713) 274–2320**
Report mistakes in this document or any other TI documentation	Fill out and return the reader response card at the end of this book, or send your comments to: Texas Instruments Incorporated Technical Publications Manager, MS 702 P.O. Box 1443 Houston, Texas 77251–1443

[†] Texas Instruments Customer Response Center

Contents

Figures

Tables

Introduction

The Texas Instruments low-cost, high-performance TMS320C31 has defined a new role for digital signal processors in embedded systems. Well-suited for general-purpose use, the TMS320C31 is finding widespread acceptance as an embedded controller in applications such as:

- [ ] Industrial automation
- [ ] Telecommunications
- [ ] Motor control
- [ ] Automotive
- [ ] Instrumentation
- [ ] Laser printers
- [ ] Scanners
- [ ] Voice mail

and is expanding the role of DSPs from math support to embedded control.

The topics covered in this chapter include:

1.1 Embedded Controller Requirements

An embedded controller is a dedicated processor used in systems or subsystems such as laser printers, voice mail systems, and bar-code readers to control a specific set of functions. Unlike a PC or workstation host CPU, an embedded controller is not accessible to the system user to run different software packages or to be reprogrammed, but is used to cost-effectively control a predetermined set of functions. To make these types of systems successful, embedded controllers must possess the following characteristics:

❑ High overall performance for peripheral device management and data flow control
❑ Software compatibility (efficient compilers and realtime operating system support)
❑ Mature development tools and third-party support
❑ Flexibility
❑ Reliability
❑ Availability
❑ Low device price
❑ Low system cost

The TMS320C31's ability to satisfy these needs makes it an excellent choice when compared to embedded RISC and high-end CISC embedded controllers.

The TMS320C31:

❑ Provides a low-cost solution
❑ Supports a general-purpose programming model
❑ Supports efficient C language compilation
❑ Enables high-performance system control
❑ Supports coprocessor math performance on-chip
❑ Integrates system peripherals on-chip
❑ Allows fast context switching

The TMS320C31 is an embedded controller with dedicated digital signal processing support that provides low cost, high performance, system integration, and ease of use. Due to these cost and performance advantages, the TMS320C31 is displacing RISC and high-end CISC processors in a wide range of applications across many industries.

1.2 TMS320C31 Key Features

The TMS320C31 includes the features normally associated with a general-purpose embedded controller, so designing with it is very similar to designing with RISC or CISC devices. But the 'C31 is distinguished by many high-performance features not found on processors in its price range:

❑ High performance

  ■ 50-ns instruction cycle
  ■ 20 MIPS (million instructions per second)
  ■ 40 MFLOPS (million floating-point operations per second)
  ■ 220 MOPS (million operations per second) (see Figure 1–1 on page 1-4)
  ■ 80-Mbytes/second I/O bandwidth
  ■ 0.200 μs interrupt response
  ■ 60-ns and 74-ns devices also available

❑ Register-based, pipelined CPU

  ■ Parallel multiply and arithmetic/logical operations on integer or floating-point numbers in a single cycle
  ■ Eight extended-precision registers
  ■ 24-bit address space
  ■ Two address generators with eight auxiliary registers, two index registers, and two auxiliary register arithmetic units
  ■ 32-bit barrel shifter

❑ Powerful instruction set

  ■ Single-cycle instruction execution
  ■ System control and numeric operations
  ■ Two and three operand instructions
  ■ Zero-overhead looping
  ■ Single-cycle branching
  ■ Conditional calls and returns
  ■ Flexible addressing modes including circular addressing and auto-increment/decrement modes allow high-speed data accesses
  ■ Single-cycle parallel math and memory operations
  ■ Interlocked instructions for multiprocessing support

❑ Integrated peripherals

  ■ DMA controller for concurrent I/O and CPU operation
  ■ Two-way set associative instruction cache maximizes performance while minimizing system cost
  ■ Flexible serial port for 8/16/24/32-bit transfers which can be configured for general-purpose bit I/O plus two 16-bit timers
  ■ Two 32-bit timers which can also be configured for bit I/O

❑ Extensive internal busing and parallelism for extremely fast data-movement capability

❑ 8K bytes of single-cycle dual-access internal RAM support two accesses per machine cycle—can act as program memory, data memory, cache to external memory, or register file extensions

❑ Memory interface optimized for single-cycle SRAM accesses and static-column decode DRAMs for high-speed external memory access while maintaining low system cost

❑ Boot loader to load/execute programs from other processors or inexpensive EPROMS

❑ On-chip emulation for true nonintrusive visibility and control during debug

❑ 132-pin plastic quad flat pack (PQFP) package

❑ Low price

The TMS320C31 is described in detail in Chapter 2.

Figure 1–1. TMS320C31 Performance

Sustained Processing:
- High-Performance CPU
- DMA Controller

Sustained I/O:
- Primary Bus
- DMA Controller
- Serial Port

50-ns
Cycle Time

+

CPU and DMA PERFORMANCE

CPU – 8 OPS/Cycle = 160 MOPS
- 2 Data Accesses   40 MOPS
- 1 FP Multiply   20 MOPS
- 1 FP ALU Operation   20 MOPS
- 2 Addr. Register Mods   40 MOPS
- 1 Loop Counter Update   20 MOPS
- 1 Branch   20 MOPS

DMA COPROCESSOR
     3 OPS/Cycle =   60 MOPS
- 1 Data Access   20 MOPS
- 1 Addr. Register Mod.   20 MOPS
- 1 Transfer Counter   20 MOPS
   Update

TOTAL MOPS = 220 MOPS

DATA THROUGHPUT

Primary Bus   80M bytes/sec
Serial Port   2M bytes/sec

TOTAL I/O = 82M bytes/sec

1.3 **Compatible Devices**

The TMS320C31 is one of two members of the TMS320C3x generation of DSPs. The other member is the TMS320C30, which is object-code compatible with the 'C31. The 'C30 is identical to the 'C31 except that it has 4K words of ROM, two serial ports, and a second external bus. For more information on the TMS320C30, refer to the *TMS320C3x User's Guide* (literature number SPRU031). Figure 1–2 is a block diagram of the TMS320C3x devices. The shaded areas highlight the features that apply only to the 'C30.

Figure 1–2. TMS320C3x Block Diagram
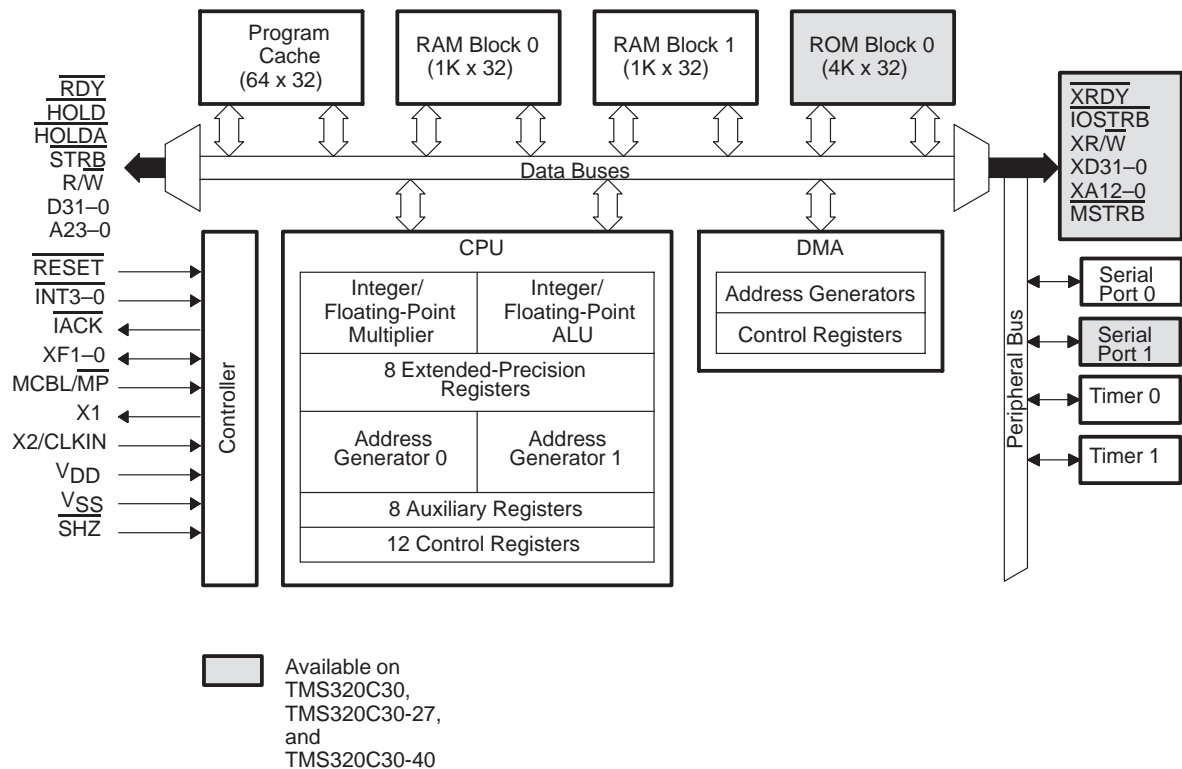
In addition, the 'C30 and 'C31 are both source-code compatible with the TMS320C4x, which is the first DSP designed specifically for parallel processing. For more information on the 'C4x, refer to the *TMS320C4x Technical Brief* (literature number SPRU076).

1.4 TMS320C31 Development Support

The 'C31's general-purpose, 32-bit architecture and TI's comprehensive set of development tools make designing systems with a 'C31 as easy as designing with a traditional controller. These tools include

❑ ANSI-compatible optimizing C compiler
❑ Realtime operating system support
❑ The programmer's interface—a window-based C-source/assembly debugger
❑ Code profiler
❑ Software simulator
❑ Low-cost evaluation module (EVM)
❑ TMS320C3x XDS scan-based emulator
❑ 'C3x application board
❑ HP64700 analysis subsystem
❑ Extensive third-party support
❑ Hotline support
❑ Bulletin board support
❑ Thousands of pages of application notes and technical documentation

A complete description of TMS320C31 development support can be found in Chapter 5. Figure 1–3 illustrates the 'C31 development flow.

Figure 1–3. TMS320C3x Development Environment
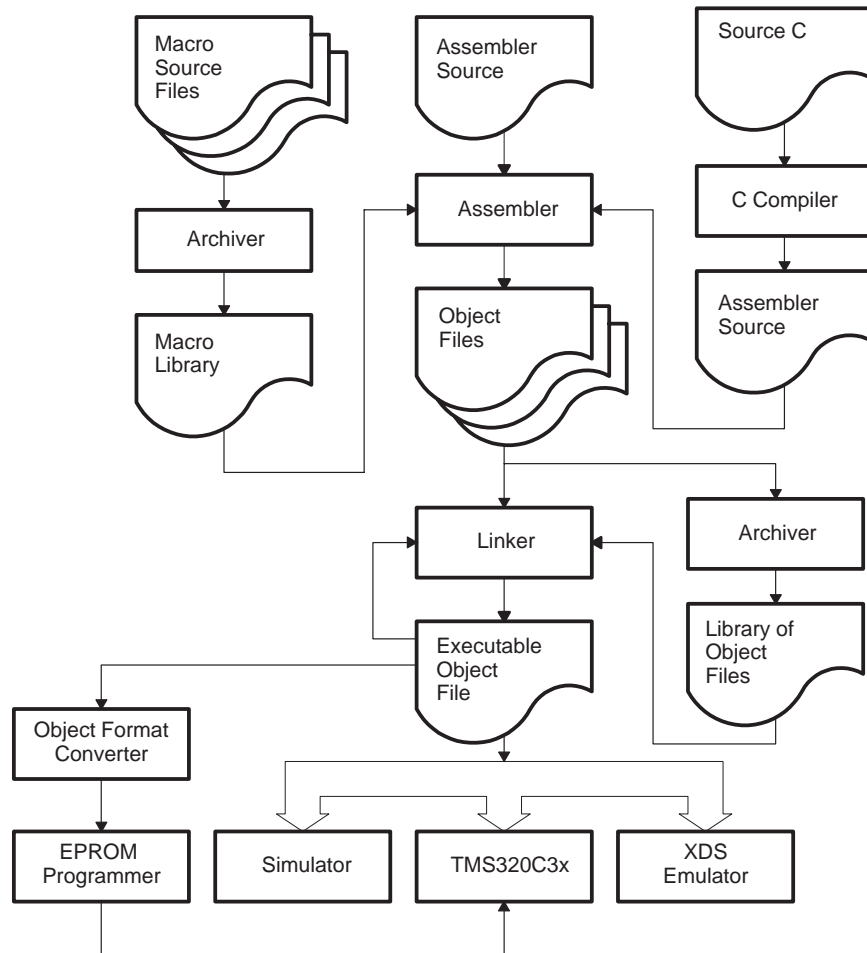
TMS320C31 performance benchmarks can be found in Chapter 3 and detailed system examples are shown in Chapter 4.

1.5 Benefits of a TMS320C31-Based Embedded System

The device price, development environment, external memory cost, and integrated peripherals of the TMS320C31 are equivalent to those of 32-bit microcontroller solutions. At the same time, the powerful instruction set and pipelined CPU provide the system control performance of a RISC processor—at a more affordable price. But the TMS320C31 is superior to RISC/CISC solutions in numerical performance and emulation capability. This best-of-both-worlds feature set delivers many benefits to next-generation embedded systems.

With a TMS320C31, many added-cost system features become reduced-cost features. Traditional embedded-system architectures use a microcontroller for system control and a coprocessor (companion math chip, programmable or special-purpose DSP, or ASIC) for math support. This traditional system architecture has performance and time-to-market drawbacks because the designer must learn two different architectures and development environments, and attempt to implement efficient communications between different types of processors. Today, designers are using a 'C31 to replace microcontrollers for higher performance and to reduce system cost and time to market in dual-processor designs. Also, for even higher performance and homogeneous systems, multiple 'C31s can be used. The 'C31 offers numerous advantages for embedded-control applications such as voice mail, industrial automation, instrumentation, audio, motor control, automotive, and laser printer systems. Figure 1–4 shows the benefits of replacing a controller/coprocessor with a TMS320C31.

Figure 1–4. Benefits of Replacing a Controller/Coprocessor With a TMS320C31-Based Embedded System
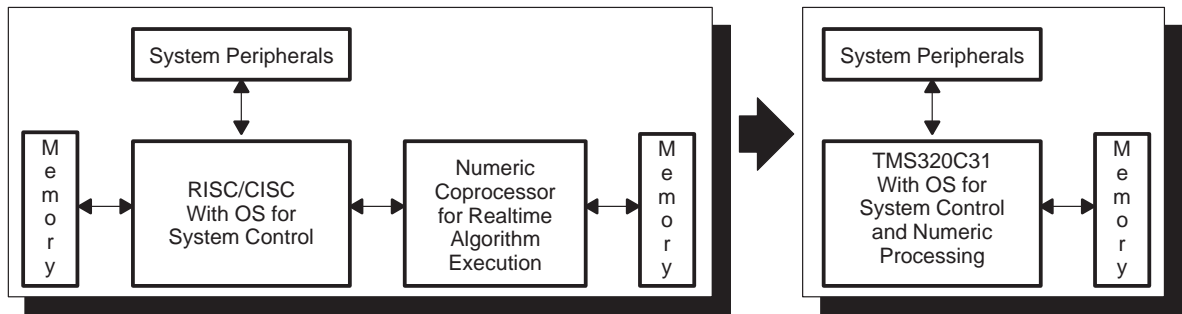
Replacement Benefits:

■ Simplified design	■ Lower memory cost
■ Reduced data flow	■ Lower device count & cost
■ Greater design flexibility	■ Fewer communication bottlenecks
■ Small form factor	■ Single development environment

TMS320C31 Architectural Overview

This chapter provides an architectural overview of the TMS320C31 embedded processor. An in-depth description of its features can be found in the *TMS320C3x User's Guide*.

Topics discussed in this chapter include:

2.1 TMS320C31 Block Diagram

Figure 2–1 is a block diagram of the TMS320C31 architecture. Throughout this chapter, refer to this block diagram to better understand the interface of the components of the 'C31 embedded controller.

Figure 2–1. TMS320C31 Block Diagram
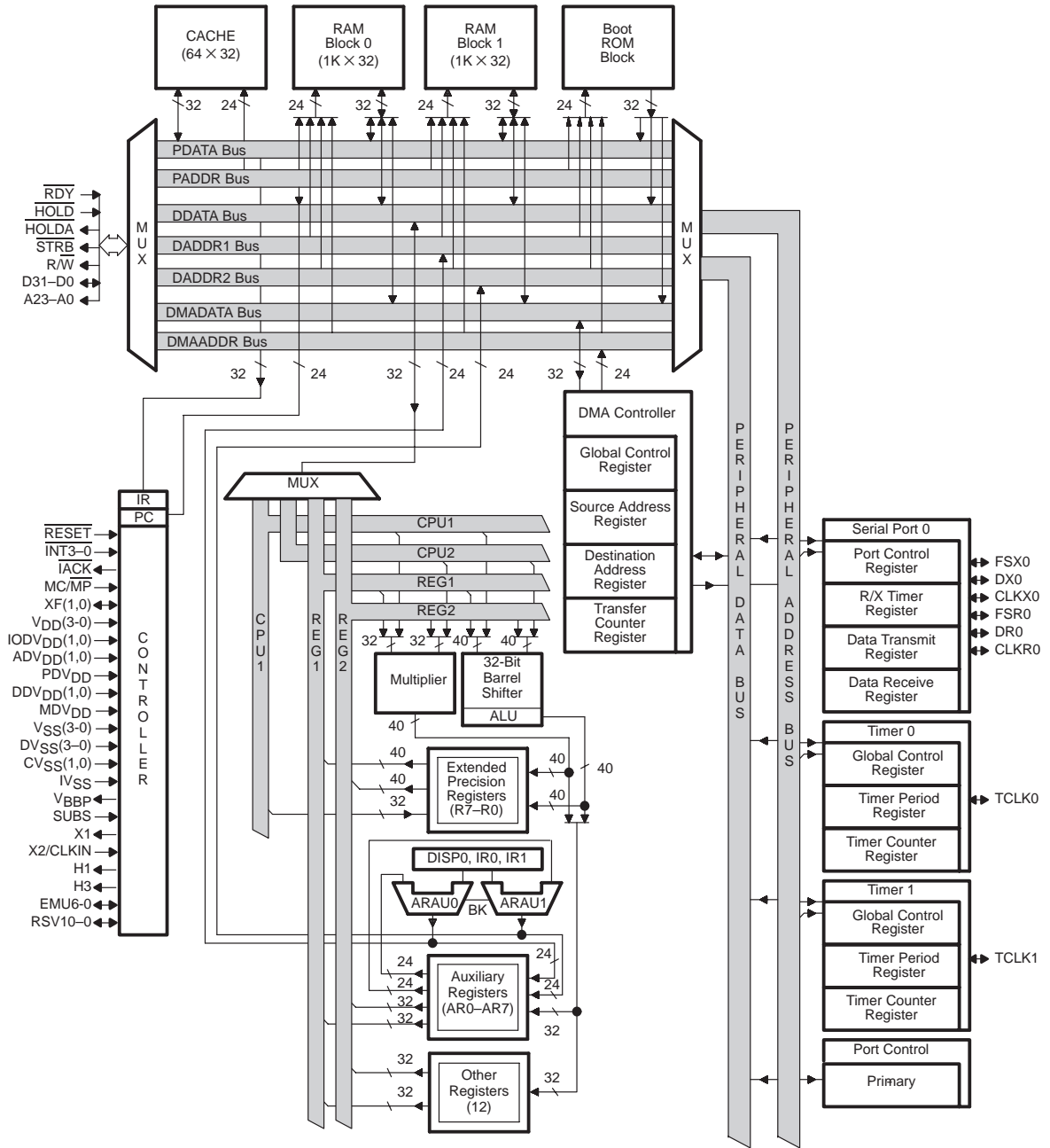
2.2   Central Processing Unit (CPU)

The TMS320C31 has a register-based, pipelined CPU architecture. The 'C31 CPU is similar to a RISC microprocessor CPU in that most instructions execute in a single cycle. However, the 'C31 instruction set is more powerful—multiple operations can be performed in a single-instruction cycle and the operands of logical and arithmetic instructions can be read from memory and operated on in a single cycle. Because its separate multiplier and ALU are incorporated into the CPU, the 'C31 supports single-cycle logical and arithmetic operations. These units do not require pipelined, staged execution to achieve maximum performance, allowing the 'C31 to achieve low-latency execution of numeric operations. In addition, the same multiplier and ALU are used for both integer and floating-point math, providing you flexibility and equal performance for either data format.

The TMS320C31 can perform a multiply and ALU operation in a single cycle, allowing realtime DSP or other math and logical functions to be done in parallel every cycle, without latency. Hence, 40 MFLOPS or 40 integer multiply-accumulates operations can be sustained with a 40-MHz TMS320C31. In addition to the integrated math support, the CPU architecture provides a high degree of parallelism on-chip, allowing on- and off-chip resources to be utilized most effectively.

Figure 2–2 is a block diagram of the 'C31 CPU.

Figure 2–2. Central Processing Unit (CPU)
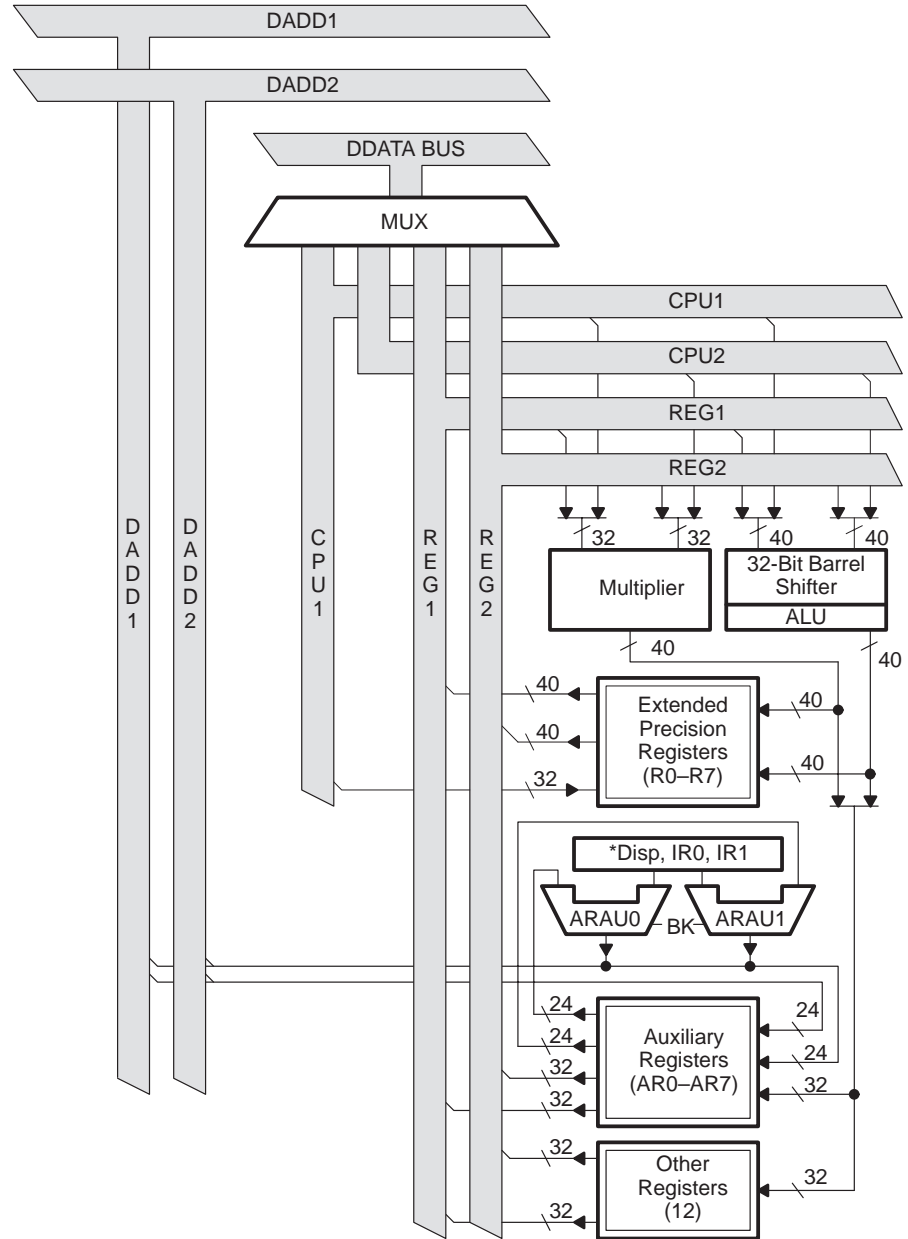
* Disp = an 8-bit integer displacement carried in a program control instruction

2.2.1 CPU Register File

The TMS320C31 provides 28 registers in a multiport register file that is tightly coupled to the CPU. All of these registers can be operated upon by the multiplier and ALU and can be used as general-purpose registers. However, the registers also have some special functions. For example, the eight extended-precision registers are especially suited for maintaining extended-precision floating-point results. The eight auxiliary registers support a variety of indirect addressing modes and can be used as general-purpose 32-bit integer and logical registers. The remaining registers provide system functions such as addressing, stack management, processor status, interrupts, and block repeat.

The register names and assigned functions are listed in Table 2–1. Following the table, the function of each register or group of registers is briefly described.

Table 2–1. CPU Registers

Register Name	Assigned Function
R0	Extended-precision register 0
R1	Extended-precision register 1
R2	Extended-precision register 2
R3	Extended-precision register 3
R4	Extended-precision register 4
R5	Extended-precision register 5
R6	Extended-precision register 6
R7	Extended-precision register 7
AR0	Auxiliary register 0
AR1	Auxiliary register 1
AR2	Auxiliary register 2
AR3	Auxiliary register 3
AR4	Auxiliary register 4
AR5	Auxiliary register 5
AR6	Auxiliary register 6
AR7	Auxiliary register 7
DP	Data-page pointer
IR0	Index register 0
IR1	Index register 1
BK	Block size
SP	System stack pointer
ST	Status register
IE	CPU/DMA interrupt enable
IF	CPU interrupt flags
IOF	I/O flags
RS	Repeat start address
RE	Repeat end address
RC	Repeat counter
PC	Program counter

The **extended-precision registers (R7–R0)** are capable of storing and supporting operations on 32-bit integer and 40-bit floating-point numbers. Any instruction that assumes the operands are floating-point numbers uses bits 39–0. If the operands are either signed or unsigned integers, only bits 31–0 are used; bits 39–32 remain unchanged. Bits 39–32 remain unchanged for all shift operations.

The 32-bit **auxiliary registers (AR7–AR0)** can be accessed by the CPU and modified by the two Auxiliary Register Arithmetic Units (ARAUs). The primary function of the auxiliary registers is the generation of 24-bit addresses. They can also be used as loop counters or as 32-bit general-purpose registers that can be modified by the multiplier and ALU.

The **data page pointer (DP)** is a 32-bit register. The eight LSBs of the data page pointer are used by the direct addressing mode as a pointer to the page of data being addressed. Data pages are 64K words long with a total of 256 pages.

The 32-bit **index registers** (IR0, IR1) contain the value used by the Auxiliary Register Arithmetic Unit (ARAU) to compute an indexed address.

The ARAU uses the 32-bit **block size register (BK)** in circular addressing to specify the data block size.

The **system stack pointer (SP)** is a 32-bit register that contains the address of the top of the system stack. The SP always points to the last element pushed onto the stack. A push performs a preincrement, and a pop performs a post-decrement of the system stack pointer. The SP is manipulated by interrupts, traps, calls, returns, and the PUSH and POP instructions.

The **status register (ST)** contains global information relating to the state of the CPU. Typically, operations set the condition flags of the status register according to whether the result is zero, negative, etc. This includes register load and store operations as well as arithmetic and logical functions. When the status register is loaded, however, a bit-for-bit replacement is performed with the contents of the source operand, regardless of the state of any bits in the source operand. Therefore, following a load, the contents of the status register are equal to the contents of the source operand. This allows the status register to be easily saved and restored.

The **CPU/DMA interrupt enable register (IE)** is a 32-bit register. The CPU interrupt enable bits are in locations 10–0. The DMA interrupt enable bits are in locations 26–16. A 1 in a CPU/DMA interrupt enable register bit enables the corresponding interrupt. A 0 disables the corresponding interrupt.

The **CPU interrupt flag register (IF)** is also a 32-bit register. A 1 in a CPU interrupt flag register bit indicates that the corresponding interrupt is set. A 0 indicates that the corresponding interrupt is not set.

The **I/O flags register (IOF)** controls the function of the dedicated external pins, XF0 and XF1. These pins may be configured for input or output and may also be read from and written to.

The **repeat counter (RC)** is a 32-bit register used to specify the number of times a block of code is to be repeated when performing a block repeat. When the processor is operating in the repeat mode, the 32-bit **repeat start address register (RS)** contains the starting address of the block of program memory to be repeated, and the 32-bit **repeat end address register (RE)** contains the ending address of the block to be repeated.

The **program counter (PC)** is a 32-bit register containing the address of the next instruction to be fetched. Although the PC is not part of the CPU register file, it is a register that can be modified by instructions that modify the program flow.

2.2.2 Auxiliary Register Arithmetic Units (ARAUs)

Two auxiliary register arithmetic units (ARAU0 and ARAU1) can generate two addresses in a single cycle. The ARAUs operate in parallel with the multiplier and ALU. They support addressing with displacements, index registers (IR0 and IR1), and circular and bit-reversed addressing.

2.2.3 Multiplier

The multiplier performs single-cycle multiplications on 24-bit integer and 32-bit floating-point values. The TMS320C31 implementation of floating-point arithmetic allows for floating-point operations at fixed-point speeds via a 50-ns instruction cycle and a high degree of parallelism. To gain even higher throughput, you can use parallel instructions to perform a multiply and ALU operation in a single cycle.

When the multiplier performs floating-point multiplication, the inputs are 32-bit floating-point numbers, and the result is a 40-bit floating-point number. When the multiplier performs integer multiplication, the input data is 24 bits and yields a 32-bit result.

2.2.4 Arithmetic Logic Unit (ALU)

The ALU performs single-cycle operations on 32-bit integer, 32-bit logical, and 40-bit floating-point data, including single-cycle integer and floating-point conversions. Results of the ALU are always maintained in 32-bit integer or 40-bit floating-point formats. The barrel shifter is used to shift up to 32 bits left or right in a single cycle.

Internal buses, CPU1/CPU2 and REG1/REG2, carry two operands from memory and two operands from the register file, thus allowing parallel multiplies and adds/subtracts on four integer or floating-point operands in a single cycle.

2.2.5 CPU Memory Addressing Modes

The TMS320C31 supports a base set of general-purpose instructions as well as arithmetic-intensive instructions that are particularly suited for digital signal processing and other numeric-intensive applications.

For use with the general-purpose and arithmetic instructions, five groups of addressing modes are provided on the TMS320C31. Six types of addressing may be used within the groups, as shown in the following list:

❑ General addressing modes:

■ Register. The operand is a CPU register.

■ Short immediate. The operand is a 16-bit immediate value.

■ Direct. The operand is the contents of a 24-bit address.

■ Indirect. An auxiliary register indicates the address of the operand.

❑ Three-operand addressing modes:

■ Register. Same as for general addressing mode.

■ Indirect. Same as for general addressing mode.

❑ Parallel addressing modes:

■ Register. The operand is an extended-precision register.

■ Indirect. Same as for general addressing mode.

❑ Long-immediate addressing mode:

■ Long-immediate. The operand is a 24-bit immediate value.

❑ Conditional branch addressing modes:

■ Register. Same as for general addressing mode

■ PC-relative. A signed 16-bit displacement is added to the PC.

The various indirect addressing options available for the 'C31 are shown in Table 2–2. The table shows the options, along with the value of the modification (mod) field, assembler syntax, operation, and function for each.

Table 2–2. Indirect Addressing

Mod Field	Syntax	Operation	Description
Indirect Addressing With Displacement			
00000	*+ARn(disp)	addr = ARn + disp	With predisplacement add
00001	*−ARn(disp)	addr = ARn − disp	With predisplacement subtract
00010	*++ARn(disp)	addr = ARn + disp ARn = ARn + disp	With predisplacement add and modify
00011	*−−ARn(disp)	addr = ARn − disp ARn = ARn − disp	With predisplacement subtract and modify
00100	*ARn++(disp)	addr = ARn ARn = ARn + disp	With postdisplacement add and modify
00101	*ARn−−(disp)	addr = ARn ARn = ARn − disp	With postdisplacement subtract and modify
00110	*ARn++(disp)%	addr = ARn ARn = circ(ARn + disp)	With postdisplacement add and circular modify
00111	*ARn−−(disp)%	add = ARn ARn = circ(ARn − disp)	With postdisplacement subtract and circular modify
Indirect Addressing With Index Register IRO			
01000	*+ARn(IR0)	addr = ARn + IR0	With preindex (IR0) add
01001	*−ARn(IR0)	addr = ARn − IR0	With preindex (IR0) subtract
01010	*++ARn(IR0)	addr = ARn + IR0 ARn = ARn + IR0	With preindex (IR0) add and modify
01011	*−−ARn(IR0)	addr = ARn − IR0 ARn = ARn − IR0	With preindex (IR0) subtract and modify
01100	*ARn++(IR0)	addr = ARn ARn = ARn + IR0	With postindex (IR0) add and modify
01101	*ARn−−(IR0)	addr= ARn ARn = ARn − IR0	With postindex (IR0) subtract and modify
01110	*ARn++(IR0)%	addr = ARn ARn = circ(ARn + IR0)	With postindex (IR0) add and circular modify
01111	*ARn−−(IR0)%	addr = ARn ARn = circ(ARn) − IR0	With postindex (IR0) subtract and circular modify

LEGEND:

addr	=	memory address
ARn	=	auxiliary register AR0 – AR7
IRn	=	index register IR0 or IR1
disp	=	displacement
++	=	add and modify
−−	=	subtract and modify
circ( )	=	address in circular addressing
%	=	where circular addressing is performed

Table 2–2. Indirect Addressing (Concluded)

Mod Field	Syntax	Operation	Description
Indirect Addressing With Index Register IR1			
10000	*+ARn(IR1)	addr = ARn + IR1	With preindex (IR1) add
10001	*−ARn(IR1)	addr = ARn − IR1	With preindex (IR1) subtract
10010	*++ARn(IR1)	addr = ARn + IR1 ARn = ARn + IR1	With preindex (IR1) add and modify
10011	*−−ARn(IR1)	addr = ARn − IR1 ARn = ARn − IR1	With preindex (IR1) subtract and modify
10100	*ARn++(IR1)	addr = ARn ARn = ARn + IR1	With postindex (IR1) add and modify
10101	*ARn−−(IR1)	addr = ARn ARn = ARn − IR1	With postindex (IR1) subtract and modify
10110	*ARn++(IR1)%	addr = ARn ARn = circ(ARn + IR1)	With postindex (IR1) add and circular modify
10111	*ARn−−(IR1)%	addr = ARn ARn = circ(ARn − IR1)	With postindex (IR1) subtract and circular modify
Indirect Addressing (Special Cases)			
11000	*ARn	addr = ARn	Indirect
11001	*ARn++(IR0)B	addr = ARn ARn = B(ARn + IR0)	With postindex (IR0) add and bit-reversed modify

LEGEND:

addr	=	memory address
ARn	=	auxiliary register AR0 – AR7
IRn	=	index register IR0 or IR1
disp	=	displacement
++	=	add and modify
−−	=	subtract and modify
circ( )	=	address in circular addressing
%	=	where circular addressing is performed
B	=	where bit-reversed addressing is performed

2.2.6 Instruction Set Summary

The 'C31 offers instructions for both embedded control and numeric support. The following tables show each instruction's mnemonic, description, and operation. Table 2–3 shows the system control instructions; Table 2–4 lists the program flow control instructions; Table 2–5 shows the logical and bit-manipulation instructions; Table 2–6 lists the load and store instructions; Table 2–7 shows the arithmetic instructions; and Table 2–8 summarizes the TMS320C31 parallel instructions, which execute in a single cycle.

Table 2–3. System Control Instruction Summary

Mnemonic	Description	Operation
IACK	Interrupt acknowledge	Dummy read of src IACK toggled low, then high
IDLE	Idle until interrupt	PC + 1 $\rightarrow$ PC Idle until next interrupt
NOP	No operation	Modify ARn if specified
POP	Pop integer from stack	*SP$--$ $\rightarrow$ Dreg
POPF	Pop floating-point value from stack	*SP$--$ $\rightarrow$ Rn
PUSH	Push integer on stack	Sreg $\rightarrow$ *++ SP
PUSHF	Push floating-point value on stack	Rn $\rightarrow$ *++ SP

RETI*cond*	Return from interrupt conditionally	If cond = true or missing:
		  *SP$--$ $\rightarrow$ PC
		  1 $\rightarrow$ ST (GIE)
		Else, continue

RETS*cond*	Return from subroutine conditionally	If cond = true or missing: *SP$--$ $\rightarrow$ PC Else, continue
SIGI	Signal, interlocked	Signal interlocked operation Wait for interlock acknowledge Clear interlock
SWI	Software interrupt	Perform emulator interrupt sequence

TRAP*cond*	Trap conditionally	If cond = true or missing:
		  Next PC $\rightarrow$ * ++ SP
		  Trap vector N $\rightarrow$ PC
		  0 $\rightarrow$ ST (GIE)
		Else, continue

LEGEND:

src	=	general addressing modes	Dreg	=	register address (any register)
src1	=	three-operand addressing modes	Rn	=	register address (R7 — R0)
src2	=	three-operand addressing modes	Daddr	=	destination memory address
Csrc	=	conditional-branch addressing modes	ARn	=	auxiliary register n (AR7 — AR0)
Sreg	=	register address (any register)	addr	=	24-bit immediate address (label)
count	=	shift value (general addressing modes)	cond	=	condition code (see Chapter 11)
SP	=	stack pointer	ST	=	status register
GIE	=	global interrupt enable register	RE	=	repeat interrupt register
RM	=	repeat mode bit	RS	=	repeat start register
TOS	=	top of stack	PC	=	program counter
			C	=	carry bit

Table 2–4. Program Flow Control Instruction Summary

Mnemonic	Description	Operation

B*cond*	Branch conditionally (standard)	If cond = true:
		  If Csrc is a register, Csrc → PC
		  If Csrc is a value, Csrc + PC → PC
		Else, PC + 1 → PC

B*cond*D	Branch conditionally (delayed)	If cond = true:
		  If Csrc is a register, Csrc → PC
		  If Csrc is a value, Csrc + PC + 3 → PC
		Else, PC + 1 → PC

BR	Branch unconditionally (standard)	Value → PC
BRD	Branch unconditionally (delayed)	Value → PC
CALL	Call subroutine	PC + 1 → TOS Value → PC

CALL*cond*	Call subroutine conditionally	If cond = true:
		  PC + 1 → TOS
		  If Csrc is a register, Csrc → PC
		  If Csrc is a value, Csrc + PC → PC
		Else, PC + 1 → PC

DB*cond*	Decrement and branch conditionally (standard)	ARn – 1 → ARn
		If cond = true and ARn ≥ 0:
		  If Csrc is a register, Csrc → PC
		  If Csrc is a value, Csrc + PC + 1 → PC
		Else, PC + 1 → PC

DB*cond*D	Decrement and branch conditionally (delayed)	ARn – 1 → ARn
		If cond = true and ARn ≥ 0:
		  If Csrc is a register, Csrc → PC
		  If Csrc is a value, Csrc + PC + 3 → PC
		Else, PC + 1 → PC

RPTB	Repeat block of instructions	src → RE 1 → ST (RM) Next PC → RS
RPTS	Repeat single instruction	src → RC 1 → ST (RM) Next PC → RS Next PC → RE

LEGEND:

src	=	general addressing modes	Dreg	=	register address (any register)
src1	=	three-operand addressing modes	Rn	=	register address (R7 — R0)
src2	=	three-operand addressing modes	Daddr	=	destination memory address
Csrc	=	conditional-branch addressing modes	ARn	=	auxiliary register n (AR7 — AR0)
Sreg	=	register address (any register)	addr	=	24-bit immediate address (label)
count	=	shift value (general addressing modes)	cond	=	condition code (see Chapter 11)
SP	=	stack pointer	ST	=	status register
GIE	=	global interrupt enable register	RE	=	repeat interrupt register
RM	=	repeat mode bit	RS	=	repeat start register
TOS	=	top of stack	PC	=	program counter
			C	=	carry bit

Table 2–5. Logical and Bit Manipulation Instruction Summary

Mnemonic	Description	Operation
AND	Bitwise logical-AND	Dreg AND src $\to$ Dreg
AND3	Bitwise logical-AND (3-operand)	src1 AND src2 $\to$ Dreg
ANDN	Bitwise logical-AND with complement	Dreg AND $\overline{\text{src}}$ $\to$ Dreg
ANDN3	Bitwise logical-ANDN (3-operand)	src1 AND $\overline{\text{src2}}$ $\to$ Dreg
CMPF	Compare floating-point values	Set flags on Rn – src
CMPF3	Compare floating-point values (3-operand)	Set flags on src1 – src2
CMPI	Compare integers	Set flags on Dreg – src
CMPI3	Compare integers (3-operand)	Set flags on src1 – src2
NOT	Bitwise logical-complement	$\overline{\text{src}}$ $\to$ Dreg
OR	Bitwise logical-OR	Dreg OR src $\to$ Dreg
OR3	Bitwise logical-OR (3-operand)	src1 OR src2 $\to$ Dreg
TSTB	Test bit fields	Dreg AND src
TSTB3	Test bit fields (3-operand)	src1 AND src2
XOR	Bitwise exclusive-OR	Dreg XOR src $\to$ Dreg
XOR3	Bitwise exclusive-OR (3-operand)	src1 XOR src2 $\to$ Dreg

LEGEND:

src	=	general addressing modes	Dreg	=	register address (any register)
src1	=	three-operand addressing modes	Rn	=	register address (R7 — R0)
src2	=	three-operand addressing modes	Daddr	=	destination memory address
Csrc	=	conditional-branch addressing modes	ARn	=	auxiliary register n (AR7 — AR0)
Sreg	=	register address (any register)	addr	=	24-bit immediate address (label)
count	=	shift value (general addressing modes)	cond	=	condition code (see Chapter 11)
SP	=	stack pointer	ST	=	status register
GIE	=	global interrupt enable register	RE	=	repeat interrupt register
RM	=	repeat mode bit	RS	=	repeat start register
TOS	=	top of stack	PC	=	program counter
			C	=	carry bit

Table 2–6. Load and Store Instruction Summary

Mnemonic	Description	Operation
LDE	Load floating-point exponent	src(exponent) $\rightarrow$ Rn(exponent)
LDF	Load floating-point value	src $\rightarrow$ Rn
LDFcond	Load floating-point value conditionally	If cond = true, src $\rightarrow$ Rn Else, Rn is not changed
LDFI	Load floating-point value, interlocked	Signal interlocked operation src $\rightarrow$ Rn
LDI	Load integer	src $\rightarrow$ Dreg
LDIcond	Load integer conditionally	If cond = true, src $\rightarrow$ Dreg Else, Dreg is not changed
LDII	Load integer, interlocked	Signal interlocked operation src $\rightarrow$ Dreg
LDM	Load floating-point mantissa	src (mantissa) $\rightarrow$ Rn (mantissa)
STF	Store floating-point value	Rn $\rightarrow$ Daddr
STFI	Store floating-point value, interlocked	Signal interlocked operation Rn $\rightarrow$ Daddr
STI	Store integer	Sreg $\rightarrow$ Daddr
STII	Store integer, interlocked	Signal interlocked operation Sreg $\rightarrow$ Daddr

LEGEND:

src	=	general addressing modes	Dreg	=	register address (any register)
src1	=	three-operand addressing modes	Rn	=	register address (R7 — R0)
src2	=	three-operand addressing modes	Daddr	=	destination memory address
Csrc	=	conditional-branch addressing modes	ARn	=	auxiliary register n (AR7 — AR0)
Sreg	=	register address (any register)	addr	=	24-bit immediate address (label)
count	=	shift value (general addressing modes)	cond	=	condition code (see Chapter 11)
SP	=	stack pointer	ST	=	status register
GIE	=	global interrupt enable register	RE	=	repeat interrupt register
RM	=	repeat mode bit	RS	=	repeat start register
TOS	=	top of stack	PC	=	program counter
			C	=	carry bit

Table 2–7. Arithmetic Instruction Set Summary

Mnemonic	Description	Operation
ABSF	Absolute value of a floating-point number	$|src| \rightarrow$ Rn
ABSI	Absolute value of an integer	$|src| \rightarrow$ Dreg
ADDC	Add integers with carry	src + Dreg + C $\rightarrow$ Dreg
ADDC3	Add integers with carry (3-operand)	src1 + src2 + C $\rightarrow$ Dreg
ADDF	Add floating-point values	src + Rn $\rightarrow$ Rn
ADDF3	Add floating-point values (3-operand)	src1 + src2 $\rightarrow$ Rn
ADDI	Add integers	src + Dreg $\rightarrow$ Dreg
ADDI3	Add integers (3-operand)	src1 + src2 + $\rightarrow$ Dreg

ASH	Arithmetic shift	If count $\geq$ 0:
		    (Shifted Dreg left by count) $\rightarrow$ Dreg
		Else:
		    (Shifted Dreg right by |count|) $\rightarrow$ Dreg

ASH3	Arithmetic shift (3-operand)	If count $\geq$ 0:
		    (Shifted src left by count) $\rightarrow$ Dreg
		Else:
		    (Shifted src right by |count|) $\rightarrow$ Dreg

FIX	Convert floating-point value to integer	Fix (src) $\rightarrow$ Dreg
FLOAT	Convert integer to floating-point value	Float(src) $\rightarrow$ Rn

LSH	Logical shift	If count $\geq$ 0:
		    (Dreg left-shifted by count) $\rightarrow$ Dreg
		Else:
		    (Dreg right-shifted by |count|) $\rightarrow$ Dreg

LSH3	Logical shift (3-operand)	If count $\geq$ 0:
		    (src left-shifted by count) $\rightarrow$ Dreg
		Else:
		    (src right-shifted by |count|) $\rightarrow$ Dreg

MPYF	Multiply floating-point values	src $\times$ Rn $\rightarrow$ Rn
MPYF3	Multiply floating-point value (3-operand)	src1 $\times$ src2 $\rightarrow$ Rn
MPYI	Multiply integers	src $\times$ Dreg $\rightarrow$ Dreg
MPYI3	Multiply integers (3-operand)	src1 $\times$ src2 $\rightarrow$ Dreg
NEGB	Negate integer with borrow	0 – src – C $\rightarrow$ Dreg
NEGF	Negate floating-point value	0 – src $\rightarrow$ Rn
NEGI	Negate integer	0 – src $\rightarrow$ Dreg
NORM	Normalize floating-point value	Normalize (src) $\rightarrow$ Rn
RND	Round floating-point value	Round (src) $\rightarrow$ Rn
ROL	Rotate left	Dreg rotated left 1 bit $\rightarrow$ Dreg

Table 2–7. Arithmetic Instruction Summary (Concluded)

Mnemonic	Description	Operation
ROLC	Rotate left through carry	Dreg rotated left 1 bit through carry $\rightarrow$ Dreg
ROR	Rotate right	Dreg rotated right 1 bit $\rightarrow$ Dreg
RORC	Rotate right through carry	Dreg rotated right 1 bit through carry $\rightarrow$ Dreg
SUBB	Subtract integers with borrow	Dreg – src – C $\rightarrow$ Dreg
SUBB3	Subtract integers with borrow (3-operand)	src1 – src2 – C $\rightarrow$ Dreg
SUBC	Subtract integers conditionally	If Dreg – src $\geq$ 0: [(Dreg – src) << 1] OR 1 $\rightarrow$ Dreg Else, Dreg << 1 $\rightarrow$ Dreg
SUBF	Subtract floating-point values	Rn – src $\rightarrow$ Rn
SUBF3	Subtract floating-point values (3-operand)	src1 – src2 $\rightarrow$ Rn
SUBI	Subtract integers	Dreg – src $\rightarrow$ Dreg
SUBI3	Subtract integers (3-operand)	src1 – src2 $\rightarrow$ Dreg
SUBRB	Subtract reverse integer with borrow	src – Dreg – C $\rightarrow$ Dreg
SUBRF	Subtract reverse floating-point value	src – Rn $\rightarrow$ Rn
SUBRI	Subtract reverse integer	src – Dreg $\rightarrow$ Dreg

Table 2–8. Parallel Instruction Set Summary

Mnemonic	Description	Operation
Parallel Arithmetic With Store Instructions		
ABSF || STF	Absolute value of a floating-point	|src2| → dst1 || src3 → dst2
ABSI || STI	Absolute value of an integer	|src2| → dst1 || src3 → dst2
ADDF3 || STF	Add floating-point	src1 + src2 → dst1 || src3 → dst2
ADDI3 || STI	Add integer	src1 + src2 → dst1 || src3 → dst2
AND3 || STI	Bitwise logical-AND	src1 AND src2 → dst1 || src3 → dst2

ASH3
|| STI
	Arithmetic shift	If count ≥ 0:
		    src2 << count → dst1
		  || src3 → dst2
		Else:
		    src2 >> |count| → dst1
		  || src3 → dst2

FIX || STI	Convert floating-point to integer	Fix(src2) → dst1 || src3 → dst2
FLOAT || STF	Convert integer to floating-point	Float(src2) → dst1 || src3 → dst2
LDF || STF	Load floating-point	src2 → dst1 || src3 → dst2
LDI || STI	Load integer	src2 → dst1 || src3 → dst2

LSH3
|| STI
	Logical shift	If count ≥ 0:
		    src2 << count → dst1
		  || src3 → dst2
		Else:
		    src2 >> |count| → dst1
		  || src3 → dst2

MPYF3 || STF	Multiply floating-point	src1 x src2 → dst1 || src3 → dst2
MPYI3 || STI	Multiply integer	src1 x src2 → dst1 || src3 → dst2
NEGF || STF	Negate floating-point	0– src2 → dst1 || src3 → dst2

Table 2–8. Parallel Instruction Set Summary (Concluded)

Mnemonic	Description	Operation
Parallel Arithmetic With Store Instructions (Concluded)		
NEGI || STI	Negate integer	0 – src2 $\rightarrow$ dst1 || src3 $\rightarrow$ dst2
NOT || STI	Complement	$\overline{\text{src1}}$ $\rightarrow$ dst1 || src3 $\rightarrow$ dst2
OR3 || STI	Bitwise logical-OR	src1 OR src2 $\rightarrow$ dst1 || src3 $\rightarrow$ dst2
SUBF3 || STF	Subtract floating-point	src1 – src2 $\rightarrow$ dst1 || src3 $\rightarrow$ dst2
SUBI3 || STI	Subtract integer	src1 – src2 $\rightarrow$ dst1 || src3 $\rightarrow$ dst2
XOR3 || STI	Bitwise exclusive-OR	src1 XOR src2 $\rightarrow$ dst1 || src3 $\rightarrow$ dst2
Parallel Load Instructions		
LDF || LDF	Load floating-point	src2 $\rightarrow$ dst1 || src4 $\rightarrow$ dst2
LDI || LDI	Load integer	src2 $\rightarrow$ dst1 || src4 $\rightarrow$ dst2
Parallel Multiply And Add/Subtract Instructions		
MPYF3 || ADDF3	Multiply and add floating-point	op1 x op2 $\rightarrow$ op3 || op4 + op5 $\rightarrow$ op6
MPYF3 || SUBF3	Multiply and subtract floating-point	op1 x op2 $\rightarrow$ op3 || op4 – op5 $\rightarrow$ op6
MPYI3 || ADDI3	Multiply and add integer	op1 x op2 $\rightarrow$ op3 || op4 + op5 $\rightarrow$ op6
MPYI3 || SUBI3	Multiply and subtract integer	op1 x op2 $\rightarrow$ op3 || op4 – op5 $\rightarrow$ op6
Parallel Store Instructions		
STF || STF	Store floating-point	src1 $\rightarrow$ dst1 || src3 $\rightarrow$ dst2
STI || STI	Store integer	src1 $\rightarrow$ dst1 || src3 $\rightarrow$ dst2

LEGEND:

src1	=	register addr (R7 — R0)	src2	=	indirect addr (disp = 0, 1, IR0, IR1)
src3	=	register addr (R7 — R0)	src4	=	indirect addr (disp = 0, 1, IR0, IR1)
dst1	=	register addr (R7 — R0)	dst2	=	indirect addr (disp = 0, 1, IR0, IR1)
op3	=	register addr (R0 or R1)	op6	=	register addr (R2 or R3)

op1,op2,op4,op5 – Two of these operands must be specified using register addr, and two must be specified using indirect.

2.3 Memory Organization

The total memory space of the TMS320C31 is 16 megawords (32 bits each). Program, data, and I/O space are contained within this 16-megaword address space, allowing tables, program code, or data to be stored in either RAM or ROM. This single address space allows you to maximize the use of the memory space and to partition it as desired.

2.3.1 RAM, ROM, and Cache

Figure 2–3 shows how the memory is organized on the TMS320C31. RAM blocks 0 and 1 are 1K x 32 bits each. Each RAM and ROM block is capable of supporting two CPU accesses in either RAM block. The 'C31 also has an on-chip bootloader ROM, which allows program stored in off-chip memory or transferred through the serial port to be loaded anywhere in the memory map. The separate program buses, data buses, and DMA buses allow parallel program fetches, data reads and writes, and DMA operations. For example, the CPU can access a data value in one RAM block and perform an external program fetch in parallel with the DMA loading another RAM block, all within a single cycle.

A 64 x 32-bit instruction cache is provided to store frequent sections of code, thus greatly reducing the number of off-chip accesses necessary. This allows code to be stored off-chip in slower, lower-cost memories. The external buses are also freed for use by the DMA, external memory fetches, or other devices in the system.
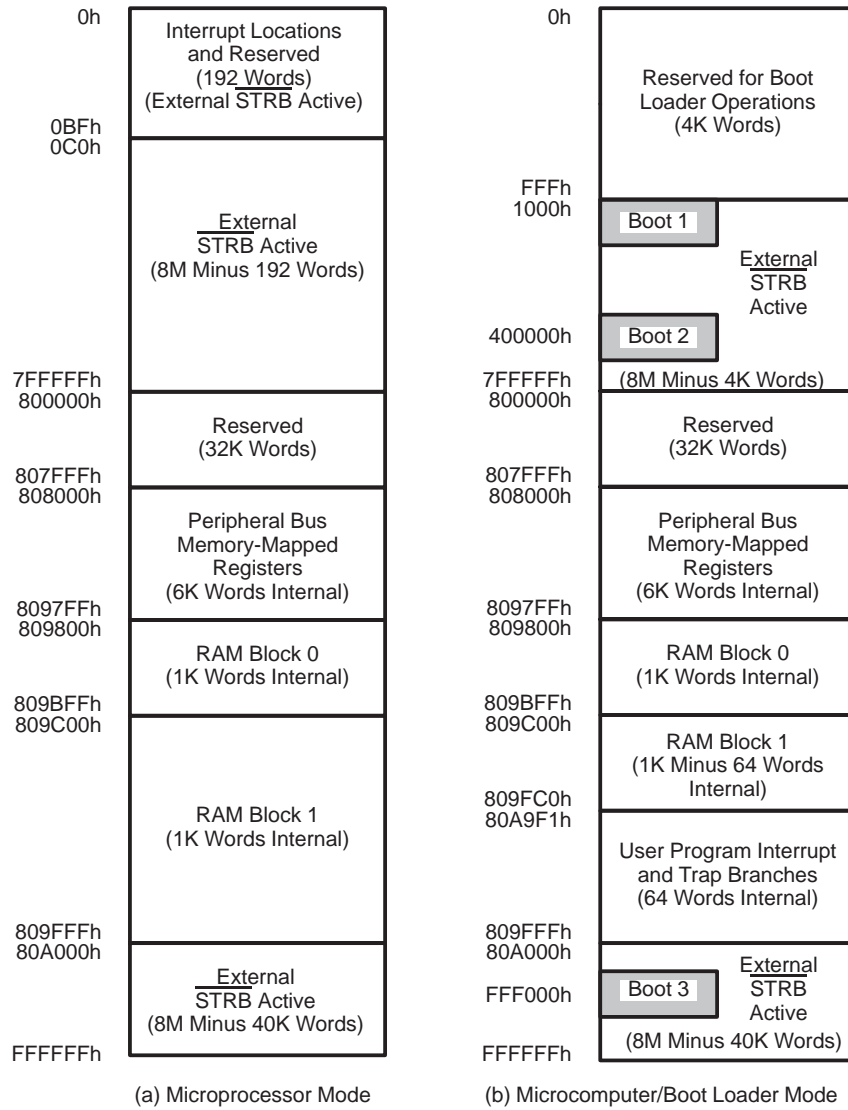
Figure 2–3. Memory Organization
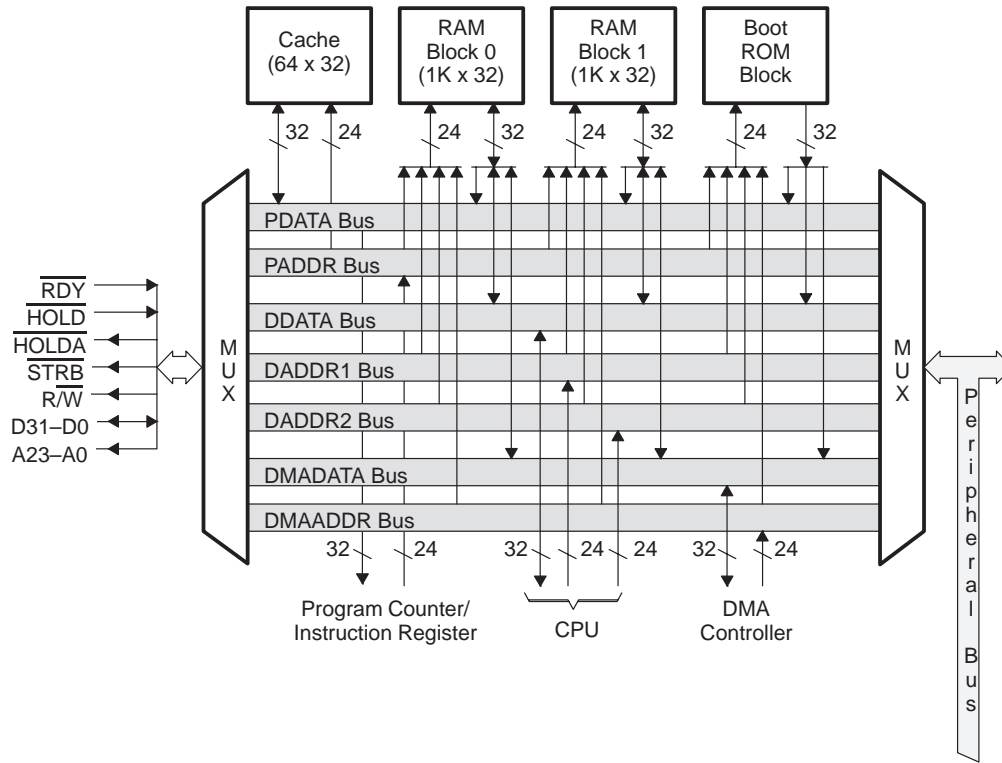
2.3.2 Memory Maps

There are two TMS320C31 memory maps. Use of either one depends on whether the processor is running in the microprocessor mode (MCBL/$\overline{MP}$ = 0) or the bootloader mode (MCBL/$\overline{MP}$ = 1). The memory maps are similar (see Figure 2–4). All of the memory-mapped peripheral registers are in locations 808000h through 8097ffh. In both modes, RAM block 0 is located at addresses 809800 through 809bFFh, and RAM block 1 is located at addresses 809c00 through 809fffh.

In microprocessor mode, the bootloader ROM is not mapped into the TMS320C31 memory map. Locations 0h through 0BFh consist of interrupt vector, trap vector, and reserved locations, all of which are accessed over the external memory port ($\overline{STRB}$ active). Locations 0C0h through 07FFFFFh and locations 80A000h through 0FFFFFFh are also accessed using $\overline{STRB}$.

In bootloader mode, the bootloader ROM is mapped into locations 0h through 0FFFh. There are 192 locations (0h through 0BFh) within this block for the 'C31 bootloader program. Locations 1000h through 07FFFFFh and locations 80A000h through 0FFFFFFh are also accessed using $\overline{STRB}$.

Figure 2–4. TMS320C31 Memory Maps

	(a) Microprocessor Mode		(b) Microcomputer/Boot Loader Mode
0h	Interrupt Locations and Reserved (192 Words) (External STRB Active)	0h	Reserved for Boot Loader Operations (4K Words)
0BFh 0C0h	External STRB Active (8M Minus 192 Words)	FFFh 1000h	Boot 1 / External STRB Active
7FFFFFh 800000h	Reserved (32K Words)	400000h 7FFFFFh 800000h	Boot 2 (8M Minus 4K Words) / Reserved (32K Words)
807FFFh 808000h	Peripheral Bus Memory-Mapped Registers (6K Words Internal)	807FFFh 808000h	Peripheral Bus Memory-Mapped Registers (6K Words Internal)
8097FFh 809800h	RAM Block 0 (1K Words Internal)	8097FFh 809800h	RAM Block 0 (1K Words Internal)
809BFFh 809C00h	RAM Block 1 (1K Words Internal)	809BFFh 809C00h	RAM Block 1 (1K Minus 64 Words Internal)
809FFFh 80A000h	External STRB Active (8M Minus 40K Words)	809FC0h 80A9F1h	User Program Interrupt and Trap Branches (64 Words Internal)
FFFFFFh		809FFFh 80A000h FFF000h FFFFFFh	Boot 3 / External STRB Active (8M Minus 40K Words)

2.4   Internal Bus Operation

A large portion of the TMS320C31's high performance is due to internal busing and parallelism. The separate program buses (PADDR and PDATA), data buses (DADDR1, DADDR2, and DDATA), and DMA buses (DMAADDR and DMADATA) allow for parallel program fetches, data accesses, and DMA accesses. These buses connect all of the physical spaces (on-chip memory, off-chip memory, and on-chip peripherals) supported by the TMS320C31. Figure 2–3 shows these internal buses and their connection to on-chip and off-chip memory blocks.

The program counter (PC) is connected to the 24-bit program address bus (PADDR). The instruction register (IR) is connected to the 32-bit program data bus (PDATA). These buses can fetch a single instruction word every machine cycle.

The 24-bit data address buses (DADDR1 and DADDR2) and the 32-bit data data bus (DDATA) support two data memory accesses every machine cycle. The DDATA bus carries data to the CPU over the CPU1 and CPU2 buses. The CPU1 and CPU2 buses can carry two data memory operands to the multiplier, ALU, and register file every machine cycle. Also internal to the CPU are register buses REG1 and REG2 that can carry two data values from the register file to the multiplier and ALU every machine cycle. Figure 2–2 shows the buses internal to the CPU section of the processor.

The DMA controller is supported with a 24-bit address bus (DMAADDR) and a 32-bit data bus (DMADATA). These buses allow the DMA to perform memory accesses in parallel with the memory accesses occurring from the data and program buses.

2.5 On-Chip Peripherals

All TMS320C31 peripherals are controlled through memory-mapped registers on a dedicated peripheral bus. The peripheral bus is composed of a 32-bit data bus and a 24-bit address bus. The peripheral bus permits straightforward communication to the peripherals. The TMS320C31 peripherals include two timers and one serial port. Figure 2–5 shows the peripherals with associated buses and signals.

Figure 2–5. Peripheral Modules
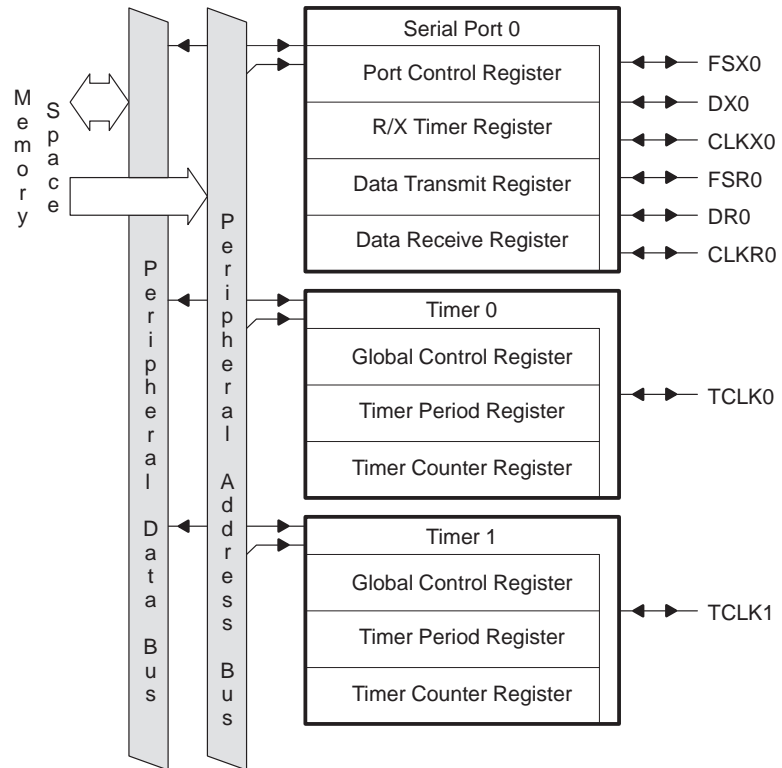
2.5.1 Timers

The two timer modules are general-purpose 32-bit timer/event counters with two signaling modes and internal or external clocking. Each timer has an I/O pin that can be used as an input clock to the timer or as an output signal driven by the timer. The pin may also be configured as a general-purpose I/O pin.

2.5.2 Serial Port

The TMS320C31 offers a full-duplex, synchronous serial port, which can be used as a general system interface or glueless logic connection to an external analog converter. The serial port can be configured to transfer 8, 16, 24, or 32 bits of data per word. The clock for each serial port can originate either internally or externally. An internally generated divide-down clock is provided. The serial port can also be configured as timers or bit I/O pins. A special handshake mode allows the TMS320C31s to communicate via their serial ports with automatic synchronization.

2.6   Direct Memory Access (DMA)

The on-chip DMA controller can read from or write to any location in the memory map without interfering with the operation of the CPU. The DMA controller can be configured to synchronize transfers with external, serial port or timer interrupts. Therefore, the TMS320C31 can interface to slow memories and to on-chip and system peripherals without reducing throughput to the CPU. The DMA controller contains its own address generators, source and destination registers, and transfer counter. Dedicated on-chip DMA address and data buses minimize conflicts between the CPU and the DMA controller for on-chip resources. A DMA operation consists of a block or single-word transfer to or from memory. Figure 2–6 shows the DMA controller with associated buses.

Figure 2–6. DMA Controller
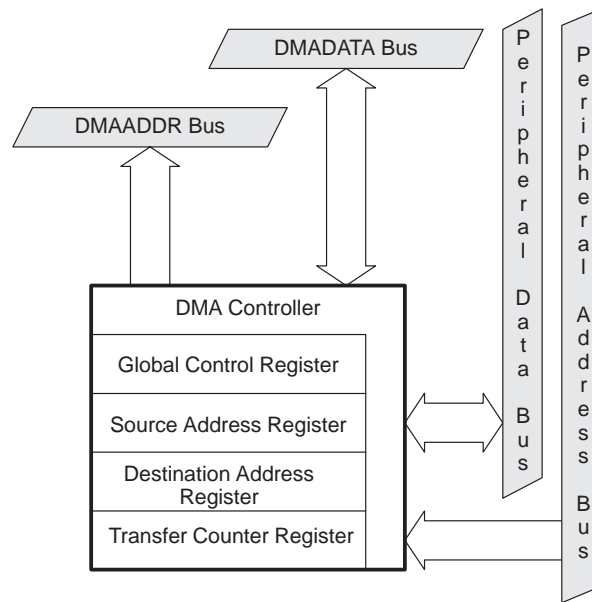
2.7 External Bus Operation

The TMS320C31 primary bus is the external memory interface. The primary bus consists of a 24-bit address bus, 32-bit data bus, and a set of control signals. It can be used to address external program/data memory or I/O space. The bus has an external ready signal ($\overline{\text{RDY}}$), which can be used in conjunction with the on-chip software for controlled wait-state generation. See Table 2–9 for a description of the TMS320C31 external signals.

2.7.1 External Bus Control Features

The TMS320C31 external bus provides flexibility to implement different types of memory systems. The $\overline{\text{STRB}}$ control signal remains active between consecutive read cycles to the same bank of memory, allowing high-speed SRAM and static-column decode accesses. In addition, the primary bus has a programmable bank switching feature, providing more time for address decoding and memory turn-off, when a bank boundary is crossed.

2.7.2 Multiprocessor Support

The TMS320C31 supports shared-memory multiprocessor systems through its $\overline{\text{HOLD}}$ and hold acknowledge ($\overline{\text{HOLDA}}$) signals. When the $\overline{\text{HOLD}}$ input is asserted, the primary bus control, address and data bus signals go into a high-impedance state after the current bus cycle is complete. The $\overline{\text{HOLDA}}$ output acknowledges that the 'C31 primary bus has gone into high-impedance state.

Interlocked operations ease the implementation of multiprocessor operations such as busy-wait loops, shared counter manipulation, and semaphores. The TMS320C31 supports interlocked operations through its XF0 and XF1 pins and dedicated interlocked operation instructions. XF0 and XF1 can also be used as bit I/O signals.

2.8 Interrupts

The TMS320C31 supports four external interrupts ($\overline{\text{INT3}}$–$\overline{\text{INT0}}$), a number of internal peripheral interrupts, 28 software interrupts (traps), and a nonmaskable external $\overline{\text{RESET}}$ signal. The external and internal peripheral interrupts can be used to interrupt either the DMA or the CPU. When the CPU responds to the interrupt, the $\overline{\text{IACK}}$ pin can be used to signal an external interrupt acknowledge. Typical interrupt latency times are less than 1 μs for a 50-ns TMS320C31.

2.9 TMS320C31 Signal Descriptions

Table 2–9 describes the external signals of the TMS320C31. They are listed according to the signal name; the number of pins allocated; the input (I), output (O), or high-impedance state (Z) operating modes; a brief description of the signal's function; and the condition that places an output pin in high impedance. A line over a signal name (for example, $\overline{\text{RESET}}$) indicates that the signal is active low (true at a logic 0 level).

Table 2–9. TMS320C31 Signal Descriptions

Signal	# Pins	I/O/Z†	Description	Condition When Signal Is in High Z‡		
Primary Bus Interface (61 Pins)						
D31–D0	32	I/O/Z	32-bit data port.	S	H	R
A23–A0	24	O/Z	24-bit address port..	S	H	R
R/$\overline{\text{W}}$	1	O/Z	Read/write signal. This pin is high when a read is performed; low when a write is performed over the parallel interface.	S	H	R
$\overline{\text{STRB}}$	1	O/Z	External access strobe.	S	H	
$\overline{\text{RDY}}$	1	I	Ready signal. This pin indicates that the external device is prepared for a transaction completion.			
$\overline{\text{HOLD}}$	1	I	Hold signal. When $\overline{\text{HOLD}}$ is a logic low, any ongoing transaction is completed. The A23–A0, D31–D0, $\overline{\text{STRB}}$, and R/W signals are placed in a high-impedance state, and all transactions over the primary bus interface are held until HOLD becomes a logic high, or the NOHOLD bit of the primary bus control register is set.			
$\overline{\text{HOLDA}}$	1	O/Z	Hold acknowledge signal. This signal is generated in response to a logic low on $\overline{\text{HOLD}}$. It signals that A23–A0, D31–D0, $\overline{\text{STRB}}$, and R/W are placed in a high-impedance state and that all transactions over the bus will be held. HOLDA will be high in response to a logic high of $\overline{\text{HOLD}}$, or the NOHOLD bit of the primary bus control register is set.	S		

† Input (I), output (O), high-impedance (Z) state.
‡ S = SHZ active, H = Hold active, R = Reset active.

Table 2–9. TMS320C31 Signal Descriptions (Continued)

Signal	# Pins	I/O/Z†	Description	Condition When Signal Is in High Z‡	
Control Signals (10 Pins)					
$\overline{\text{RESET}}$	1	I	Reset. When this pin is a logic low, the device is placed in the reset condition. When reset becomes a logic 1, execution begins from the location specified by the re-set vector.		
$\overline{\text{INT3}}$ — $\overline{\text{INT0}}$	4	I	External interrupts.		
$\overline{\text{IACK}}$	1	O/Z	Interrupt acknowledge signal. $\overline{\text{IACK}}$ is active during the IACK instruction. This can be used to indicate the be-ginning or end of an interrupt service routine.	S	
MCBL/$\overline{\text{MP}}$	1	I	Microcomputer boot loader/microprocessor mode pin.		
$\overline{\text{SHZ}}$	1	I	Shut down high Z. An active low shuts down the TMS320C31 and places all pins in a high-impedance state. This signal is used for board-level testing to en-sure that no dual drive conditions occur. **CAUTION:** An active low on the $\overline{\text{SHZ}}$ pin corrupts TMS320C31 memory and register contents. Reset the device with an $\overline{\text{SHZ}}$ = 1 to restore it to a known operating condition.		
XF1, XF0	2	I/O/Z	External flag pins. They are used as general-purpose I/O pins or to support interlocked processor instruc-tions.	S	R
Serial Port 0 Signals (6 Pins)					
CLKR0	1	I/O/Z	Serial port 0 receive clock. This pin serves as the serial shift clock for the serial port 0 receiver.	S	R
CLKX0	1	I/O/Z	Serial port 0 transmit clock. This pin serves as the serial shift clock for the serial port 0 transmitter.	S	R
DR0	1	I/O/Z	Data receive. Serial port 0 receives serial data via the DR0 pin.	S	R
DX0	1	I/O/Z	Data transmit output. Serial port 0 transmits serial data on this pin.	S	R
FSR0	1	I/O/Z	Frame sychronization pulse for receive. The FSR0 pulse initiates the receive data process over DR0.	S	R
FSX0	1	I/O/Z	Frame synchronization pulse for transmit. The FSX0 pulse initiates the transmit data process over pin DX0.	S	R

† Input (I), output (O), high-impedance state (Z).
‡ S = SHZ active, H = Hold active, R = Reset active.

Table 2–9. TMS320C31 Signal Descriptions (Concluded)

Signal	# Pins	I/O/Z†	Description	Condition When Signal Is in High Z‡
Timer Signals (2 Pins)				
TCLK0	1	I/O/Z	Timer clock 0. As an input, TCLK0 is used by timer 0 to count external pulses. As an output pin, TCLK0 outputs pulses generated by timer 0.	S
TCLK1	1	I/O/Z	Timer clock 1. As an input, TCLK0 is used by timer 1 to count external pulses. As an output pin, TCLK1 outputs pulses generated by timer 1.	S
Supply and Oscillator Signals (49 Pins)				
H1	1	O/Z	External H1 clock. This clock has a period equal to twice CLKIN.	S
H3	1	O/Z	External H3 clock. This clock has a period equal to twice CLKIN.	S
V_{DD}	20	I	+5-V_{DC} supply pins. All pins must be connected to a common supply plane. §	
V_{SS}	25	I	Ground pins. All ground pins must be connected to a common ground plane.	
X1	1	O/Z	Output pin from the internal crystal oscillator. If a crystal is not used, this pin should be left unconnected.	S
X2/CLKIN	1	I	The internal oscillator input pin from a crystal or a clock.	
Reserved (4 Pins) ¶				
EMU2 — EMU0	3	I	Reserved. Use 20-kΩ pull-up resistors to +5 volts.	
EMU3	1	O/Z	Reserved.	S

† Input (I), output (O), high-impedance state (Z).
‡ S = SHZ active, H = Hold active, R = Reset active.
§ Recommended decoupling capacitor value is 0.1 μF.
¶ Follow the connections specified for the reserved pins. 18- to 22-kΩ pull-up resistors are recommended. All +5 volt supply pins must be connected to a common supply plane, and all ground pins must be connected to a common ground plane.

TMS320C31 Features/Performance Comparison

This chapter compares the device features and performance of the TMS320C31 to other embedded controllers. The TMS320C31's CPU provides higher system and numeric performance than CISC microprocessors and microcontrollers and also provides higher sustained numeric performance than RISC embedded controllers. The TMS320C31 also incorporates several peripherals on-chip, which helps reduce system cost and complexity. It also possesses a significant amount of on-chip memory, which facilitates the real-time execution of time-critical routines, reducing the need for expensive, high-speed external memory.

The topics discussed include:

3.1 TMS320C31 Feature Comparison Versus Other Embedded Controllers

Table 3–1 lists and describes the fields shown in Table 3–2. Table 3–2 highlights the features and performance of several embedded controllers in the same price range, including the TMS320C31.

Table 3–1. Description of the Fields in Table 3–2

Field Name	Description
Device–MHz	Device part number and speed in MHz
MIPS	Millions of instructions executed per second
No. Buses,	Number of external memory buses
Width	Width of the external data buses
On-Chip RAM (Bytes)	Amount of on-chip program, data and cache memory.
Serial Ports	Number of on-chip serial ports.
Timer	Number of on-chip counter timers.
DMA Chan	Number of DMA controller channels.
Multiply Time (ns) Integ/Float	The time the processor takes to perform a single, nonpipelined integer multiply/ floating point multiply.

Table 3–2. Feature/Performance Comparison of Embedded Controllers

Device–MHz	MIPS	No. Buses, Width	On-Chip RAM (Bytes)	Peripherals			Multiply Time (ns) Integ/Float
				Serial Ports	Timer	DMA Chan	
TMS320C31–27	14	1 , 32	8352	1	2	1	74/74
TMS320C31–33	17	1 , 32	8352	1	2	1	60/60
TMS320C31–40	20	1 , 32	8352	1	2	1	50/50
MC68332–16	2–4	1 , 16	2048	2	5	2	180/NA
MC68331–16	2–4	1 , 16	0	2	4	1	180/NA
i960KA–16	8	1 , 32	512	0	0	0	540/NA
i960KA–25	12	1 , 32	512	0	0	0	360/NA
i960KB–16	8	1 , 32	512	0	0	0	540/660
i960KB–25	12	1 , 32	512	0	0	0	360/440
AMD29005–16	10–16	1 , 32	0	0	1	0	2640/21660
AMD29000–16	10–15	2 , 32	512	0	1	0	2640/21660
AMD29035–16	12–16	1 , 32	4096	0	1	0	1320/10830

Key:

    NA — The device does not support this feature in hardware.

3.2 TMS320C31 Benchmark Performance Versus Other Embedded Controllers

The best method to evaluate a processor's performance in a given application is to benchmark the execution time of the applications software under target system constraints. The next best evaluation method is to benchmark the performance of similar code or code that is representative of the target application. However, due to short product development cycles, the processor evaluation period is rarely long enough to do the code development and system emulation necessary to perform such a rigorous performance analysis for each candidate device. Consequently, many system designers use published device benchmarks to obtain rough performance estimates for different classes of algorithms.

Table 3–3 shows the published manufacturer benchmarks for several C language programs. These benchmarks have been used by processor manufacturers to highlight the general performance of their devices and are a subset of a group of benchmarks referred to as the "Intel Intro Benchmarks". Even though these benchmarks do not necessarily reflect controller performance for many realtime applications, the results are presented here to illustrate that high system-control performance can be achieved with the TMS320C31 using high-level language code. "Intel Intro Benchmarks" results for embedded processors at the same price level as the TMS320C31 are also shown in Table 3–3 to show that the TMS320C31 is a low-cost, high-performance solution relative to other embedded controllers.

Table 3–3. Benchmark Comparison of the TMS320C31 With Embedded Controllers at the Same Price Level

Benchmark (Units)	'C3x[1] 60 ns	AMD29000 60 ns [2] (YARC Board)	i960KA[3] 40 ns	68030[3] 30 ns
Dhrystones/(sec)	*32,237	*24,388	*23,423	*9,049
Bubble-sort (msec)	67.875	122	109	176
Quick-sort (msec)	40.692	95	81.8	173
matmult (msec)	17.192	91.942	45.378	113.062
anneal (sec)	15.120	14.86	12.67	22.552

Notes: 1) The 'C31 benchmarks were run on the Texas Instruments 'C3x application board using zero wait-state SRAM. The C code was compiled using the TMS320 Floating-Point DSP Optimizing C compiler. The benchmarks yield the same results for both the 'C30 and 'C31.

2) AMD29000 results are taken from an AMD application note, *Intel i960CA Benchmark Report Critique* by Tim Olson.

3) The i960KA and 68030 numbers are from the February 1990 issue of Electronic Engineering.

4) An asterisk (*) denotes compiler in-lining of application functions. Without using in-lining, the TMS320C31 provides 24,876 Dhrystones/sec.

3.2.1 Dhrystone Benchmark

The Dhrystone benchmark was originally used to measure device performance and compiler efficiency in typical host CPU integer applications. It does not include input/output or operating system operations. In Table 3–3, the results for Dhrystone version 1.1 are shown due to the widespread availability of processor benchmark results for version 1.1 over later versions of the benchmark.

3.2.2 Bubble- and Quick-Sort Benchmarks

The bubble-sort program performs a bubble sort on an array of elements, and the quick-sort program uses the quick-sorting algorithm to sort an array of elements.

3.2.3 matmult Benchmark

matmult is a routine that multiplies two 7×7 matrices together. The 7×7 matrices are subsets of 8×8 matrices.

3.2.4 anneal Benchmark

anneal solves the travelling salesman's problem—given a number of cities that the salesman wants to visit, find the shortest route to visit all of the cities by visiting each city only once. The problem is solved using simulated annealing techniques.

3.2.5 Benchmark Summary

For the system control benchmarks described above, the TMS320C31 performs at the same level as higher priced devices and overall, outperforms devices at the same price level. For the matmult benchmark, the TMS320C31 offers superior results due to its single-cycle multiply support on-chip. These benchmarks focus on CPU performance and do not reflect that the TMS320C31 possesses more on-chip peripherals than the other processors shown. On-chip peripheral integration reduces system cost and complexity and is an important consideration in embedded controller selection.

Application Examples

This chapter presents four application examples that show how the TMS320C30 and TMS320C31 have been used to integrate system control and signal-processing functions in several application areas. In two of the examples, SPOX, a realtime embedded operating system from Spectron Micro-Systems, is used to facilitate the integration. For more information on SPOX, refer to Chapter 6. The examples discussed are as follows:

4.1   Telecommunications Example Using SPOX

4.1.1   Speech Recognition With TMS320C31 and SPOX

Voice Processing Corp. (VPC) of Cambridge, Massachusetts, a leader in speech recognition technology, develops and markets proprietary technology for speaker-independent continuous and discrete word recognition. VPC has taken an approach to speech recognition that is particularly adept for handling voices over the telephone. Telephone transactions is one area in which speech recognition technology has a compelling market need.

VPC has been supplying speech recognition technology to telecom system manufacturers and over-the-phone service providers for several years, allowing these firms to replace human operators. VPC recognizers are being used in a wide array of applications, such as credit card verification, operator intercept, telephone order entry, and voice-mail.

4.1.2   Lower Cost and More Recognizers

The VPC recognition software requires a high-performance platform that can execute both signal processing and general-purpose algorithms. Since such hardware platforms did not exist on the market, in 1989 VPC developed and built an ISA board with two different processors: the Intel i386 microprocessor and Texas Instruments TMS320C25 signal processor. All of the cycles of the ISA board were needed to execute one speaker-independent speech recognizer in realtime. Since 1989, as their customers required more and more lines of speech recognition to automate over-the-phone services, VPC needed a new hardware platform that could provide more lines of recognizers at a lower cost per line. VPC also needed a more powerful hardware platform to run new recognition algorithms being developed in their research lab.

As VPC engineers saw it, there were two ways to reduce the cost of the speech recognition hardware. They could go to faster hardware that would execute multiple recognizers per chip or they could pack more recognizers onto a single ISA board so they could amortize the board and system cost over more recognizers. They also wanted this new platform to give them more power and flexibility to handle new algorithms. Some of their customers wanted to port different voice functions, such as speech synthesis, to the VPC hardware platform. To ensure that the hardware platform could be easily reprogrammed, VPC wanted to replace their heterogeneous architecture (viz i386 and 'C25) with a homogeneous multiprocessing architecture, which makes it much easier to partition functions across processors. Since the new processor had to take on the functions of both the 386 and 'C25, the support of a multitasking operating system was important.

The VPC criteria for selecting the processor for their next generation platform were as follows:

1) The cost of hardware per recognizer.

2) The number of microprocessors (viz. recognizers) they can incorporate on a board.

3) C compiler and operating system support for pre-emptive multitasking and multiprocessing.

4.1.3   VPRO-4: A Homogeneous Multi-DSP Architecture

The new platform VPC developed, called the VPRO-4, is an ISA board with four 'C31s and a shared-memory architecture. Each 'C31 has 512K bytes of zero-wait-state local memory, and there is 1–8 megabytes of multiported shared memory on the board. All four 'C31s and the PC host can read and write into this shared memory. A robust set of tokens, semaphores, and interrupts facilitates interprocessor communications via software-defined memory structures. Communications with the PC are streamlined by a PC bus I/O-mapped control port which provides for unintrusive polling operations. Realtime voice I/O to a standard voice bus (Dialogic PEB or Natural Microsystems MVIP) is done over the serial port of the 'C31 via an ASIC interface chip.

Figure 4–1. VPRO-4 Hardware Architecture
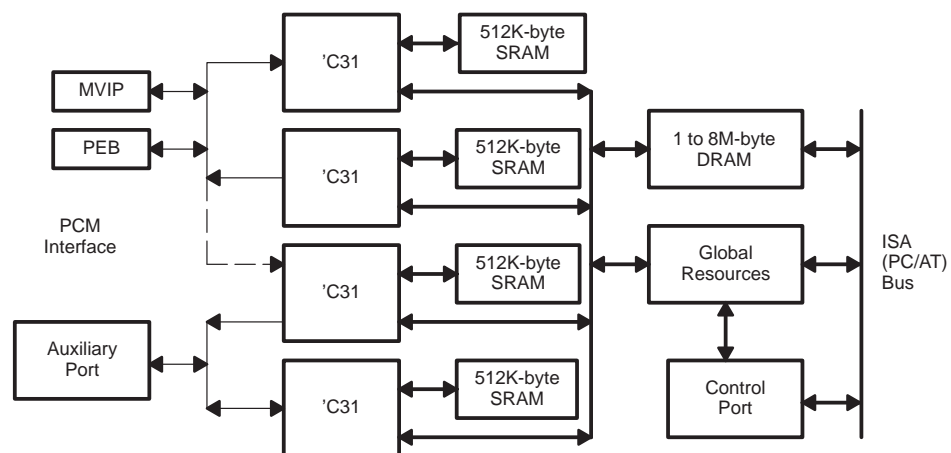
4.1.4   From Tiger 30 to Realtime Recognition

VPC developed software with the Tiger 30 development board from DSP Research. Two discrete word recognizers could run on a single 'C31—eight recognizers on a single ISA board. They also used the board to experiment with SPOX to help them understand its capabilities and performance better.

It took about six months to build the VPRO-4 hardware prototype using the Tiger 30 and SPOX. Because the Tiger 30 board did not interface to the voice bus, they tested their recognizer with canned voice data stored on the host file system. After the VPRO-4 hardware and the necessary low-level software for loading and interfacing to the board was completed, it took just one day to move the SPOX realtime kernel and the recognizer software over to the VPRO-4 hardware.

Each 'C31 on the VPRO-4 runs several tasks using the preemptive multitasking capability of SPOX. A high-priority task moves time-critical voice data to and from the voice bus. The bulk of the 'C31 cycles, however, are used for speech recognition—it runs one recognition task for continuous word input or two recognition tasks for discrete word input. There are also background tasks for communicating with the host and other housekeeping functions.

4.1.5   A New Level of Interoperability

VPC's 'C31-based platform gives them a higher performance system and it lets them serve their customers better. Research continues at VPC to improve the recognition algorithms and take advantage of the processing power of the VPRO-4. In some customer applications, speech recognition has to be complemented with other voice functions, such as speech synthesis. The VPRO-4 makes it easy to port third-party voice algorithms to the DSP platform, significantly reducing total system costs by removing the need for multiple hardware platforms. Other VPC customers have their own 'C31/SPOX hardware. The commonality in the system environment makes it much easier for VPC to port their recognition software to the customer's hardware. This level of interoperability is a significant milestone for speech recognition and signal processing technology. Over-the-phone service providers can now quickly incorporate new voice technology on either VPC's hardware or their own hardware to suit different applications.

4.2 Instrumentation Application and Processor Evaluation Example

4.2.1 Background and System Description

Nicolet Instruments developed the first digital oscilloscope 20 years ago. They have since developed and marketed a variety of other data-acquisition products based on the concept of digitizing analog waveforms. Although they design 8-bit digitizers that collect data at rates of up to 200 million samples/second, their product strength is in the higher-precision, lower-speed digitizers (10 to 16 bits wide, 1–50 million samples/second) with very long memories (greater than 1 million samples). Nicolet's requirements for an embedded processor were low system cost, and high data movement and numeric processing performance.

Figure 4–2 is a block diagram of a typical Nicolet high-precision data-acquisition system using a dual-processor architecture. The master CPU controls the data-acquisition subsystem, which includes the analog converters, digitizer memory and arbitration logic. In the current implementation of this architecture, a CISC processor is used as the master CPU. The slave processor handles high-speed data transfers within, in, and out of the system and performs numeric operations on the digitized data. To perform these operations efficiently, Nicolet wanted a slave processor that would allow low-system cost, and high-data movement and numeric-processing performance using the C language. Nicolet selected the 'C31 due to its balance of price and performance over RISC solutions. In addition, for extremely cost-sensitive designs, Nicolet is considering the 'C31 to integrate the functionality of both the master and slave processors.

Figure 4–2. System Diagram
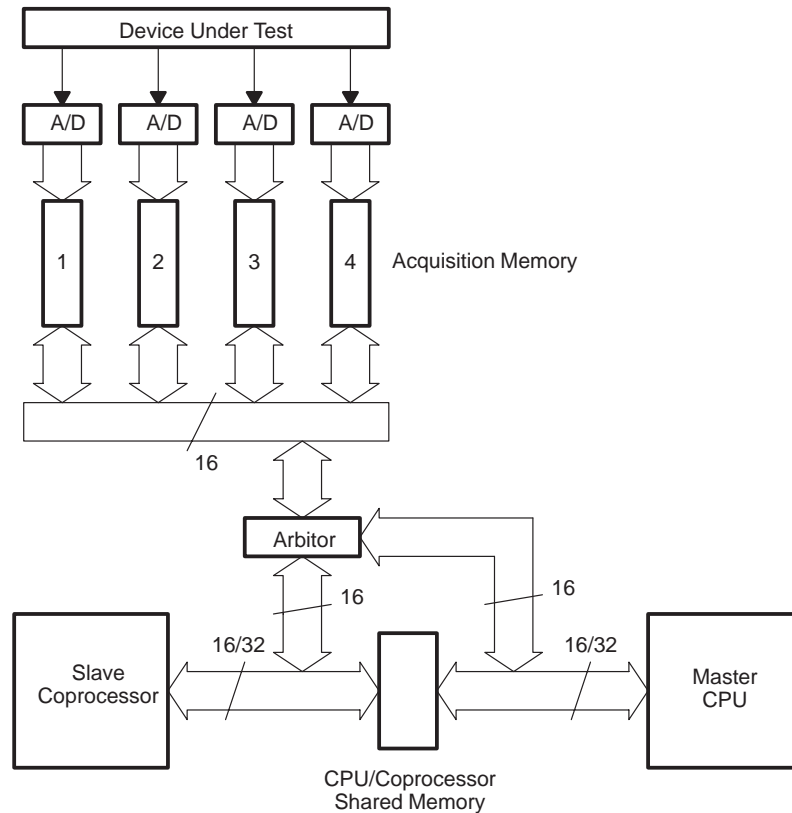
For Nicolet's data-acquisition equipment, the processor must move data and complete calculations in realtime, and also have enough performance to display the information in a reasonable amount of time. To fulfill these requirements, the slave processor needed the following characteristics:

❑ High data-movement rate

❑ Fast address-generation capability

❑ Realtime calculation of waveform pulse parameters

❑ Floating-point Fast-Fourier transformation (FFT) of input samples to enable the frequency domain display of the data

❑ Performance of other realtime DSP operations including filtering, correlation, and convolution

The importance of these device characteristics is illustrated in some of the algorithms Nicolet uses in its data acquisition equipment—archive shuffle, waveform processing and FFT.

4.2.2   Archive Shuffle

When Nicolet's equipment digitizes a waveform, the trigger or start point is not necessarily at the first location in digitizer memory. The archive shuffle algorithm moves the trigger point to the first location without using additional data memory (in-place data movement). Even though the archive shuffle algorithm did not take advantage of a DMA controller, the 'C31 is efficient at performing the data shuffle due to its single cycle instructions and auxiliary register arithmetic units, which can generate two pointer addresses every instruction cycle. Nicolet modified the algorithm to use the 'C31's on-chip DMA to move blocks of data in and out of the 'C31, in parallel with the 'C31 CPU calculating the source and destination addresses of subsequent blocks. With the use of the DMA, Nicolet estimated that the time required to shuffle a block of data was reduced to 35% of the time required for the non-DMA implementation.

4.2.3   Waveform Processing

Waveform processing involves calculating waveform parameters such as area, rise time, root-mean-square (RMS), and standard deviation. The waveform processing must be performed on 1K samples fast enough to allow 5–10 user-screen updates/second. The 'C31 provided more than enough performance to meet the screen update requirements. With its single-cycle multiply capability, the 'C31 especially excels in operations that require multiplies in the inner loop. In addition to the on-chip hardware math support, the 'C31 performs the waveform calculations quickly due to its 2K words of on-chip, general-purpose memory and on-chip program cache.

4.2.4   Fast Fourier Transform

The requirements for the floating-point FFT are similar to those for waveform processing. The processor must perform a 1K FFT fast enough to allow 10 screen updates/second. The 'C31 FFT performance far exceeded the user-update requirement. And if greater FFT performance was needed, Nicolet observed that they could use the C-callable, hand-optimized assembly-language FFT routines available from Texas Instruments. This is not an option with many RISC processors.

4.2.5   Advantages of a TMS320C31 System

Nicolet explained their choice of a 'C31 as the embedded processor with the following comments:

1) The 'C31 offers a good balance of data movement and numeric performance for the price.

2) The 'C31's performance is on par with more expensive processors, making many of the extra-cost product options either no-cost options or extra-margin options.

3) The 'C31 is very efficient at accessing arrays of data due to its ability to do auto-increment indirect addressing.

4) The majority of their code consists of small loops, which makes good use of the 'C31's on-chip instruction cache.

5) The 'C31 allows the user to implement algorithms using either floating-point or integer math, while achieving the same performance with either data format.

6) C callable, optimized DSP algorithms are available for the 'C31.

7) Code development is not required to build a software monitor for the 'C31. A target monitor plugs directly into the target system's 'C31.

8) The 'C31 has a clear family road-map for higher performance with the availability of the 'C3x and 'C4x generations of TMS320s.

4.3 Test Equipment Example Using SPOX

Developed by Doble Engineering in the 1930s, the Doble test is run routinely by power utility companies to test insulation material used in power substations. Over time, the electrical insulation material can break down and can lead to severe damage to the substation and interruptions to service if the problems go undetected. The insulation test procedure involves applying an alternating voltage across the material specimen and a reference sample. The electrical current, capacitance, dielectric-loss, and power factor across the test specimens are measured and analyzed in realtime. To make the test procedure practical, Doble has designed their equipment to be quick and easy to operate and able to make accurate measurements in the presence of a high level of electrical interference.

Figure 4–3. Doble Test Set-Up

Power factor = $\cos \theta$

E = Test Voltage
C_P = Equiv Parallel Capacitor
R_P = Equiv Parallel Resistor
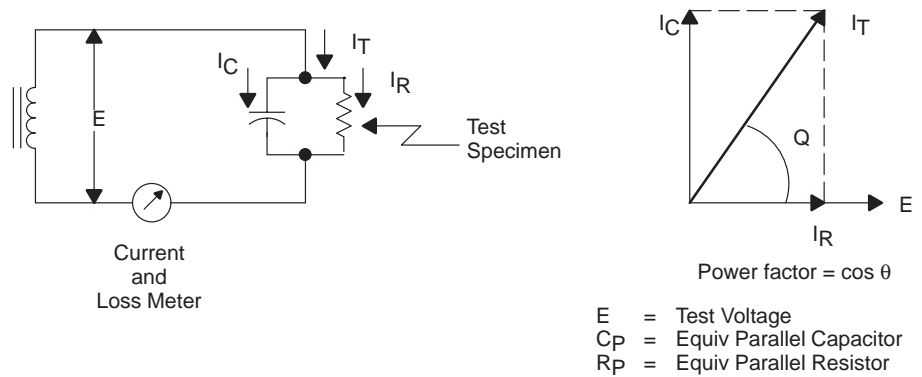
Doble Engineering upgraded their M-series test system from an all-analog design to an all-digital design to reduce production cost, provide portability, increase accuracy, and provide expert advice to the operator. Elegantly simple, the new system consists of an IBM-compatible PC-AT with an attached DSP. The DSP replaces the analog signal processing hardware and executes proprietary signal processing algorithms which produce more accurate measurements. The PC host serves as an expert system and provides a graphical user interface (GUI) complete with dials and meters for operator ease.

Figure 4–4. The New Doble M Series System
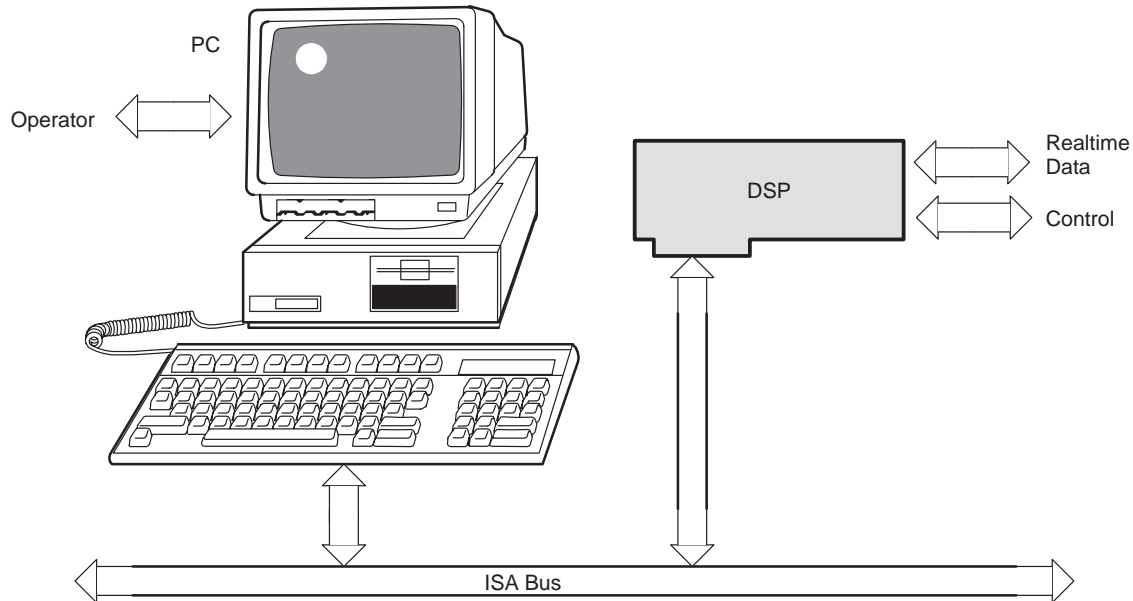
4.3.1 TMS320C30 and SPOX—Merging DSP and Control

By building an experimental DSP-based system using a fixed-point DSP, Doble began the transition from analog to digital technology. Because the DSP lacked many general-purpose functions and was difficult to program, they used it as a black box to replace the analog circuitry that performed filtering and modulation. The DSP code was kept short and was written entirely in assembly language. Realtime I/O and instrument control were performed with an existing attached microprocessor board (with an Intel 80186) running a commercial realtime operating system. Problems with this black-box approach indicated that what Doble needed was a more programmable DSP platform that could handle both signal processing and realtime instrument control.

When Doble went from the experimental system to a production system, their engineers evaluated six DSPs. The TMS320C30 offered a general-purpose architecture that could perform both realtime control and signal-processing functions. The floating-point arithmetic capability made data analysis easier because it guaranteed sufficient accuracy in the analysis algorithms over a wide dynamic range. Doble engineers also evaluated C compilers for the 'C30 and other DSPs—the 'C30 C compiler clearly generated better code. When they learned of the Spectron Microsystems SPOX operating system, they were ready to revise the architecture of the system: the realtime I/O and instru-

ment-control functions of the 80186 and the traditional signal-processing functions of the fixed-point DSP would be performed by the TMS320C30. Using SPOX would also allow Doble to use an object-oriented approach to all of their software development and help them make their code maintainable and easy to modify.

4.3.2   From Proof-of-Concept to the Final Product

To validate this new architecture, Doble purchased the Sonitech Spirit 30 development board for the PC. Because SPOX had already been ported to the Sonitech board, Doble completed a prototype of the new system in two months. Doble then ported SPOX to their customer's 'C30 platform using the SPOX-OS component product. This effort involved reconfiguring SPOX and writing a few device drivers for data I/O and host I/O. Because the two hardware platforms had the same SPOX system software, almost all of the prototype code was reused in the product.

While all of Doble's DSP code had been written in assembly language, the TMS320C30 was programmed in C using the SPOX realtime kernel and math library. Because the SPOX math library had been coded by Spectron in assembly language, the signal-processing algorithms ran efficiently, using only about 50% of the 'C30 cycles. This left enough cycles to perform realtime control functions and new signal processing algorithms. Because the resultant DSP software architecture was more modular, new functions could be added or changed easily. The multitasking capability of SPOX allowed math functions to run concurrently as the DSP acquired data in realtime and communicated with the PC host. Because of the flexibility of the DSP platform, Doble planned to provide different services and products to their customers using the same platform.

Development Support

Throughout the design of the TMS320C3x DSPs, hardware and software engineers worked with device architects to create a processor ideally suited to today's development tool technologies. The result is a full set of hardware and software tools. From the friendly Programmers Interface to TI's unique scan-based emulator, the development environment makes the design of embedded systems fast and easy.

This chapter provides an overview of the development support products supporting TMS320C3x design.

Note:

A floating-point compiler, assembler, and linker support the TMS320C31, TMS320C30, TMS320C40, and all future spin-offs of the 'C3x and 'C4x generations. Complete support for all 32-bit TMS320 processors provides an efficient upgrade path without requiring the purchase of additional compilers, assemblers, or linkers. Throughout this chapter, this compiler will be referred to as the *TMS320C31 compiler;* the assembler/linker will be referred to as the *TMS320C31 assembler/linker*.

5.1   TMS320C3x Optimizing ANSI C Compilers

Fast code development and code maintenance over the life of a product are concerns that all developers share. TI supports embedded system developers with an optimizing compiler for the TMS320C31, which translates ANSI-standard, C language files into highly efficient TMS320C31 assembly language source files, which are then input to a TMS320C31 assembler/linker. The compiler has been validated for conformance to the ANSI C specification, using the industry-standard, Plum-Hall test suite.

The TMS320C31 compiler is complemented by the standard TMS320 Programmers Interface for debugging C and assembly source code. The C compiler produces a rich set of debugging information, which is used by the debugger, allowing source-level debugging in C. This enhances productivity and shortens the development cycle for embedded system designers.

Key features include:

❑ Complete and exact conformance with the ANSI C specification.

❑ Highly efficient code. The compiler incorporates state-of-the-art generic and target-specific optimizations (described in detail within the succeeding subsections). The TMS320C31 compiler performs both global optimizations and loop optimizations such as strength reduction. Additionally, it thoroughly analyzes code in order to optimize the usage of memory and register variables.

❑ ANSI-standard runtime-support library.

❑ ROM-able, relocatable, and re-entrant code.

❑ The ability to link C programs with assembly language routines, allowing hand coding of time-critical functions in assembly language.

❑ A full-featured, flexible linker that allows total control over memory allocation, memory configuration, and partial linking and contains features that allow easy runtime relocation of code.

❑ A C shell program that facilitates one-step translation from C source to executable code.

❑ Fast compilation to increase productivity.

❑ Unlimited symbol table space (up to the amount of available host memory).

❑ Complete and useful diagnostics (error messages).

❑ An archiver utility that allows you to collect files into a single archive file or library by adding new files or by extracting, deleting, or replacing files. You can use a library of object files as input to the linker.

❑ Ability to expand in-line both runtime-support and user-defined functions.

❑ A utility that builds object libraries from source libraries.

❑ A variety of listing files, including:

   ■ Assembly-source file, which can optionally include interlisted, C-source code as well as register-usage information.

   ■ Preprocessed output file useful for separating preprocessing/parsing (if memory limitations dictate) and for troubleshooting macro definitions.

   ■ Assembly-listing file with line numbers and opcodes.

❑ A big memory model with unlimited space for global data, static data, and constants. In the small (default) model, this space is limited to 64K words for faster, more efficient coding/execution.

5.1.1 TMS320C31 Compiler Optimizations

The efficiency of a C compiler depends upon the scope and number of optimizations the C compiler performs, as well as upon the application. The TMS320C31 compiler performs a wide variety of optimizations to improve the efficiency of the compiled code. The following list and explanations that follow describe some of the optimizations and highlight particular strengths of the C compilers.

❑ General-Purpose C Optimizations

   ■ Algebraic reordering, symbolic simplification, constant folding

   ■ Alias disambiguation

   ■ Data flow optimizations

      ▪ Copy propagation
      ▪ Common subexpression elimination
      ▪ Redundant assignment elimination

   ■ Branch optimizations/control-flow simplification

   ■ Loop induction variable optimizations, strength reduction

   ■ Loop rotation

■ Loop-invariant code motion

■ In-line expansion of function calls

❑ Optimizations Specific to the TMS320C31 compiler

■ Register variables

■ Register tracking/targeting

■ Cost-based register allocation

■ Autoincrement addressing modes

■ Repeat blocks

■ Delayed branches

■ Use of registers for passing function arguments

■ Parallel instructions

■ Conditional instructions

■ Loop unrolling

5.1.1.1 General-Purpose Optimizations

Algebraic Reordering, Symbolic Simplification, Constant Folding

For optimal evaluation, the compiler simplifies expressions into equivalent forms requiring fewer instructions or registers. For example, the expression **(*a* + *b*) − (*c* + *d*)** requires more instructions and registers to evaluate than the equivalent expression **((*a* + *b*) − *c*) − *d*.** Operations between constants are folded into single constants. For example, ***a* = (*b* + 4) − (*c* + 1)** becomes ***a* = *b* − *c* + 3**. See Figure 5–1.

Alias Disambiguation

Programs written in C generally use many pointer variables. Frequently, compilers are unable to determine whether or not two or more l (lower case L) values (symbols, pointer references, or structure references) refer to the same memory location. This aliasing of memory locations often prevents the compiler from retaining values in registers, because it cannot be sure that the register and memory continue to hold the same values over time. Alias disambiguation is a technique that determines when two pointer expressions cannot point to the same location, allowing the compiler to freely optimize such expressions.

Data Flow Optimizations

Collectively, the following three data flow optimizations replace expressions with less costly ones, detect and remove unnecessary assignments, and avoid

operations that produce values already computed. The compiler performs these data flow optimizations both locally (within basic blocks) and globally (across entire functions). See Figure 5–1 and Figure 5–2.

❑ Copy Propagation

Following an assignment to a variable, the compiler replaces references to the variable with its value. The value could be another variable, a constant, or a common subexpression. This may result in increased opportunities for constant folding, common subexpression elimination, or even total elimination of the variable.

❑ Common Subexpression Elimination

When the same value is produced by two or more expressions, the compiler computes the value once, saves it, and reuses it.

❑ Redundant Assignment Elimination

Often, copy propagation and common subexpression elimination optimizations result in unnecessary assignments to variables (variables with no subsequent reference before another assignment or before the end of the function). The compiler removes these dead assignments.

Figure 5–1. Data Flow Optimizations for TMS320C31 Compilers

```
    simp(int j)
    {
        int a = 3;
        int b = (j * a) + (j * 2);
        int c = (j << a);
        int d = (j >> 3) + (j << b);

        call(a,b,c,d);
        ...
    }
```

TMS320C31 compiler output is:

```
    _simp:
    *
    * RC   is allocated to user var 'j'
    * RS   is allocated to temp var 'T$2'
    * RE   is allocated to temp var 'T$1'
    *
            ...
            LDI     2,R0              ; (j*a + 2j) == (3j + 2j) == (5j) ==(4j + j)
            LSH     R0,RC,R1         ; R1 = (4j) == (j << 2)
            ADDI    R1,RC,RE         ; b = (4j + j) == 5j
            LDI     3,R1             ; load shift count
            LSH     R1,RC,RS         ; c = (j << a) == (j << 3)
            LSH     RE,RC,R2         ; R2 = (j << b)
            ADDI    RS,R2,R3         ; R3 = (j << b) + (j << a)
            PUSH    R3               ; push R3 (d)
            PUSH    RS               ; push c
            PUSH    RE               ; push b
            PUSH    R1               ; push a (tracked in R1)
            CALL    _call
            ...
```

*The constant **3**, assigned to **a**, is copy-propagated into all uses of **a**. **a** becomes a dead variable and is removed completely. The sum of multiplying **j** by **3 (a)** and **2** is simplified into a multiply by **5,** which is computed with a shift and add. The expression (**j** << **a**) is computed once for assignment to **c** and then reused for calculating **d**. These optimizations are also performed across jumps.*

Branch Optimizations, Control-Flow Simplification

The compiler analyzes the branching behavior of a program and rearranges the linear sequences of operations (basic blocks) to remove branches or redundant conditions. Unreachable code is deleted, branches to branches are bypassed, and conditional branches over unconditional branches are simplified to a single conditional branch. When the value of a condition can be determined at compile time (through copy propagation or other data flow analysis), a conditional branch can be deleted. Switch case lists are analyzed in the

same way as conditional branches and are sometimes eliminated entirely. Some simple, control-flow constructs can be reduced to conditional instructions, totally eliminating the need for branches. See Figure 5–2.

Loop Induction Variable Optimizations, Strength Reduction

Loop induction variables are variables whose value within a loop is directly related to the number of executions of the loop. Array indices and control variables of **FOR** loops are very often induction variables. Strength reduction is the process of replacing costly expressions involving induction variables with more efficient expressions. For example, code that indexes into a sequence of array elements is replaced with code that increments a pointer through the array. Loops controlled by incrementing a counter are written as repeat blocks, or by using efficient decrement-and-branch instructions. Induction variable analysis and strength reduction together often remove all references to the programmer's loop control variable, allowing it to be eliminated entirely.

Loop Rotation

The compiler evaluates loop conditionals at the bottom of loops, saving a costly extra branch out of the loop. In many cases, the initial entry conditional check and the branch are optimized out.

Loop-Invariant Code Motion

This optimization identifies expressions within loops that always compute the same value. The computation is moved in front of the loop, and each occurrence of the expression in the loop is replaced by a reference to the precomputed value.

In-Line Expansion of Function Calls

The special keyword *inline* directs the compiler to replace calls to a function with in-line code, saving the overhead associated with a function call as well as providing increased opportunities to apply other optimizations. See Figure 5–2 and Figure 5–3.

Figure 5–2. Copy Propagation and Control-Flow Simplification for TMS320C31 Compilers

```
fsm()
{
    enum { ALPHA, BETA, GAMMA, OMEGA } state = ALPHA;
    int *input;

    while (state != OMEGA)
        switch (state)
        {
            case ALPHA: state = ( *input++ == 0 ) ? BETA   : GAMMA;    break;
            case BETA : state = ( *input++ == 0 ) ? GAMMA  : ALPHA;    break;
            case GAMMA: state = ( *input++ == 0 ) ? GAMMA  : OMEGA;    break;
        }
}
```

TMS320C31 compiler output is:

```
_fsm:
*
* AR4       is allocated to user var 'input'
*
      LDI     *AR4++,R0  ; initial state == ALPHA.
      BZ      L4         ;    if input == 0     goto state BETA
      B       L12        ;    else          goto state GAMMA
L9:   LDI     *AR4++,R0  ; state == ALPHA.
      BNZ     L12        ;    if input != 0     goto state GAMMA
L4:   LDI     *AR4++,R0  ; state == BETA.
      BNZ     L9         ;    if input != 0     goto state ALPHA
      LDI     *AR4++,R0  ; state == GAMMA.
      BNZ     EPI0_1     ;    if input != 0     goto state OMEGA
L12:  LDI     *AR4++,R0  ; state == GAMMA.
      BZ      L12        ;    if input == 0     goto state GAMMA
EPI0_1:...              ; state == OMEGA.
     ...
```

The switch statement and the state variable from this simple finite-state machine process are optimized completely away, leaving a streamlined series of conditional branches.

Figure 5–3. In-Line Function Expansion for TMS320C31 Compilers

```
     inline blkcpy (char *to, char *from, int n)
     {   if (n > 0 )
            do *to++ = *from++; while (--n !=0);
     }
     struct s { int a,b,c[10]; } s;
     initstr (struct s *ps, char t[12])
     {   blkcpy((char *)ps, t, 12);
     }
```
TMS320C31 compiler output is:
```
     _initstr
   * R2   assigned to variable 't'
   * AR2  assigned to variable 'blkcpy_1_to'
   * AR4  assigned to variable 'blkcpy_1_from'
   * BK   assigned to variable 'ps'
   * RC assigned to variable 'L$1'

         LDI       BK,AR2       ;blkcpy_1_to = ps
         LDI       R2,AR4       ;blkcpy_1_from = t
         LDI       *AR4++,R0    ;+-----------------------------------
         RPTS      10           ;| expansion of blkcpy:
         STI       R0,*AR2++    ;| copy 12 words
    ||   LDI       *AR4++,R0    ;+-----------------------------------
         STI       R0,*AR2++    ;
     ...
```

*The special in-line declaration of **blkcpy** results in the call being replaced with the function's body. The compiler creates temporary variables **blkcpy_1_to** and **blkcpy_1_from**, corresponding to the parameters of **blkcpy**. Often, copy propagation can eliminate assignments to such variables when the argument expressions are not reused after the call.*

5.1.1.2   Optimizations Specific to the TMS320C31 Compiler

Register Variables

The compiler helps maximize the use of registers for storing local variables, parameters, and temporary values. Variables stored in registers can be accessed more efficiently than variables in memory. This optimization is particularly effective for pointers that arise when array index constructs are turned into loop induction variables. See Figure 5–4 and Figure 5–5.

Figure 5–4. Register Variables and Register Tracking/Targeting

```
    int gvar;
    reg(int i, int j)
    {
        gvar = call() & i;
        j    = gvar + i;
    }
```
TMS320C31 compiler output is:
```
    _reg:
    *
    *R4 is allocated to user var 'i'
    *R5 is allocated to user var 'j'
    *
        ...
        CALL        _call          ;R0 = call()
        AND         R4,R0          ;R0 &= i
        STI         R0,@_gvar      ;gvar = R0
        ADDI        R4,R0,R5       ;tracks gvar in R0,
        ...                        ;targets result into R5 (j)
```
The compiler allocates local variables ***i*** *and* ***j*** *into registers* ***R4*** *and* ***R5***,
as indicated by the comments in the assembly listing. Allocating ***i*** *to*
R4 and tracking ***gvar*** *in R0 allows the sum* ***gvar + i*** *to be computed*
with a 3-operand instruction, targeting the result directly into ***j*** *in R5.*

Register Tracking/Targeting

The compiler tracks the contents of registers so that it avoids reloading values if they are used again soon. Variables, constants, and structure references such as (***a.b***) are tracked through both straight-line code and forward branches. The compiler also uses register targeting to compute expressions directly into specific registers when required, as in the case of assigning to register variables or returning values from functions. See Figure 5–4.

Cost-Based Register Allocation

The compiler, when enabled, allocates registers to user variables and compiles temporary values according to their type, use, and frequency. Variables used within loops are weighted to have priority over others, and those variables whose uses don't overlap may be allocated to the same register. Variables with specific requirements are allocated into registers that can accommodate them.

Autoincrement Addressing Modes

For pointer expressions of the form **p++*, **p– –*, **++p*, or **– –p*, the compiler uses efficient TMS320C31 autoincrement addressing modes. In many cases, where code steps through an array in a loop, such as **for** *(i = 0; i < N; ++i)* *a[i]...*, the loop optimizations convert the array's references to indirect references through autoincremented register variable pointers. See Figure 5–5.

Repeat Blocks

The TMS320C31 compiler supports zero-overhead loops with the RPTS (repeat single) and RPTB (repeat block) instructions. The compiler can detect loops controlled by counters and generate them by using the efficient repeat forms: RPTS for single-instruction loops, or RPTB for larger loops. For both forms, the iteration count can be either a constant or an expression. See Figure 5–3 and Figure 5–5.

Induction variable elimination and loop test replacement allow the compiler to recognize the loop as a simple counting loop and then generate a repeat block. Strength reduction turns the array references into efficient pointer autoincrements.

Figure 5–5. Repeat Blocks, Autoincrement Addressing Modes, Parallel Instructions, Strength Reduction, Induction Variable Elimination, Register Variables, and Loop Test Replacement for Floating-Point Compilers

```
    float a[10], b[10];
    scale(float k)
    {
        int i;
        for (i = 0; i < 10; ++i)
            a[i] = b[i] * k;
        ...
```

TMS320C31 compiler output is:

```
    _scale:
        ...
        LDI       @CONST+0,AR4       ; AR4 = &a[0]
        LDI       @CONST+1,AR5       ; AR5 = &b[0]
        MPYF      R4,*AR5++,R0       ; compute first product
        RPTS      8                  ; loop for next 9
        STF       R0,*AR4++          ; store this product...
    ||  MPYF      R4,*AR5++,R0       ; ...and compute next
        STF       R0,*AR4++          ; store last product
        ...
```

This process shows general and floating-point-specific optimizations working together to generate highly efficient code. Induction variable elimination and loop test replacement allow the compiler to recognize the loop as a simple counting loop and then generate a repeat block. Strength reduction turns the array's references into efficient pointer autoincrements. The compiler unrolls the loop once to separate the first multiply and last store, allowing the body of the loop to be written as a single parallel instruction.

Delayed Instructions

The TMS320C31 compiler supports delayed branch instructions that can be inserted three instructions early in an instruction stream, avoiding costly pipeline flushes associated with normal branches. The compiler uses unconditional delayed branches wherever possible, and conditional delayed branches for counting loops. See Figure 5–6.

Figure 5–6. TMS320C31 Compiler Delayed Branch Optimizations

```
        wait(volatile int *p)
        {
                for(;;)
                  if (*p & 0x80) *p |= 0xF0;
        }
```

TMS320C31 compiler output is:

```
        _wait:
        L6:
                LDI  *AR4,R0      ; R0 = *p (AR4 is allocated to p)
                TSTB 128,R0       ; test *p & 0x80
                BZ   L6           ; false: loop back
                BD   L6           ; true: loop back (delayed)
                LDI  *AR4,R0      ; R0 = *p
                OR   0f0h,R0      ; R0 = *p | 0xF0
                STI  R0,*AR4      ; *p = R0
        ***     B    L6           ; branch occurs
```

*The unconditional branch at the bottom of this loop is written as a delayed branch,
allowing it to execute in one machine cycle.*

Use of Registers for Passing Function Arguments

The compiler supports a new, optional calling sequence that passes arguments to registers rather than pushing them onto the stack. This can result in significant improvement in performance, especially if calls are important in the application. See Figure 5–2.

Parallel Instructions

Several floating-point or integer instructions such as load/load, store/operate, and multiply/add can be paired with each other and executed in parallel. When adjacent instructions match the addressing requirements, the compiler combines them in parallel. Although the code generator performs this optimization, the optimizer greatly increases effectiveness because operands are more likely to be in registers. See Figure 5–3 and Figure 5–5.

Conditional Instructions

The load instructions in the 'C31 C compiler can be executed conditionally. For simple assignments such as **a = condition ? expr1 : expr2** or **if (condition) a = b**, the compiler can use conditional loads to avoid costly branches.

Loop Unrolling

When the compiler can determine that a short loop is executed a low, constant number of times, it replicates the body of the loop rather than generating the loop; note that **low** and **short** are subjective judgments made by the compiler. This avoids any branches or use of the repeat registers. See Figure 5–7.

Figure 5–7. Loop Unrolling

```
add3(int a[3])
{
    int i, sum = 0;
    for (i = 0; i < 3; ++i) sum += a[i]

    return sum;
}
```
TMS320C31 compiler output is:

```
_add3:
        ...
        LDI        *-FP(2),AR4   ; AR4 = &a[0]
        LDI        *AR4++,RC     ; sum += a[0]
        ADDI       *AR4++,RC     ; sum += a[1]
        ADDI       *AR4++,RC     ; sum += a[2]
        LDI        RC,R0         ; return sum
        ...
```

The compiler determines that this loop is short enough to unroll, resulting in a simple 3-instruction sequence and no branches.

5.2 TMS320 Programmer's Interface (C/Assembly Source Debugger)

The TMS320 Programmer's Interface brings new levels of power and flexibility to embedded systems development. The interface/debugger is now available on virtually all TMS320 development tools, so moving to another tool or another generation of processor is greatly simplified.

The debugger is an advanced software interface that runs on a PC and supports TI's unique, scan-based, realtime, TMS320C3x XDS emulator. The debugger provides complete control over programs written in C or assembly language.

The debugger improves productivity by enabling you to debug a program in the language in which it is written. Programs can be debugged in C, assembly language, or both. The debugger also has profiling capabilities that show where to focus development time by quickly identifying the "hot" or time-consuming sections of a program.

Figure 5–8. The Basic Debugger Display
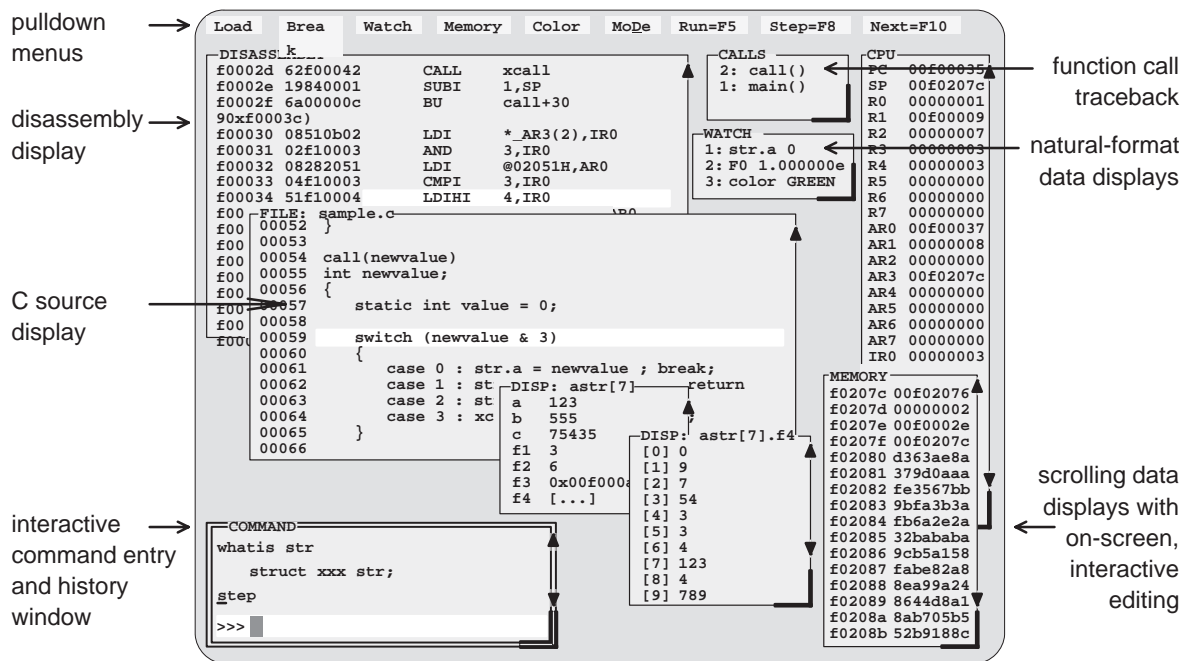
The debugger is easy to learn and use. Its window-/mouse-/menu-oriented interface reduces learning time and eliminates the need to memorize complex commands. The debugger's customizable displays and flexible command entry let you develop a debugging environment that suits the system's needs

(see Figure 5–8). A shortened learning curve and increased productivity reduce the software development cycle, speeding products to market.

Conditional execution and single-stepping (including single-stepping into and over function calls) give you complete control over program execution. A breakpoint can be set or cleared with a click of the mouse or by typing commands. A memory map identifies the portions of target memory that the debugger can access and that can be defined. You can load only the symbol tables' portion of an object file to work with systems that have code in ROM. The debugger can execute commands from a batch file, providing an easy method for entering often-used command sequences. Key features include:

❑ **Multilevel debugging**. The debugger allows you to debug both C and assembly language code. While debugging a C program, you can choose to view the C source, the disassembly of the object code created from the C source, or both.

❑ **Fully configurable, state-of-the-art, window-oriented interface**. The debugger separates code, data, and commands into manageable information. You can select from several displays. Or, since the debugger's display is completely configurable, you can create the interface that best suits the application. The display's colors, physical appearance of displayed features (such as window borders), and window size and position can be changed.

❑ **Flexible command entry**. Commands can be entered by using a mouse, the function keys, or the pull-down menus. The debugger's command history can be used to re-enter commands.

❑ **On-screen editing**. Any data value displayed in any window can easily be changed by pointing (with the mouse) at the value, clicking, and entering the correct value.

❑ **Continuous update**. The debugger continuously updates information on the screen, highlighting changed values.

❑ **Comprehensive data display**. You can easily create windows for displaying and editing the values of variables, arrays, structures, pointers — any kind of data — in their natural format (float, int, char, enum, or pointer). Entire linked lists can be displayed (see Figure 5–9).

❑ **Patch assembler**. You can modify code from the debugger command line without reassembling your assembly source.

Figure 5–9. Debugger's Data Display

```
┌─DISP: str ──────────┐                    ┌─WATCH ──────────┐
│ a    123           ▲│                    │ 1: str.a 0      │
│ b    0             ││                    │ 2: F0 1.000000e │
│ c    75435┌─DISP: *str.f3 ──┐            │ 3: color GREEN  │
│ f1   3    │ a    8327      ▲│            └─────────────────┘
│ f2   6    │ b    666       ││
│ f3   0x00f│ c    87213┌─DISP: *str.f3─>f3 ─┐
│ f4   [...]│ f1   45   │ a    75           ▲│
└───────────│ f2   27   │ b    3212         ││
            │ f3   0x00f│ c    782          ││
            │ f4   [...]│ f1   7            ▼│
            └───────────│ f2   9            ││
                        │ f3   0x00f000a    ││
                        │ f4   [...]        ││
                        └───────────────────┘
```
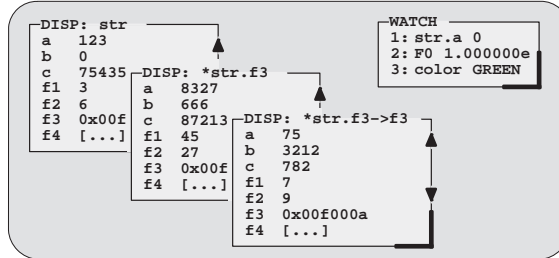
❏ **Powerful command set**. The TMS320 debugger supports a small but powerful command set that makes full use of C expressions. One debugger command performs actions that might require several commands in another system.

❏ **Compatibility**. The TMS320C31 C source debugger runs on IBM PC/ATs and compatible PCs. For the simulator, the debugger is available on Sun workstations.

❏ **Profiler.** The C source debugger has an option for profiling software. When you are deciding whether to convert portions of a program from C to assembly, it is helpful to know which functions take the most time. A profiler that measures the amount of execution time in different functions or portions of a program is very helpful. The profiler is easy to use and provides a number of features, including

■ **Elegant user interface**. The TI code profiler shares the same fully configurable, window-oriented, and mouse-driven interface as the TI C source debugger, so learning to profile is quick and easy.

■ **Multilevel profiling**. An assembly window and a C window are displayed, so you can profile C code, assembly code, or both simultaneously.

■ **Powerful command set**. A rich set of commands is available to select and manipulate profile areas on the global, module, function, and explicit levels, so you can efficiently profile even the most complex applications.

■ **Comprehensive statistics**. The profiler provides all the information you need to identify bottlenecks in your code:

- The number of times each area was entered during the profile session.
- The total execution time of an area, including or excluding the execution time of any subroutines called from within that area.
- The maximum time for one iteration of an area, including or excluding the execution time of any subroutines called from within that area.

■ **Versatile display**. The ability to choose profile areas, the type of statistical data, and sorting criteria ensures an efficient, customized display of the statistics. The data can also be accompanied by histograms to show the statistical relationship between profile areas.

■ **Disabled areas**. You can disable portions of a profile area to prevent them from adding to the statistics. This is convenient for removing the timing impact of standard library functions or a fully optimized portion of code.

■ **Simplicity**. The profiler's simple setup, default configurations, "canned" commands, and inherent flexibility facilitate sophisticated profiling within a short time.

5.3 TMS320C31 Assembly Language Tools

The TMS320C31 assembly language tools are code-generation tools that convert assembly language source files into executable object code. Key features include:

❑ Macro capabilities and library functions

❑ Conditional assembly

❑ Relocatable modules

❑ Complete error diagnostics

❑ Symbol table and cross-references

The **assembler** translates assembly language source files into machine language object files. Source files can contain instructions, assembler directives, and macro directives. Assembler directives control various aspects of the assembly process such as the source-listing format, symbol definition, and the way the source code is placed into sections. The assembler has the following features:

❑ Processes the source statements in a text file to produce a relocatable object file

❑ Produces a source listing (if requested) and provides control over this listing

❑ Appends a cross-reference listing to the source listing (if requested)

❑ Allows segmentation of user's code

❑ Maintains an SPC (section program counter) for each section of object code

❑ Defines and references global symbols

❑ Assembles conditional blocks

❑ Supports macros, allowing the user to define macros either in-line with or within a macro library

The **linker** combines object files into a single executable object module. As it creates the executable module, it performs relocation operations and resolves external references. The linker accepts COFF (common object file format) object files (created by the assembler) as its input. It can also accept archive library members and modules created by a previous linker run. Linker directives allow you to combine object file sections, bind sections and symbols to specific addresses, and define/redefine global symbols. The linker has these features:

❑ Defines a memory model that conforms to the target system's memory

❑ Combines object file sections

❑ Allocates sections into specific areas within the target system's memory

❑ Defines or redefines global symbols to specific values

❑ Relocates sections to final addresses

❑ Resolves undefined external references between the input files

❑ Allows separate load-time and runtime addresses for sections of code

The **archiver** makes it possible to collect a group of files into a single archive file. For example, several macros can be collected together into a macro library. The assembler will search through the library and use the members that are called as macros by the source file. Also, it is possible to use the archiver to collect a group of object files into an object library. The linker will include the members in the library that resolve external references during the link.

Most EPROM programmers do not accept COFF object files as their input. The ROM30 object format converter must be utilized to convert the COFF object file into Intel, Tektronix, or TI-tagged hex object format. ROM30 is part of the assembler, linker, and archiver package. The converted file can then be downloaded into the EPROM programmer.

5.4 TMS320C3x Software Simulator

A simulator is a software program that simulates the TMS320C3x micropro-
cessor and microcomputer modes for cost-effective software development
and program verification in non-realtime. With the inexpensive software simu-
lator, you can debug without target hardware. Files can be associated with I/O
ports so that specific I/O values can be used during test and debug. Time-criti-
cal code, as well as individual portions of the program, can be tested. The
clock's counter allows loop timing during code optimization. Breakpoints can
be established according to read/write executions (using either program or
data memory) or instruction acquisitions. The simulator uses the standard
C/assembly source debugger interface (described in Section 5.1), allowing the
user to debug code in C, assembly, or both.

Key features of the TMS320C3x software simulator include:

❑ Execution of user-oriented DSP programs on a host computer

❑ Inspection and modification of registers

❑ Data and program memory modification and display:

■ Modification of an entire block at any time

■ Initialization of memory before a program is loaded

❑ Simulation of peripherals, caches, and pipelined timings

❑ Extraction of instruction cycle timing for "device performance" analysis

❑ Programmable breakpoints on:

■ Instruction acquisition

■ Memory reads and writes (data or program)

■ Data patterns on the data bus or the program bus

■ Error conditions

❑ Trace on:

■ Accumulator

■ Program counter

■ Auxiliary registers

❑ Single-stepping of instructions

❑ Interrupt generation at user-specified intervals

❑ Error messages for:

■ Illegal opcodes

■ Invalid data entries

❑ Execution of commands from a journal file

❑ A branch to "self" is detected

❑ Execution is halted

Once program execution is suspended, the internal registers and both program and data memories can be inspected and/or modified. The trace memory can also be displayed. A record of the simulation session can be maintained in a journal file so that it can be re-executed to regain the same machine state during another simulation session.

❑ Simulation of the TMS320C31's entire instruction set

❑ Simulation of the TMS320C31 peripheral's key features

❑ Command entry from either menu-driven keystrokes (menu mode) or line mode

❑ Help menus for all screen-displayed modes

❑ Interface that can be user-customized

❑ Simulation parameters quickly stored/retrieved from files to facilitate preparation for individual sessions

❑ Reverse assembly for editing and reassembling source statements

❑ Memory that can be displayed (at the same time) as

■ Hexadecimal 32-bit values

■ Assembled source

❑ Execution modes

■ Single/multiple instruction count

■ Single/multiple cycle count

■ Until condition is met

■ While condition exists

■ For set loop count

■ Unrestricted run with halt by keyed input

❑ Trace execution with display choices

■ Designated expression values

■ Cache memory

■ Instruction pipeline

❑ Simulation of cache utilization

❑ Cycle counting

■ Display of the number of clock cycles in a single-step operation or in the run mode

■ Externally generated mode that can be configured with wait states for accurate cycle counting

The simulator lets you verify and monitor the state of the processor. Simulation speed can be either thousands of instructions per second (VAX VMS and SUN-3 UNIX) or hundreds of instructions per second (PC-DOS/MS-DOS).

The TMS320C31 simulator is available for the IBM PC-DOS/MS-DOS (5.25-inch floppy), the VAX/VMS (in backup format on 1600-bpi magnetic tape), and the SUN-3/4 UNIX (in TAR format on 1600-bpi magnetic tape) oper-ating systems. The PC configuration requires a minimum of 512K bytes for the TMS320C31 simulator.

5.5   TMS320C3x Evaluation Module

The TMS320C3x evaluation module (EVM) is a low-cost development board used for device evaluation, benchmarking, and limited system debug. The TMS320C3x EVM (see Figure 5–10) eliminates the cost barrier to evaluating and developing embedded systems based on the TMS320C31.

Features include:

❑ Assembler

❑ On-board memory

❑ Host upload/download capabilities

❑ I/O capability

Figure 5–10. TMS320C3x EVM

Insert Photo D

Get this photo from the TMS320 Family Development Support Reference Guide (job # 61136); page 5–16; Figure 5–4.
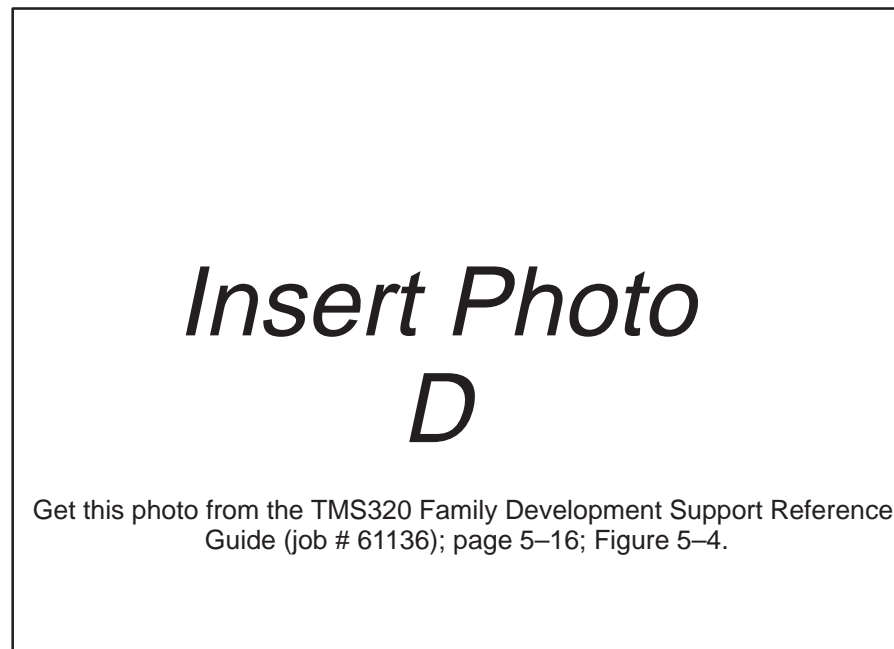
The TMS320C3x EVM enables you to benchmark and evaluate code in real-time while the device is operating at 30 MHz in the rich development environment of the TMS320C3x assembler/linker and C/assembly source debugger interface. Applications can be benchmarked and tested easily with the analog-ready interface.

The TMS320C3x EVM comes complete with a PC half-card and software package. The EVM board contains:

❏ One TMS320C30 — a 33-MFLOP, 32-bit processor. TMS320C31 applications can be developed by using only those 'C30 features available on a 'C31

❏ 16K-word, zero wait-state SRAM, allowing coding of most algorithms directly on the board

❏ Analog interface for embedded systems development

❏ An external serial-port interface that can be used for connecting multiple EVMs or for extra analog interfacing

❏ A host port for PC communications

❏ Embedded emulation support via the 74ACT8990 test bus controller

The system also comes with all of the software required to begin application development on a PC host:

❏ The window-oriented, mouse-driven interface supports downloading, executing, and debugging of assembly code or C code, including modification/display of memory and registers, software single-step, and breakpoint capabilities.

❏ The TMS320C3x assembler/linker is also included with the EVM. For high-level language programming, the optimizing ANSI C and the Ada compilers are offered separately.

The TMS320C3x EVM is supported on PC-AT/MS-DOS (version 3.00 or higher) platforms.

5.6 TMS320C3x Emulator

The TMS320 Extended Development Systems (XDSs) are powerful, full-speed emulators used for system-level integration and debug. TI developed the world's first in-system scan-based emulator (XDS) for TMS320C3x processors.

Scan-based emulation is a unique, nonintrusive approach to system emulation, integration, and debug. This approach was conceived and developed by TI to address hardware/software characteristics (reduced internal bus visibility, highly pipelined architectures, faster cycle times, higher-density packaging) that are inherent to sophisticated VLSI systems.

Scan-based emulation eliminates special "bond-out" emulation devices, target cable/buffer signal degradation, and the mechanical and reliability problems associated with target connectors and surface-mount packaging. With scan-based emulation, your program can execute in realtime from internal or external target memory — no extra wait states are introduced by the emulator at any clock speed.

The TMS320C31's architecture implements scan-based emulation through internal, shift-register, scan chains accessed by a single serial interface. The scan chains provide access to internal device registers and state machines, allowing complete visibility and control. This nonintrusive approach even operates in a production environment where the DSP is soldered into a target system.

Since program execution takes place on the TMS320C31 in the target system, there are no timing differences during emulation. This new design offers significant advantages over traditional emulators. These advantages include:

❑ No cable length transmission line problems

❑ Nonintrusive system

❑ No loading problems on signals

❑ No artificial memory limitations

❑ TMS320C3x C/assembly source debugger interface

❑ Easy installation

❑ In-system emulation

❑ No variance from device's data sheet specifications

The TMS320C3x XDS emulator (see Figure 5–11) is a user-friendly, PC-based development system that supports hardware development on the TMS320C30 and TMS320C31. This emulator provides a means for developing the software and hardware within a target system. Access is provided to every memory location and register of the TMS320C3x through the use of a revolutionary scan path interface. The TMS320C3x XDS emulator board interprets commands and converts these commands into the appropriate signal sequences necessary to control the TMS320C3x in your target system. Key features of the TMS320C3x XDS emulator include:

❏ Full-speed execution and monitoring of the TMS320C3x in your target system via a 12-pin target connector

❏ TMS320 C/assembly source debugging (PC/MS-DOS) via TI's standard windowed Programmer's Interface (see Section 5.2)

❏ 200 software breakpoints

❏ Software trace/timing

❏ Single-step execution

❏ Loading/inspecting/modification of all registers

❏ Uploading/downloading of program memory and data memory

❏ Benchmarking of execution time of clock cycles

Figure 5–11. TMS320C3x XDS Emulator

Insert Negative F

Get this photo from the TMS320 Family Development Support Reference Guide (job # 61136); page 5–28; Figure 5–10.
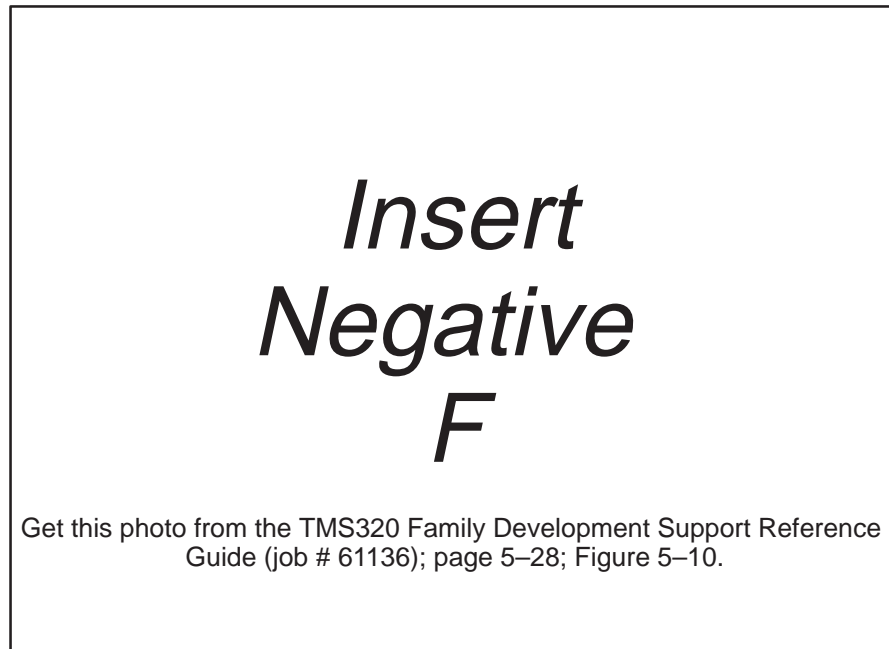
Software breakpoints allow program execution to be halted at a specified instruction address. When a given breakpoint is reached, the program halts execution. At this point, the status of the registers and of the CPU is available. Their contents are visible in the appropriate windows; to view the contents of other memory locations, only one command is required.

Software trace lets you view the state of the TMS320C3x when a breakpoint is reached. This information can be saved in a file for future analysis. Software timing allows you to track the clock cycles between breakpoints for benchmarking of time-critical code.

Single-step execution gives you the capability to step through the program, one instruction at a time. After each instruction, the status of the registers and CPU are displayed. This provides greater flexibility during software debug and helps reduce the development time.

Object code can be downloaded to any valid TMS320C3x memory location (program or data) via the scan path interface. Downloading a 1K-byte object program typically takes 100 ms. In addition, by inspecting and modifying the registers while single-stepping through a program, you can examine and modify program code or parameters.

The emulator's configurability gives your system flexibility. You can configure both memory and screen color. The address range, memory type, and access

type assigned to each location can also be configured. The memory map, which may include EPROM, SRAM, DRAM, and on-chip memory and peripherals, can be configured to reflect the actual peripheral environment of the target system, including wait states and access privileges.

TMS320C3x XDS System Requirements

Host	IBM PC-AT
Slot	One and one-half 16-bit slots
Memory	Minimum of 640K words
Storage	One floppy drive and one hard drive
Operating System	PC/MS-DOS 2.0 or later version
Power Supply	Minimum; approximately 3 amps @ 5 volts (150 watts)

5.7 TMS320C3x Application Board With Software Demo

Key features of the TMS320C3x application board are:

❑ 16K×32-bit, zero wait-state, full-speed SRAM on the primary bus

❑ Two selectable banks of 8K×32-bit, zero wait-state, full-speed SRAM on the expansion bus

❑ TMS320C30 DSP

❑ 512K×32-bit DRAM (user-upgradable to 1M×32-bits)

The large amount of on-board SRAM affords realtime emulation and memory storage flexibility for a variety of algorithms. The on-board SRAM provides zero wait-state access to memory allowing read/write in realtime.

Three types of DRAM cycles are used on the TMS320C3x application board: Single-word read, single-word write, and page-mode read. These operations require four, two, and one wait state per access, respectively. Note that when you invoke page mode read while accessing the emulator's DRAM, fewer wait states are required. Page-mode DRAM is often used to improve "bulk storage" performance. Page-mode read cycles are automatically invoked when the TMS320C3x performs two or more back-to-back read cycles on the same memory page; one page of memory holds 256 words — the default memory bank size for the TMS320C3x.

SPOX Operating System software is also available for the application board.

5.8 HP 64776 Analysis Subsystem

TI and Hewlett-Packard jointly designed and developed the HP 64776 Analysis Subsystem, an emulator/analyzer for the TMS320C3x (see Figure 5–12). (For TMS320C31 analysis, an adapter is available from HP to use the subsystem with a surface-mounted TMS320C31.) The HP 64776 combines with the TI TMS320C3x XDS emulator to yield a complete tool set for integrating hardware with software, producing an extremely powerful debug environment. HP's active probe technology yields the maximum electrical and mechanical transparencies, improved signal quality, and realtime control and debug of the target system at full operating speed.

The complete analysis subsystem integrates the HP 64776, the TMS320C3x XDS, and the C source debugger (described in Section 5.2) in a stand-alone PC environment. The TI debugger acts as the user interface, and communications between the subsystem and the PC are handled through an RS-232C connector. This powerful system provides software and hardware breakpoint and trace, as well as sophisticated bus-cycle analysis.

Figure 5–12. HP 64776 Analysis Subsystem

Insert Negative H

Get this photo from the TMS320 Family Development Support Reference Guide (job # 61136); page 5–34; Figure 5–12.
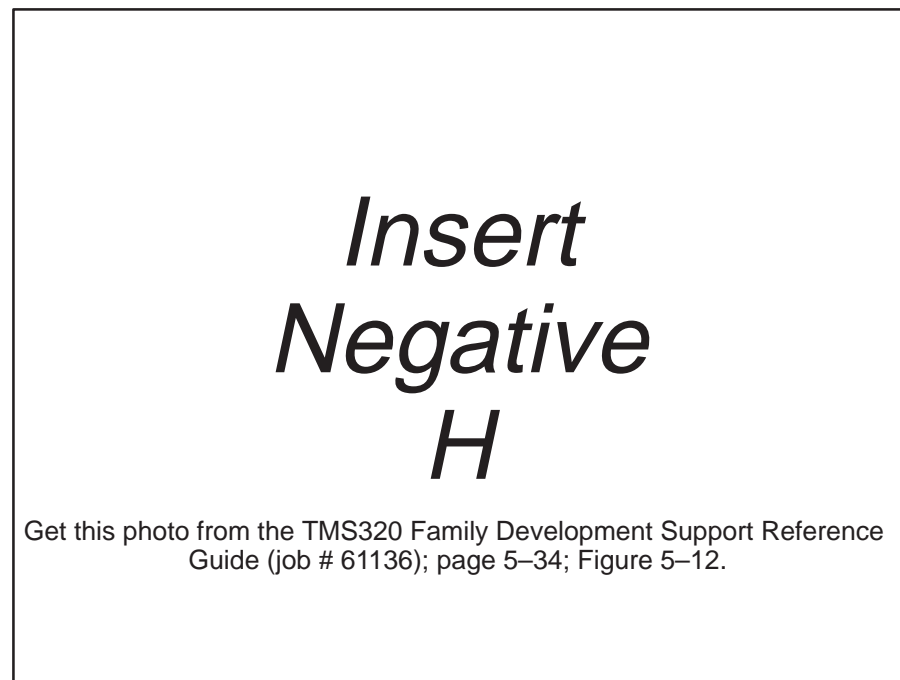
Key features of the subsystem include:

❑ 64 analysis channels that can trace the TMS320C3x's primary or expansion bus as well as status information. Nonintrusive analysis lets you view the processor's bus cycles in realtime. Analysis can be performed on the following signals:

- A0 – A23
  (primary-bus address)
- D0 – D31
  (primary-bus data)
- $\overline{STRB}$
- R/$\overline{W}$
- $\overline{HOLDA}$

- XA0 – XA12
  (expansion-bus address)
- XD0 – XD32
  (expansion-bus data)
- $\overline{MSTRB}$
- $\overline{IOSTRB}$
- $\overline{IACK}$

- $\overline{INT0}$ – $\overline{INT3}$
- TCLK0
- TCLK1
- XF0
- XF1

❑ Trace specifications that can be set up easily, using address, data, and status-event comparators. A range comparator can also be used to qualify addresses or data.

❑ Hardware breakpoint capabilities that enable you to detect a specified event and stop the processor. Once the processor is stopped, the debug capabilities of the TMS320C3x XDS facilitate isolation of target's hardware/software problems.

❑ The ability to drive triggered signals to and receive them from other instruments such as logic analyzers and oscilloscopes, allowing synchronized measurements between tools.

The HP 64776 operates on PC/AT platforms utilizing DOS (version 3.0 or higher).

5.9   TMS320 Technical Support

5.9.1   Technical Documentation

A wide variety of technical literature is available to assist you through the design cycle. These documents include product and preview bulletins, data sheets, user's and reference guides, over 2000 pages of application notes, and textbooks offered by Prentice-Hall, John Wiley and Sons, and Computer Science Press. To inquire about available TMS320 literature, call the Customer Response Center (CRC):

(214) 995–6611

The following list describes the general contents of each major category of technical documentation available through the Customer Response Center:

❏ Product and preview bulletins and product briefs give an overview of the devices and development support within the TMS320 family, presenting capabilities, diagrams, and hardware/software applications.

❏ User's guides for TMS320 processors provide detailed information regarding the architecture of the device, its operation, assembly language instructions, and hardware and software applications.

❏ Data sheets include electrical specifications, timing characteristics, and mechanical data for a device.

❏ Application books/reports describe theory and implementation of selected TMS320 applications, including algorithms, code, and block/schematic/ logic diagrams. Currently, there are over 2000 pages of application reports to support the TMS320 family.

❏ Technology brochures provide an overview of various implementations of DSP technology.

5.9.2 *Details on Signal Processing* Newsletter

The TMS320 newsletter, *Details on Signal Processing*, is published quarterly to update TMS320 customers on product information and industry trends. It covers TMS320 products, documentation, third-party support, application boards, mini-application reports, development tool updates, contacts for support, design workshops, seminars, conferences, and the TMS320 university program.

To be added to the mailing list, call the Customer Response Center:

(214) 995–6611

5.9.3 TMS320 Bulletin Board Service

The TMS320 Bulletin Board Service (BBS) is a telephone-line computer bulletin board that provides access to information about the TMS320 family. The BBS is an excellent means of communicating specification updates for current or new TMS320 application reports as they become available. It also serves as a means to trade programs with other TMS320 users.

The BBS contains TMS320 source code from the more than 2000 pages of application reports written to date. These programs include macro definitions, FFT algorithms, filter programs, ADPCM algorithms, echo cancellation, graphics, control, companding routines, and sine-wave generators.

You can access BBS with a terminal or PC and a modem. The modem must be able to communicate at a data rate of either 300, 1200, 2400, or 9600 bps. A character length of eight bits is required, with one stop bit and no parity. The telephone number of the bulletin board is (713) 274–2323. There is a 90-minute access limit per day on the bulletin board. The BBS is open 24 hours a day. ROM-code algorithms may be submitted by secure electronic transfer via the TMS320 BBS.

5.9.4 TMS320 DSP Technical Hotline

The TMS320 group at Texas Instruments maintains a DSP Hotline to answer TMS320 technical questions. Specific questions regarding TMS320 device problems, development tools, third-party support, consultants, documentation, upgrades, and new products are answered.

The TMS320 DSP Technical Hotline is open five days a week from 8:00 AM to 6:00 PM Central Time. It is staffed with engineers ready to provide the support needed for your TMS320 design or evaluation.

To assure the maximum support from this service, first consult your product documentation. If your question is not answered there, gather all of the infor-

mation that applies to your problem. With your information, manuals, and products close at hand, call:

TMS320 DSP Technical Hotline   (713) 274–2320

For realtime transmission of information, a facsimile machine is available:

FAX   (713) 274–2324

or you may submit information via electronic mail:

The Hotline Internet address is

4389750@mcimail.com

The MCI mail address is

4389750 or **TMS320 Hotline**

Questions on pricing, delivery, and availability should be directed to the nearest TI Field Sales Office.

5.9.5   TMS320 Application Software

To simplify development of applications, TI and its third parties offer a wide variety of software that can be licensed. This software covers a range of DSP functionality that includes vocoders, speech recognition, modems, audio coders, and image coders. The software available for license can provide a headstart in the development of your final application. In addition, software applications that have been published in TI DSP user's guides and application books are available via the BBS.

Contact the DSP Hotline for a list of software available for the TMS320C31.

5.9.6   Design Workshops

Texas Instruments offers a wide array of up-to-date technical product seminars and design workshops through its Technical Training Organization (TTO) to assist designers in developing the skills needed to implement their ideas quickly, produce a quality product, and shorten time to market. Applications assistance is also offered through local Regional Technology Centers (RTCs).

The DSP design workshops give design engineers hands-on experience using the latest TMS320 products, development tools, and design techniques. These workshops go beyond the standard lecture format. The exercises and lab experiments start with the basics and move quickly into hands-on exercises. In these workshops, the student learns by doing, not just listening or ob-

serving. The workshops are designed to help customers shorten the design cycle, control development costs, and solve design challenges.

Further information on courses and schedules in North America can be obtained by contacting the TTO Central Registration office at (800) 336-5236, ext. 3904.

5.9.6.1 *TMS320C3x Design Workshop*

The TMS320C3x DSP design workshop introduces design engineers to the powerful TMS320C3x generation of DSPs. Hands-on, EVM-based exercises throughout the course give the designer a rapid start in utilizing TMS320C3x design skills. Experience with digital design techniques is desirable. Assembly language experience is required. C language programming experience is desirable.

Topics covered in the TMS320C3x DSP design workshop include:

❑ TMS320C3x architecture/instruction set

❑ Use of the PC-based TMS320C3x EVM

❑ Floating-point and parallel operations

❑ Use of the TMS320C3x assembler/linker

❑ C programming environment

❑ System architecture considerations

❑ Memory and I/O interfacing

❑ TMS320C3x development support

5.9.6.2 *Digital Control Design Workshop*

The digital control design workshop covers all the fundamental issues involved in the design and implementation of physical control systems using TMS320 DSPs. The workshop is divided into two major parts. The first part covers theory and design of control systems and discusses practical aspects that a control design engineer should be aware of before attempting to implement a controller. The second part is devoted to hands-on experience with TMS320C25 DSPs to demonstrate and practice control implementation examples. A design and implementation software package is used to test algorithms on an actual motor positioning system.

Topics covered in the digital control design workshop include:

❑ System modeling

❑ Stability analysis

❏ Analysis of numerical problems

❏ Quantization effects

❏ Truncation, rounding, and scaling issues

❏ Sampling rate selection

❏ Algorithm structural optimization

5.9.6.3   *Applications in C Design Workshop*

The Applications in C design workshop is an advanced, C programming course, which is tailored for practical, hands-on applications using Turbo C and the TI TMS320C3x C compiler. This course is for hardware and software engineers with a background in programming and an introductory knowledge of C. The course centers around data structure concepts illustrated with application examples. Program examples include file filters, sorting, Huffman coding for data compression, memory management, graphics algorithms, and other utilities.

Topics covered in the Applications in C design workshop include:

❏ Review of C language (syntax and conventions)

❏ Data structures, constructs, and concepts

❏ Optimization and efficiency techniques

❏ Arrays and pointers

❏ Portability issues

❏ Algorithms (FFT, discrete transforms, bit manipulation, etc.)

5.9.7   Design Services

The TI technical staff can offer applications assistance with customer designs through local Regional Technology Centers. Services include:

❏ Design assistance

❏ Simulation

❏ Emulation

Each Regional Technology Center uses up-to-date development systems, including workstations and personal computers, plus demonstration, test, and evaluation equipment. TI staff designers use fully equipped laboratories to provide efficient design assistance.

The first step to a successful design is an explanation of the project's parameter: production requirements, design function(s), and price. The results of these discussions will allow TI and a customer to explore:

❑ Design/cost trade-offs

❑ Product implementation options

Once the various trade-offs/options are selected and approved, Texas Instruments can provide further assistance in the design of a customer's product, sharing a mutual goal of bringing a successful product to market as quickly as possible.

5.9.8   RTC Locations

The following list gives the worldwide locations of the TI Regional Technology Centers.

Table 5–1.   RTC Worldwide Locations

North American Locations	
ATLANTA Texas Instruments 5515 Spalding Drive Norcross, GA 30092 (404) 662–7950	**NORTHERN CALIFORNIA** Texas Instruments 5353 Betsy Ross Drive Santa Clara, CA 95054 (708) 748–2220
BOSTON Texas Instruments 950 Winter Street, Suite 2800 Waltham, MA 02154–1263 (617) 895–9196	**SOUTHERN CALIFORNIA** Texas Instruments 1920 Main St., Suite 900 Irvine, CA 92714 (714) 660–8140
CHICAGO Texas Instruments 515 W. Algonquin Road Arlington Heights, IL 60005 (708) 640–2909	**OTTAWA** Texas Instruments Canada, Ltd 301 Moodie Drive, Suite 102 Nepean, Ontario Canada, K2H 9C4 (613) 726–1970
DALLAS Texas Instruments 7839 Churchill Way Park Central V, MS 3984 Dallas, TX 75251 (214) 917–3881	**MEXICO CITY** Texas Instruments de Mexico Alfonso Reyes 115 Col. Hipodromo Condesa Mexico, D.F., Mexico 06170 (52) (5) 515–6081 (52) (5) 515–6249
INDIANAPOLIS Texas Instruments 550 Congressional Blvd., Suite 100 Carmel, IN 46032 (317) 573–6400	
International Locations	
AUSTRALIA Texas Instruments Australia Ltd. 6–10 Talavera Road, North Ryde New South Wales, Australia 2113 Tel: (61) (2) 8789000	**JAPAN** (Tokyo) Texas Instruments Japan Ltd Ms Shibaura Building 9F 4–13–23 Shibaura Minato-Ku, Tokyo, JAPAN 108 Tel: (81) (3) 3769–8700

Table 5–1. RTC Worldwide Locations (Concluded)

International Locations	
BRAZIL Texas Instruments Electronicos do Brasil Ltda Av. Eng. Luiz Carlos Berrini 1461–11o. andar 04571 Sao Paulo, SP, Brazil Tel: (55) (11) 535–5133	**JAPAN** (Osaka) Texas Instruments Asia LTD Osaka Branch Nissho-Iwai Bldg 5F 2–5–8 Imabashi Chuou-Ku Osaka, Japan 541 Tel: (81) (6) 204–1881
FEDERAL REPUBLIC OF GERMANY Texas Instruments Deutschland GMBH Kirchhorster Strasse 2 3000 Hannover 51, FR Germany Tel: (49) (511) 648021	**KOREA** Texas Instruments Korea Ltd. 28th Floor, Trade Tower 159 Samsung-Dong Kangnam-Ku, Seoul Trade Center P.O. Box 45 Seoul, Korea 135–729 Tel: (82) (2) 5512800
FEDERAL REPUBLIC OF GERMANY Texas Instruments Deutschland GMBH Haggertystrasse 1 8050 Freising, FR Germany Tel: (49) (8161) 80–0	**SINGAPORE** Texas Instruments Singapore (Pte) Ltd. Asia Pacific Division 101 Thomson Road #23–01 United Square Singapore 1130 Tel: (65) 2519818
FRANCE (Paris) Texas Instruments France 8–10 Avenue Morane Saulnier Borte Postale 67 Velizy Villcoublay Cedex, France Tel: (33) (13) 0701001	**SWEDEN** Texas Instruments International Trade Corporation Box 30 S–164 93 Kista Isafjordsgatan 7, Sweden Tel: (8) 752–5800
HONG KONG Texas Instruments Hong Kong Ltd. 8th Floor, World Shipping Centre 7 Canton Road Kowloon, Hong Kong Tel: (852) 7351223	**TAIWAN** Texas Instruments Taiwan Ltd. Taipei Branch 10 Floor, Bank Tower 205 Tung Hua N. Road Taipei, Taiwan 105 Republic of China Tel: (886) (2) 7139311
ITALY (Milan) Texas Instruments Italia S.P.A. Centro Direzionale Colleoni Palazzo Perseo Via Paracelso, North 12 20041 Agrate Brianza, MI, Italy Tel: (39) (39) 63221	**UNITED KINGDOM** Texas Instruments Ltd. Regional Technology Center Manton Lane Bedford, England MK41 7PA Tel: (44) (234) 270111

TMS320C31 Third-Party Support

This chapter lists third-party manufacturers and suppliers alphabetically by name and describes their current 'C31 products.

The third parties discussed in this chapter include:

6.1 Accelerated Technology, Inc.
P. O. Box 850245
Mobile, AL 36685
(800) 468–NUKE
(205) 661–5770

❑ **Nucleus RTX**

Nucleus RTX is a multitasking executive specifically designed for realtime embedded applications using the TMS320C3x microprocessors. Nucleus provides applications with advanced realtime facilities that encompass management of task execution, task communication and synchronization, system resources, predefined memory partitions, and dynamic-length memory.

Nucleus RTX facilities are designed to operate in a consistent, reliable, and efficient manner. Each task executing under Nucleus has a priority. When multiple tasks are ready to execute, the task with the highest priority is executed first. Tasks of the same priority execute in a first-in-first-out (FIFO) manner. In addition to the many standard realtime facilities, Nucleus also provides facilities such as task priority modification, task time slicing, item sizes for communication queues defined by the user, suspension of full queues, suspension on multiple empty queues, both types of memory management, suspension on unavailable memory, and event flag consumption. Additionally, any Nucleus task suspension can be given a maximum amount of time to stay suspended.

❑ **Software Products**

Accelerated Technology offers other realtime software products for use with the TMS320C3x generation. These include a multitasking debugger, a reentrant C library, an MS/DOS-compatible file system, and in the near future networking support in the form of TCP/IP protocols.

The Nucleus debugger provides access to all Nucleus structures in a user-readable fashion. Control structures for tasks, queues, semaphores, event flags, and memory management are all available for inspection. Additionally, the Nucleus debugger allows you to dynamically execute most of the Nucleus RTX service calls.

The reentrant C libraries supplied by Accelerated Technology provide standard ANSII interfaces for all functions, with the exception of file services (file services are provided by the Nucleus file system). Because the library routines are fully reentrant, application tasks running under Nucleus can use them.

Nucleus File is an MS/DOS-compatible file system that is capable of reading and writing standard floppy- and hard-disk formats. Nucleus File is specifically designed for embedded applications.

Accelerated Technology's realtime software products are primarily written in ANSII C and are optimized for performance on the TMS320C3x DSPs. All software products are delivered with complete source code and without any royalties.

❑ **Features of the Nucleus RTX Realtime Multitasking Executive**

- ■ Realtime, multitasking, executive for the TMS320C3x DSPs
- ■ Complete source code
- ■ No royalties
- ■ Priority base with optional preemption and time slicing
- ■ Task communication with user-defined public queues
- ■ Item size of each queue defined by user
- ■ Optional task suspension on full queues
- ■ Optional task suspension on multiple queues
- ■ Task synchronization with event flags
- ■ Optional consumption of event flags
- ■ Resource management with semaphores
- ■ Predictable fixed-length memory management
- ■ Flexible variable-length memory management
- ■ Optional task suspension when memory is unavailable
- ■ Optional timeout for any task suspension
- ■ System history log
- ■ Task performance analysis facilities
- ■ Task-oriented debugger
- ■ MS/DOS-compatible floppy file system
- ■ TCP/IP network support (Q4 92)

❑ **Technical Support**

- ■ Structured and documented source code
- ■ Detailed programmer's reference manual
- ■ Detailed internal design manual
- ■ Telephone consultation
- ■ Warranty and maintenance service
- ■ Extensive counseling and contract services

❑ **Shipping Media**—MS/DOS 5-1/4-inch diskette

Figure 6–1. Realtime Application Tasks

Schedule Management	Development Support
NU_Start() NU_Change_Priority() NU_Change_Time_Slice() NU_Control_Interruptor() NU_Enable_Preemption() NU_Disable_Preemption() NU_Relinquish() NU_Sleep() NU_Stop() NU_Reset() NU_Retrieve_Task3 () NU_Current_Task_ID()	NU_Reset_Performance_Timer() NU_Retrieve_Next_History_Entry() NU_Retrieve_Performance_Info() NU_Start_History_Saving() NU_Start_Performance_Timer() NU_Stop_History_Saving() NU_Stop_Performance_Timer()
Clock Management	**Fixed-Size Memory Management**
NU_Set_Time() NU_Read_Timer()	NU_Alloc_Partition() NU_Available_Partitions() NU_Dealloc_Partition()
Management	**Variable-Size Memory Management**
NU_Send_Item() NU_Force_Item_In_Front() NU_Retrieve_Item() NU_Retrieve_Item_Mult() NU_Retrieve_Queue_Status()	NU_Alloc_Memory() NU_Available_Memory() NU_Dealloc_Memory()
Management	**Resource Management**
NU_Set_Events() NU_Wait_For_Events()	NU_Request_Resource() NU_Retrieve_Resource_Status() NU_Release_Resource()

6.2 A.T. Barrett & Associates, Inc.
11501 Chimney Rock
Houston, Texas 77035
(800) 525–4302
(713) 728–9688
FAX: (713) 728–1049

❑ **RTXC, Realtime Kernel for single processor systems**
❑ **RTXC/MP, Realtime Kernel for multiple processor systems**

RTXC and RTXC/MP are fully preemptive, priority-driven, realtime kernels written in ANSI C that enable you to tap the full power of the TMS320C3x processors in realtime environments. Released in 1985, RTXC has been continuously upgraded.

Demonstration and benchmark disks on RTXC and RTXC/MP are available free of charge. An evaluation package containing a full kernel, a special user's manual, and special utilities to assist in evaluation of the kernel is also available. The package gives you the complete picture of the capabilities, performance, scalability and ease of use of these realtime kernels.

RTXC and RTXC/MP are available for a one-time site license fee. All configurations of processor and compiler bindings include full source code and require no runtime royalties. Most compilers are supported.

The combination of RTXC and RTXC/MP address a broad range of applications. RTXC is aimed at embedded applications, which would typically use a single TMS320C2x, 'C3x, or 'C5x DSP. RTXC/MP is targeted at applications employing multiple TMS320C3x or 'C4x processors.

RTXC and RTXC/MP share many of the same attributes and components. Most importantly, both kernels use a similar application program interface (API). However, RTXC/MP extends the RTXC API to include those functions which are necessary for the special requirements of the multiprocessing environment. The API provides a wide range of kernel services such as task management, timer management (including timeouts), intertask communication and synchronization, memory and resource management, and processor-specific ones. Intertask communication can occur via semaphores, messages, and FIFO queues. Because of the commonality of the API, software developed for the RTXC single processor system is highly portable to the multiprocessing world of RTXC/MP.

A set of high-end utilities help you configure, compile and fine-tune the application. Both kernels use a system generation utility, RTXCgen, which permits interactive definition of the system components, tasks, queues, semaphores, memory partitions, and mailboxes. RTXCgen maintains the

user-defined list of all application or topology-dependent attributes. For example, resizing of a memory partition requires only the regeneration of the C source file for memory partitions and no changes in the application source code. RTXCgen automatically monitors changes made to the system component definitions. When directed to generate C source code for system tables, RTXCgen also produces header files only for those system components that have been changed. Thus, RTXCgen promotes concordance between the source code, representing the specified components of the application, and the header files used for referencing members of that application. In addition, RTXCgen provides listings of all system components that serve as a primary source for system-level documentation.

A system-level debug utility, RTXCbug, is also common to both kernels. RTXCbug examines the current state of the tasks, queues, and semaphores and presents a coherent picture, or snapshot, of the interaction between the system and the application tasks. It even permits manual task management.

RTXC/MP includes two special utilities not found in the single processor RTXC kernel. RTXC monitors the system and provides, on demand, a list of the last 256 scheduled events, permitting you to trace the immediate history of the application. The second utility, a built-in work load monitor, acts to measure and to redistribute the workload at runtime.

RTXC and RTXC/MP address two important problems. First, the use of ANSI standard C protects you from technology changes, thus preserving the software development investment. The easy upgrade path from a single processor version of RTXC to the multiple processor version of RTXC/MP ensures that the software investment is future proof. Second, the difficulties of parallel or distributed programming become less problematic through RTXC/MP's use of a virtual single-processor model. The implementation is geared towards maximum performance so that hard realtime constraints are still satisfied even in a multiple-processor system architecture.

❑ **RTXC Specifics**

With an implementation history dating from 1978, RTXC provides a sound foundation for the solution of complex realtime systems. It is based on the concept of preemptive multitasking that permits a system to make efficient use of both time and system resources. RTXC is distributed in three source code configurations defined by the set of kernel services embodied in each. The different configurations are available to meet the real needs of the embedded systems marketplace where there is a wide diversity of functional capabilities required in a realtime kernel. RTXC allows you to license the source code library that most closely fits your needs. If you need more capabilities later on, there is a simple upgrade path.

The three source code libraries, basic, advanced, and extended are compatible with each other. All of the services in the Basic Library are included in the advanced library. All of the advanced library is part of the extended library. If you obtain a license to the basic library, you can upgrade to either the advanced or extended library without changing the application programs developed with the basic library.

❑ **RTXC/MP Specifics**

The range of applications is vast, from single-processor-embedded systems to complex control systems with various degrees of fault-tolerance and using tens of processors. Throughout the spectrum of applications, RTXC/MP provides transparent distributed realtime processing without the need to change any line of application source code when changing attributes of system resources (for example, the location of tasks, queues, semaphores, memory blocks, and priority of tasks).

The transparency simply means that any cluster of processors can be regarded as a single realtime-processing engine. While processors give you scalable computing power, RTXC/MP gives you scalable realtime software. Transparency is achieved by the implementation of a virtual single-processor model. The model uses a global naming scheme in which all system resources are known system wide. The use of the global naming scheme relies on the embedded router in RTXC/MP. The RTXC/MP router which supports up to 64K processor nodes and 64K tasks is attractive for pure communication applications. The routing tables, automatically generated by RTXCgen from the link connections table, allow you to write all communication between tasks as if they were located on the same processor. Under high communication loads, prioritized handling in the router avoids lower priority messages blocking higher priority messages.

While the single processor kernel (RTXC) can be used with multiple processors if you define your own communication protocols, the distributed

version frees you from this burden. Moreover, because RTXC/MP uses a message-based mechanism, ports to common-memory, local-memory, and LAN-based systems can easily be done.

A distributed I/O library and graphics server is also available for RTXC/MP.

The design philosophy behind RTXC/MP has proven to be a major step forward to shield software applications from technology changes. It offers a future-proof environment for the transparent development of scalable realtime software on scalable processor hardware.

6.3 Biomation
19050 Pruneridge Ave.
Cupertino, CA 95014
(800) 944–2466
FAX: (408) 988–1647

❑ **CLAS 2000 and CLAS 4000 Logic Analyzers**

The CLAS 2000 and CLAS 4000 Logic Analyzers provide measurement capability for examining high-speed CISC, RISC, ASIC, and general-logic design including:

- 96-channel module with 50/100/200 MHz capture
- Measurement widths of up to 384 channels
- Configurations with 1 to 4 logic analyzers per CLAS 4000
- Configurations with 1 or 2 logic analyzers per CLAS 2000
- Full-speed triggering with multilevel trace control
- Time-stamped transitional recording
- Disassembling of all DSP instructions
- Full-speed operation for clock and data rates
- Monitoring of every 'C3x signal with a single-probe connection
- Small interface probes for dense boards
- Reliable high-speed probing
- Timing and state measurements made through the processor probe
- Full symbolic display and triggering for address, data, and control groups
- Support for multiprocessor systems

❑ **Operation**

Operation is quick and simple. To connect to your target, just install the probe board between the 'C3x CPU and its socket. Click on the icon representing the 'C3x disassembler setup and the entire logic analyzer will be configured automatically. The setup assigns channels to all of the CPU's signals, arranges the channels into address, data, and status groups, and sets up the clocking for the 'C3x. Predefined trigger patterns are also provided so that you can quickly specify which samples are captured.

❑ **Display**

Data captured on the CLAS can be viewed simultaneously in several windows with each window displaying the data in different formats. Results are displayed as symbolic, hex, octal, and binary radices in a state window; as waveforms in a timing window; and as decoded mnemonics in a disassembly window. Display radices can be added or changed at any time without taking a new measurement.

Decoded instructions for the TMS320C3x processor are displayed in the disassembly window. The MAP hardware is capable of capturing all bus cycles. The 'C3x must be executing out of external RAM in order for the disassembler to operate effectively. Four disassembly display modes are available: display all bus cycles, delete non-executed cycles, delete data read/writes, and display executed code only. These modes allow the display to be tailored to your needs. Hardware engineers will appreciate "DIsplay All Bus Cycles," while the "Display Executed Code Only" will look much like the program listing to which a software engineer is accustomed (with symbolic labels for addresses).

❏ **Passive Interface**

Biomation uses passive interfaces in microprocessor probe adapters. Passive interfaces bring the processor signals directly to the logic analyzer's high-impedance data probes. Direct connection to the CPU allows timing measurements to be made directly through the probe. Where loading is critical, clock signals have an active buffer on the probe board to ensure proper operation of the system under test.

❏ **Specifications**

Signals Monitored: Two 96-channel pyramid measurement modules per CPU support full TMS320C3x disassembly. Additional pyramid modules can be added to monitor other system signals.

Input Impedance: The input impedance of all signals are 1 MΩ shunted by 8 pF except $\overline{STRB}$, $\overline{RDY}$, $\overline{MSTRB}$, $\overline{IOSTRB}$, $\overline{XRDY}$, and H1. Input impedance on these signals are approximately 500 kΩ shunted by 16 pF.

❏ **Sampling**

External clock: DC to 50 MHz

Internal clock: 100 ms to 5 ns

Setup time: 7.0 ns-typical (reduced to 4 ns with timebase sync probe)

Hold time: 0 ns

❏ **Power**

All MAP power is provided by the CLAS chassis. No power is required from the target system.

❏ **Mechanical**

Connection to the target is made using a 190-pin PGA package (15×15 grid) mounted on the MAP probe adapter. The probe adapter is placed between the CPU and its socket. A zero-insertion-force (ZIF) socket is included, but can be removed when space is limited.

❑ **Probing Considerations**

The MAP probe adapter is made as small as possible to allow an easy con-
nection when other chips are mounted next to the CPU. The probe adapter
extends a maximum of 1.5 cm (0.6 in) from the chip on the sides and 8.6
cm (3.4 in) along the back.

❑ **Miscellaneous**

Size:	Interface Box	4.0 cm (1.6 in) high, 21.3 cm (8.4 in) wide, 22.9 cm (9.0 in) deep
	Probe Adapter	2.1 cm (0.8 in) high (with ZIF) 6.5 cm (2.5 in) wide 13.7 cm (5.4 in) long
	Cable	34 cm (13.5 in) long
Weight:		0.8 kg (1.75 lb) with cables and probe adapter
Temperature:		0–50°C, noncondensing

6.4 Byte-BOS
P.O. Box 3067
Del Mar, CA 92014
(800) 788–7288 or
(619) 755–8836

❑ Byte-BOS Multitasking Operating System

Byte-BOS Multitasking Operating System (BOS) is a low-cost, full-featured, realtime preemptive multitasking operating system and is available for TMS320 DSPs. Byte-BOS brings the cost of multitasking within reach of all embedded software applications by providing a common code base across a wide range of processors, including the TMS320C3x DSPs. BOS consists of a C library of realtime multitasking functions with the following features:

■ Preemptive and nonpreemptive prioritized task scheduling
■ Task control and management
■ Timer management
■ Event synchronization
■ Message passing
■ Resource management
■ Serial I/O management
■ Interrupt stack and nested interrupt handling
■ Low power management
■ Function timeout, blocking, and nonblocking return
■ TMS320 on-chip timer and serial port integration
■ Application code for TMS320 embedded platform
■ External UART serial I/O management (add-on library)
■ Fixed block memory management (add-on library)
■ Multiple programmable event timers (add-on library)
■ Multiple message buffers (add-on library)
■ BOSVIEW realtime operating system view port (add-on library)
■ Library and applications code-compiler batch and make files
■ Comprehensive reference manual with many examples
■ Prototype and test TMS320 BOS applications on a PC
■ Source code site license (unlimited product usage)
■ No royalty executable code distribution
■ One year of technical support and revision updates

BOS is optimized for all TMS320 DSPs and has excellent performance. BOS is configured to work with the Texas Instruments C development systems and includes a working application.

6.5 Computer Motion, Inc.
270 Storke Rd., Suite 11
Goleta, CA 93117
(805) 685–3729
FAX: (805) 685–9277

❏ C++ Compiler

Computer Motion Inc. has introduced object-oriented programming using C++ for the TI TMS320C30 and TMS320C31 DSPs. This compiler is based on the GNU C++ retargetable compiler and executes on SPARCstation platforms. This compiler translates programs directly to TMS320 assembly language. The TI assembler and linker can then be used to create the final executable code. The object code generated from the assembly language output can be linked with other programs compiled with both the TI C compiler and the runtime-support libraries. The package includes documentaiton manuals and a quarter-inch cartridge tape that contains both a C++ and a C compiler.

6.6   Electronic Tools GmbH
Zum Blauen See7
4030 Ratingen
Germany
0049–2102–88010
FAX: 0049–2102–880123

❑ miniKit-320C31 Embedded DSP System

miniKit-320C31 is a complete embedded DSP system based on the Texas Instrument's TMS320C31 and is not larger than the size of a credit card. The module addresses two significant areas of DSP-based system design: it can either be used as a fully functional development system on which algorithms can be rapidly implemented and debugged or as a module which is easily integrated into any user's end-system. The module is particularly attractive for low to medium volume embedded solutions requiring a fast turnaround time as it may be designed into any industrial product just like a large IC. This proven platform manufactured in SMD-technology offers a number of standardized intefaces which allow full access to all of the DSP's features. Compatibility is guaranteed with other products of Electronic Tool's miniKit range. Debugging is performed on a PC with the TI db30 source-level debugger which is linked to miniKit-320C31 via a small PC controller board and the emulation port of the TMS320C31. A rich set of software utilities ensure that all steps from algorithm implementation in C- or assembler code right down to programming miniKit's boot EPROM can be achieved on the fly.

■ Credit card sized DSP system: 85mm × 61mm
■ TMS320C31 (33 MHz)
■ 128K × 32 zero wait-state static RAM
■ 64K × 8 boot EPROM; booting possible via EPROM, host
■ Interface, RAM or serial interface
■ Watchdog timer
■ Power failure detection
■ Battery backup
■ miniBus interface: standardized 16-bit parallel bus for attaching peripherals
■ HostBus interface: standardized 8-, 16-, 32-bit parallel interface for attaching microcontrollers; also available for bit I/O
■ ExpansionBus interface: TMS320C31-specific bus (32-bit parallel) for expanding memory and attaching peripherals
■ Serial interface
■ Timer interface
■ Emulation interface

6.7   Integrated Motion, Incorporated
758 Gilman Street
Berkeley, California 94710
(510) 527–5810
FAX: (510) 527–7843

❑ MX31 Modular Embedded System

The MX31 is a low-cost, modular, small-footprint general-purpose embedded controller with expansion daughter boards designed for applications involving motion control. The system is based on a motherboard/daughter board architecture for flexibility and low cost. The motherboard is a processor unit consisting of a 33-MHz TMS320C31 floating-point DSP, ROM, RAM, and other support devices. Each daughter board provides the peripherals required to control a two-axis servo-actuated mechanical system. Up to four daughter boards can be stacked in a single system to control up to eight servo axes.

■ Motherboard features

- 33-MHz TMS320C31 floating-point DSP
- 16- to 256K-word ROM
- Up to 256K-word zero-wait-state RAM
- RS232 serial port
- 16-bit parallel I/O

■ Daughter board features

- 2-channel, 16-bit shaft encoder interface
- 2-channel, 16-bit analog output
- 12-bit digital input, 6-bit digital output
- Up to 32K bytes nonvolatile RAM
- All-digital I/O optically isolated

Daughter boards for other applications such as binary image acquisition and general-purpose I/O are currently under development. A serial port-based software monitor program is available to aid with the development of embedded control algorithms.

Figure 6–2. MX31 Fitted With a Preliminary CCD Camera Interface Daughter Board

*Insert photo
H1*

Get this photo from the TMS320 3rd
Party Support Reference Guide (job
61119); page 3–137; Figure 3–34.
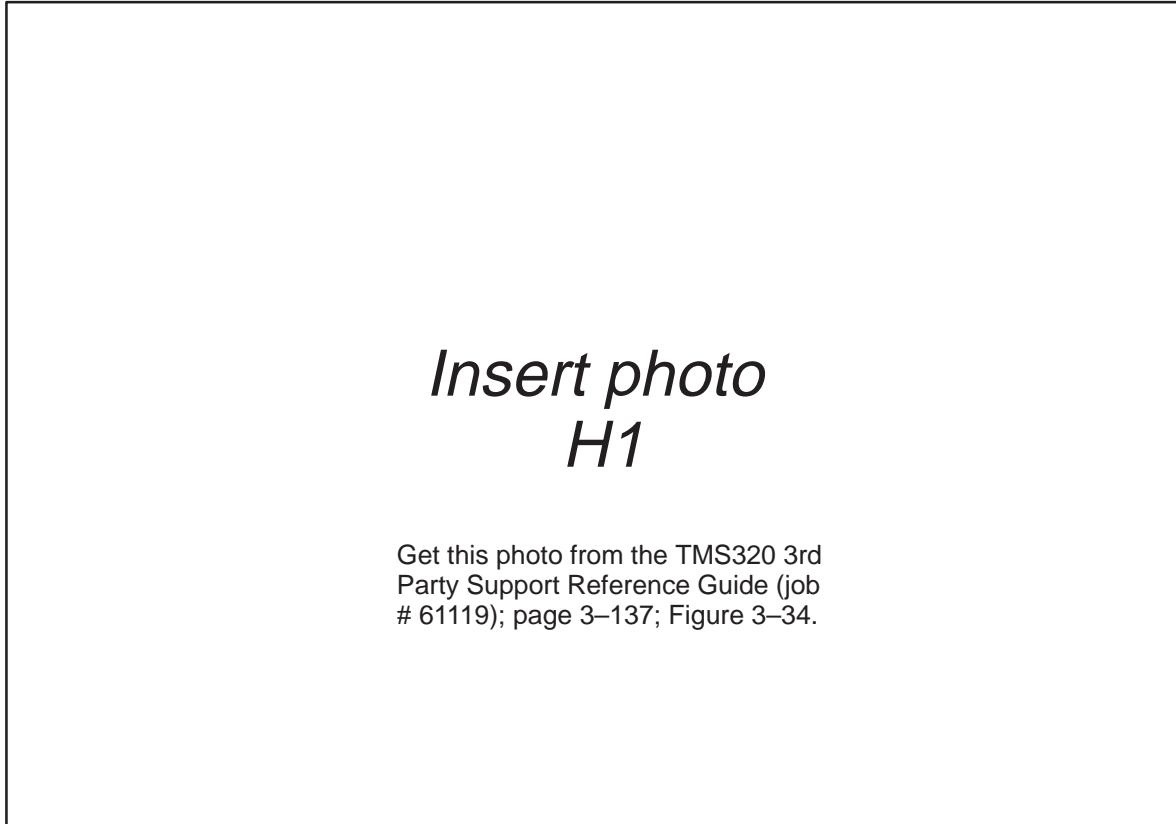
6.8 Loughborough Sound Images Ltd.
The Technology Centre
Epinal Way, Loughborough
Leicestershire, LE11 OQE
England
(44) 509 231843

❑ **TMS320C31 PC/AT Embedded DSP Board**

The PC/C31 is a 3/4 length PC/AT-compatible board intended for embedded signal processing and control applications. The board's architecture gives complete access to all of the TMS320C31's facilities and adds a variety of peripheral interface options.

The PC/C31 is ideal for a wide range of embedded applications, from real-time closed loop control to online signal processing. The high-performance, low-cost 32-bit floating-point TMS320C31's features make it ideally suited to application areas not previously considered. Coupled with LSI's range of peripherals, complete application systems can be assembled quickly and easily.

Features include

■ Complete TMS320C31 processing system
■ Small, 3/4 length PC/AT board format
■ Boot EPROM for standalone operation
■ Zero wait-state SRAM up to 640K words
■ Dual-port SRAM host interface
■ High quality on-board analog interfaces
■ Uprated DSPLINK parallel bus expansion
■ Comprehensive software support

The board format has been designed to the familiar PC/AT specification to ease initial evaluation and development work. Existing users of LSI TMS320C30 products can quickly transfer code to the PC/C31 to implement a target system. The 3/4 length board format aids in keeping occupied space to a minimum. True standalone operation is achieved by the use of the boot EPROM. Using the built-in boot loader of the 'C31, the board can be configured to self initialize and begin execution of applications.

The wide range of zero-wait-state SRAM options, from 32K to 640K words, allows any size of system to be specifically configured for the required application. From an intelligent microcontroller in industrial use to a multitasking signal processing design, all can be accommodated in a high-speed solution. The 2K-word dual-port memory host interface allows rapid

communication, allowing a host PC to transfer data to and from the PC/C31 without halting the DSP. This facility is a great asset in systems that use both the DSP and the host machine in a dual-processing arrangement, where efficient communication between the two is needed.

The PC/C31 is fitted with two of LSI's daughter module sites, giving it access to the high-quality interfaces that make up the daughter module range. This presently comprises both delta-sigma and successive approximation devices and is continually expanding. Using the currently available successive approximation modules, it is possible to construct a 4-input/4-output analog system with a maximum sampling frequency of 200 KHz on the inputs and 500 KHz on the outputs. The modules are designed for quality of conversion. Signal-to-noise and distortion figures of 90 dB for the delta-sigma part have been measured with modules mounted on DSP boards and placed within a PC.

Parallel expansion is provided by an updated version of LSI's DSPLINK interface standard. The bus provides a standardized interface to all of LSI's DSP boards and allows the use of a range of readily available peripheral boards including multichannel analog I/O and AES/EBU pro-audio digital interfaces. The DSPLINK specification is published, allowing users to easily interface a custom design to the bus. Improvements to the original DSPLINK include a 32-bit data bus and additional address lines.

Code development support will be provided by the Texas Instruments floating-point DSP tools that include an optimizing ANSI C compiler, assembler, and linker. These tools cover the whole TI floating-point DSP range, making upgrades or changes to/from other devices a simple matter.

Debug of DSP code is supported by LSI's command line MON31 and Windows 3.0-compatible View31. Both provide a comprehensive range of debug features. View31 allows multiple-board debug sessions, and the windows display is configurable to meet the needs of the debug session. Several memory areas can be viewed simultaneously while multiple-register windows let you view just the registers of interest.

The LSI high-level language interface library allows the integration of the DSP functionality into the host PC. Functions are provided to control and pass data to and from the board, and the libraries are provided in both Microsoft and Turbo C formats.

6.9 Precise Software Technologies Inc.

301 Moodie Drive, Suite 308
Nepean, Ontario
Canada, K2H 9C4
(613) 596–2251
(613) 596–6713

❑ **Precise/MPX Realtime Multiprocessor Executive**

Realtime-embedded control applications are increasingly being solved by using DSPs instead of CISC-based 16- and 32-bit processors. The benefits of using DSPs are increased performance, simpler designs, and cost-effective multiprocessor applications. The TMS320C3x devices are cost effective for many embedded applications such as voice or data communications controllers, LAN controllers, peripheral controllers, laser printers, and biomedical devices. Applications that require additional processors to handle high throughput, high interrupt rates, or building block flexibility, can easily use 2 or more TMS320C3x DSP chips to make simple, easy-to-use multiprocessor systems.

The maximum capabilities of the hardware can be realized by using the Precise/MPX executive. Precise/MPX is a library of primitives that are used by a realtime software designer to extend the C language to a realtime concurrent C language with transparent support for multiprocessor applications. Designing applications using a concurrent programming model is the simplest and most natural paradigm for expressing a realtime problem in terms of a high-level programming language, and is the basis for modern programming languages such as Ada, C++, Objective-C, and Smalltalk. The Precise/MPX kernel has been designed such that the benefits of this programming paradigm can be successfully applied to realtime-embedded controller applications. These capabilities are provided in a very efficient ROMable kernel that typically requires only 16K bytes. Additional benefits of using Precise/MPX are

■ Portability—the concurrent paradigm is hardware independent
■ Reusability—task objects communicate with other task objects or physical interrupts via specified interfaces
■ Scalability—any application that uses Precise/MPX can be mapped from one to any number of DSPs without any change to the application software and no increase in the kernel overhead (in fact the overhead decreases).

❑ **Concurrent Program Development**

Precise/MPX provides over 90 primitives to support program development. These can be grouped into the following major categories:

■ Task management
■ Inter-task communication
■ Interrupt management
■ Memory management
■ Server management

A software designer uses the Precise/MPX tasking model, interrupt management primitives, and inter-task communications primitives to solve a realtime problem by breaking it down into concurrent tasks that communicate via well defined messages. A task is simply a C language function implemented as an iterative loop. Inter-task communication primitives pass messages between tasks and implicitly provide concurrency, which simplifies realtime design and implementation.

■ **Task Management**

Precise/MPX has the capability to completely manage the state of tasks while an application is executing. This capability is especially important for realtime applications that require recovery, reconfiguration, or have resource limitations.

Application tasks are defined to the Precise/MPX kernel through a data structure which specifies *priority*, *stack size*, and the *symbolic* name of the first function of the task. All application tasks except for "main" tasks are managed explicitly by the application using the _*Create()* and _*Destroy()* task management primitives.

Tasks are very lightweight. A task context is maintained in a 128-byte task descriptor. An application can _Create() any number and any type of tasks subject only to available memory.

After system initialization, the Precise/MPX kernel will _Create() a user-specified "main" task and dispatch this task. The "main" task is written by the user to create and dispatch all remaining components of the realtime application.

Once a task has been created, it will execute subject to its own priority and the actions it performs. Task switching occurs only when a task executes a Precise/MPX primitive that "readies" a higher priority task or when an interrupt event readies a higher priority task.

■ **Inter-Task Communication**

Inter-task communication and task synchronization are supported with messages passed between tasks. A software designer usually

uses the *_Send()*, *_Receive()*, or *_Reply()* primitives for message passing. These three primitives are the core interface to the Precise/ MPX executive.

The structure of a design is represented by how the application uses inter-task communication. *_Send()* is used to send a message to another task and cause the kernel to ready that task and run it. *_Receive()* is used by a task to request that a message be sent to it and cause the kernel to ready another task. *_Reply()* is used to issue a response from a receiving task to a sending task and to ready the sending task. Thus, with three simple primitives a designer can specify all inter-task communication and all scheduling required for a concurrent application.

■ **Interrupt Management**

Precise/MPX supports dynamic direct connection to interrupts. Interrupts can be either exceptions generated by the DSP or external device interrupts. The software designer is responsible for writing the interrupt service routine, called the *notifier*. Notifiers can be implemented in C or in assembly language. Interrupts and notifiers can be defined during executive initialization or they can be installed by any task during execution.

Notifiers are equivalent to tasks except they do not require the overhead of tasks and are not scheduled by the executive. A task that is ready to receive an interrupt uses the *_Await_interrupt()* primitive. A notifier needs only to perform two actions to reply to a waiting task. First, it calls *_Task_awaiting_interrupt()* to determine which task is waiting. Then, it calls *_Add_ready()* which readies the waiting task.

■ **Memory Management**

Precise/MPX includes a dynamic memory manager that tasks use to allocate extra temporary or private memory areas exclusive of the tasks' stack. The memory management algorithm is a first on request. On release, it groups together the nearest neighbors to minimize memory fragmentation.

■ **Server Management**

Precise/MPX includes primitives that support Client/Server design paradigms. The client/server model is a powerful design method for developing robust reusable applications for communications and peripheral controllers. Clients and servers are Precise/MPX tasks. The only difference is that a server is created with the _Server_create() primitive, and after it is created, it initializes itself differently. Part of this

initialization is registering the Server's service with a registry so that any client task can use the server.

❏ **I/O Components**

The Precise/MPX is augmented with optional I/O software components that support the following services:

■ SDLC
■ LAPB
■ Mil-Std 1553
■ TCP/IP

These components are written almost entirely in C and are completely re-usable for any new hardware configuration.

❏ **Multiprocessing**

The Precise/MPX kernel has been designed to support various commonly used multiprocessor hardware configurations. It is a unique technology, due to the support for multiprocessor applications using DSPs or mixes of DSP and non-DSP processors.

Precise/MPX has been successfully used on multiprocessors based upon VMEbus and NuBus hardware consisting of from two to 20 microprocessors and using the parallel backplane as a high speed interconnection network. It has also been used in proprietary hardware applications where from three to nine microprocessors are interconnected with memory or high-speed serial data interfaces. In all cases, the applications software has been designed independently of the underlying hardware or interconnection network and the designer was able to reconfigure the application to take advantage of the number and type of processors used in the hardware without having to change the design or any applications source code.

6.10 Spectron Microsystems Inc.
5266 Hollister Avenue
Santa Barbara, CA 93111
(805) 967–0503
FAX: (805) 683–4995

❑ **SPOX Architecture**

SPOX is a highly modular and configurable runtime environment that supports the 'C3x hardware platforms and can be integrated with application programs targeted for these systems. While it provides most of the functionality found in many realtime executives used with general-purpose microprocessors, SPOX has been specifically designed for the more demanding environment of TMS320C3x-based DSP systems:

■ Extensive numeric computation
■ Realtime I/O
■ High-frequency data rates
■ Limited program memory
■ Multi-DSP system architectures
■ Integration with an adjoining host computer

Because of its modular software architecture, SPOX can address a wide range of DSP applications—telecommunications, imaging, speech and audio, test and measurement, and multimedia to name a few—without comprising system functionality and performance. The SPOX runtime environment can be reduced to as little as a few thousand words of code for small embedded applications requiring only a limited number of kernel functions. SPOX can also be integrated into a more comprehensive environment that supports larger applications executing a variety of numerically-intensive algorithms and performing system control and communication functions.

Figure 6–3. SPOX Architecture
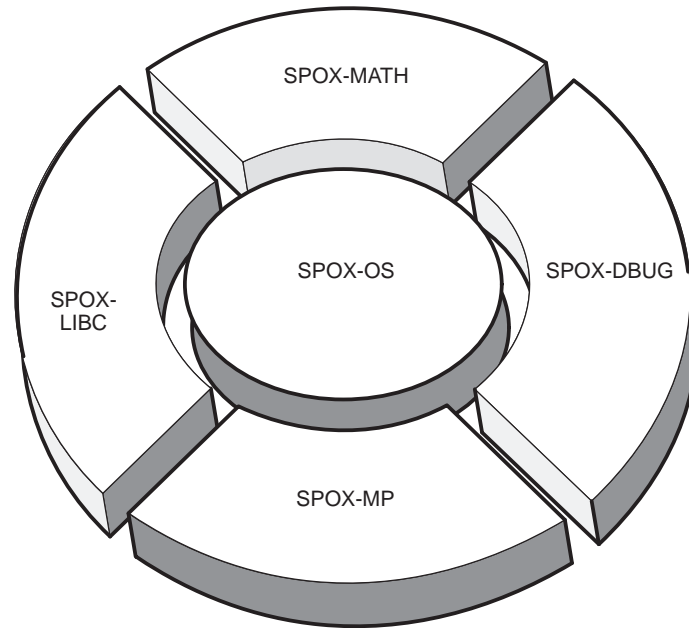
Figure 6–3 depicts the overall architecture of SPOX, illustrating its major functional capabilities along with their organization into the following distinct software components.

■ *SPOX OS* is the foundation of SPOX that provides a set of system capabilities that include: memory management supplying dynamic allocation of arrays from multiple-memory segments; hardware interrupt handling; control of multiple-realtime tasks executing within a single program; and a uniform device-independent stream I/O interface to platform-specific drivers that manage peripherals used for system I/O and communications. It serves as the foundation for the remaining application libraries and system components.

■ *SPOX LIBC* is a library of standard C runtime environment that provides rudimentary file I/O capabilities on the DSP or seamless integration with adjoining host-computer-file system.

■ SPOX MATH is a comprehensive library of optimized DSP math functions that operate on vectors, matrices, and filters.

■ *SPOX DBUG* extends the capabilities of DSP C source debuggers, such as the Texas Instruments db30 to simplify the development of realtime-multitasking SPOX-OS applications. It allows developers to perform debug and profile functions from within the C debugger.

■ *SPOX MP* is a set of software functions that provide a foundation for multi-DSP applications. These include interprocessor communication

primitives, management of shared memory, and the ability to reassign tasks across processor boundaries.

❑ **Realtime Multitasking**

The *SPOX OS* offers all of the features typically found in other realtime multitasking kernels:

■ Preemptive, event-driven scheduling
■ Dynamically prioritized tasks
■ Synchronization and communication facilities
■ Timer services
■ Handling of device interrupts

By offering these features, SPOX OS enables realtime-multitasking applications typically relegated to general-purpose microprocessors to execute on the DSP. Older configurations with 16-bit DSPs, used as slave processors controlled by a more intelligent, general-purpose master, can now be replaced by single-chip 32-bit DSP solutions. Thus, SPOX manages multiple tasks executing numerically-intensive algorithms in parallel with other system control and communication functions.

❑ **Memory Management, Device-Independent I/O, and Host Communication**

While numerical processing may dominate DSP applications, memory allocation, I/O, and communication are equally vital when turning a theoretical algorithm into a practical application. Where the data is located in memory and how this data is input or output have just as much effect on overall system performance as does the algorithm itself.

Using the SPOX memory management functions, application programs create individual-array objects whose respective data buffers can be dynamically allocated and freed during the course of execution. Unlike the standard C functions `malloc()` and `free()`, the SPOX array functions enable the application to supply a parameter specifying the segment of memory in which these buffers will reside. Since production DSP hardware platforms typically contain a hierarchy of memory types (on-chip RAM, external SRAM, bulk DRAM, etc.) retaining explicit control over the location of data becomes essential to meeting realtime constraints in many applications.

SPOX OS supports device-independent I/O, meaning that a *uniform* set of I/O operations are mapped into an otherwise diverse set of devices. The high-level nature of device-independent I/O operations provides a consistent programming interface for a number of off-the-shelf device drivers for accessing and controlling each device within the system and insulates applications from the low-level details of managing these devices.

SPOX OS also provides a mechanism for adding platform-dependent *drivers*—software modules that encapsulate low-level hardware details by interpreting device-independent I/O requests in a device-dependent fashion. Device drivers are the key to customizing SPOX for a particular system environment, and to ensuring portability of SPOX applications from one system to the next.

Unlike virtually every other operating system or realtime executive, the device-independent I/O interface supported by SPOX does not include a `read()` or `write()` function in the traditional sense. Rather than mandating one pair of general-purpose functions for all input and output, SPOX allows for a broader set of I/O operations optimized for two fundamentally different forms of program interaction with underlying devices found in realtime DSP systems:

■ *Asynchronous data streaming*, in which the program and device are in a producer/consumer relationship and

■ *Synchronous message passing*, in which the program and device are in a client/server relationship.

❑ **C Runtime Environment**

The SPOX application libraries include many of the standard functions which are typically not implemented by C compilers targeted for DSP processors. Included among these are the routines comprising the C `stdio` library together with other standard functions requiring operating system support:

■ Opening/closing named files (`fopen`, `fclose`, ...)
■ Reading/writing byte streams (`getc`, `putc`, ...)
■ Formatted I/O (`printf`, `scanf`, ...)
■ Utility functions (`system`, `time`, ...)
■ Program termination (`exit`, `abort`, ...)
■ Memory management (`malloc`, `free`, ...)

By furnishing these functions, the SPOX application libraries enable many standard C programs normally run on a host computer under UNIX or MS-DOS to be literally recompiled and executed faster on attached DSP hardware.

❑ **DSP Math Functions**

SPOX furnishes over 100 standard math functions that can be used as building blocks for algorithms employed in advanced DSP applications, such as:

■ *Vector functions* — arithmetic and logical operations, dot product, convolution, correlation, FFT, windowing, LPC analysis
■ *Matrix functions* — arithmetic and logical operations, row and column manipulation, matrix multiplication, 2-D FFT
■ *Filter functions* — FIR, IIR, and LMS adaptive filtering

The goal of the SPOX math library is to allow DSP application developers to write as much of their program in C as possible without sacrificing overall system performance. To accomplish this goal, all SPOX math functions are optimized in assembly language. Just as importantly, they are tightly integrated into the SPOX memory management and I/O system so that critical data operated by the math algorithms is situated in the appropriate memory, and the overhead incurred in exchanging data between I/O streams and math algorithms is kept to a minimum.

❑ **Multiprocessing Systems**

SPOX addresses the needs of multi-DSP applications with a set of functions that extend the multi-tasking, I/O, and memory management capabilities of SPOX OS from a uniprocessor to a multiprocessor architecture. In a SPOX multiprocessing system, a copy of SPOX-OS is required at each node of the system to manage load resources such as tasks and memory. The following independent software modules are provides:

■ Inter-task communication application programming interface (API)
■ Multiprocessor global shared memory manager
■ Shared memory interprocessor resource locks
■ On-chip peripheral support

❑ **Debug Support**

The C source debugger can provide the following debug and profile capabilities via additional runtime support to the SPOX OS and extensions to the debugger, as shown in Figure 6–4:

■ Display of SPOX OS objects
■ Set task-specific breakpoints
■ Monitor and display system performance characteristics
■ Invoke SPOX OS system calls

Figure 6–4. SPOX Debug Support

Development Host
With In-Circuit Emulation Controller
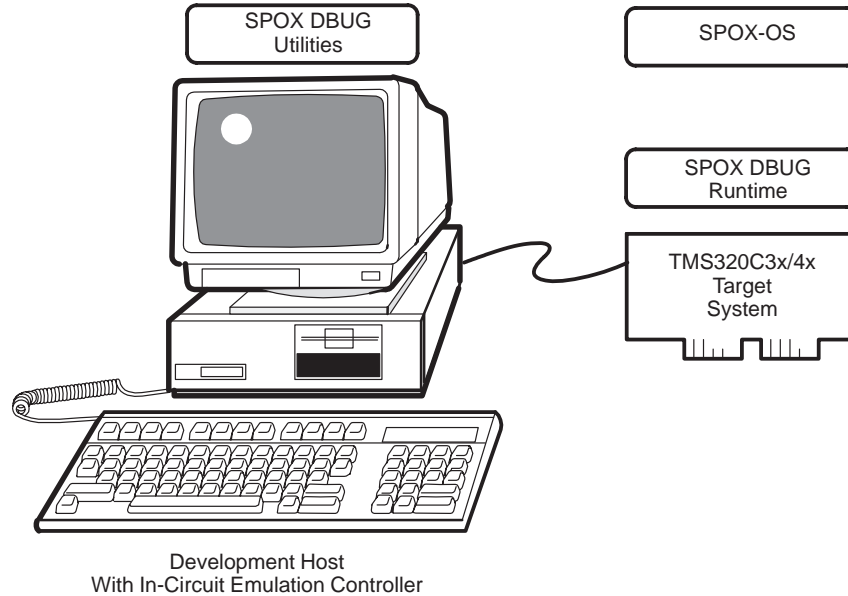
❑ **SPOX Products**

SPOX products that are generally used by application developers and system integrators include:

■ **Software Components for Embedded Systems**

For customers who build and develop realtime embedded DSP systems, SPOX is offered as a suite of software components (shown in Figure 6–3) which can be configured and customized for the customer's hardware.

■ **Application Library Packages**

All major suppliers of plug-in DSP boards offer the complete library of SPOX application functions for C runtime environment, realtime stream I/O, DSP math, and host-DSP communication. These SPOX application library packages are transforming PCs and workstations into signal-processing systems that integrate the flexibility of a host computer with the power of attached DSP hardware.

■ **SPOX Evaluation Kit**

The SPOX EVM evaluation system provides DSP system developers with a low-cost, easy-to-use solution for evaluating the SPOX system kernel on a TMS320C3x hardware platform. The SPOX EVM product

streamlines the evaluation process by integrating all of the necessary hardware and software components into a single turnkey package:

- TMS320C3x EVM hardware platform
- TMS320C3x C compiler and assembly language tools
- SPOX-OS software

SPOX EVM can also serve as a development platform for building rapid prototypes of new DSP systems. All application software developed initially under SPOX EVM can later be reused on any production hardware platform using SPOX.

❑ **Open Signal Processing Architecture (OSPA)**

While improvements in application productivity and portability are proven benefits of SPOX, the true power of a standard software interface to underlying DSP hardware comes with bringing together a wide range of interoperable products. With SPOX serving as the common thread, application developers and system integrators not only can apply these products toward solving today's problems, but are also afforded a bridge to future DSP technologies through the SPOX OSPA (Open Signal Processing Architecture). Figure 6–5 depicts the OSPA framework for interoperability.

Figure 6–5. Open Signal Processing Architecture
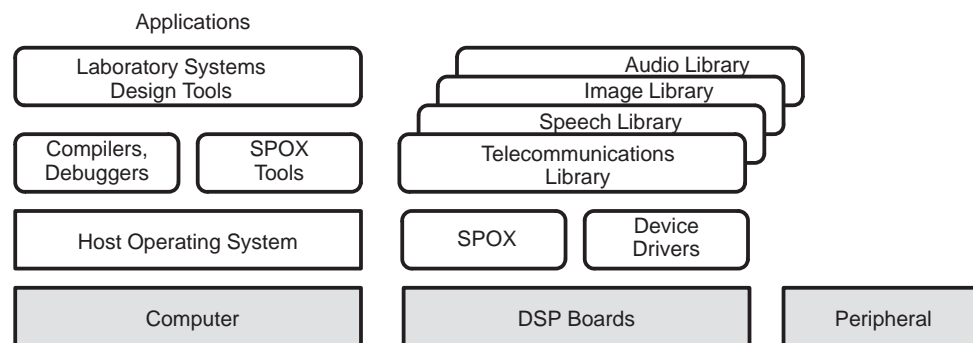
■ **Board-Level Products**

SPOX is rapidly proliferating across a wide variety of board-level products targeted for current and emerging bus architectures (VMEbus, NuBus, EISA, SBus, etc.), allowing developers to buy off-the-shelf DSP platforms and data-acquisition boards rather than building custom hardware.

■ **Program Development Tools**

SPOX supports development of application programs using high-level languages including C and Ada. Several source-level debuggers are also being enhanced with knowledge of the SPOX runtime environment.

■ **DSP Function Libraries**

A growing number of vendors are offering "platform-independent" DSP functions ranging from SPOX-compatible math libraries for audio or image processing to complete implementations of system capabilities such as FAX/modem, speech recognition, and image compression.

■ **Integrated Host Applications**

To facilitate integration of host application programs with realtime DSP software, Spectron provides host computer software that transparently controls and communicates with SPOX tasks executing realtime algorithms on attached DSP hardware.

6.11 Spectrum Signal Processing Inc.
250 'H' Street
P.O. Box 8110–25
Blaine, WA 98230
(800) 663–8986
FAX: (604) 438–3046

❑ DSP/PC Single-Board Computer

Designed for applications such as multimedia, the DSP/PC single-board computer integrates PC technology with DSP technology on a full-size IBM AT plug-in card. A 25-MHz 80386 provides a 100% PC/AT-compatible platform for running DOS programs such as Microsoft Windows, Lotus 1-2-3, and Hypersignal Workstation, while a TMS320C31 provides up to 33 MFLOPS of DSP power.

Features include

■ 2 megabytes of System DRAM, expandable to 8 megabytes
■ High-performance SCSI interface with 32-bit bus-mastering DMA controller
■ Dual floppy disk controller
■ Two serial RS-232 ports
■ Parallel/printer port
■ Realtime clock calendar
■ Keyboard and speaker ports
■ TMS320C31 32-bit floating-point DSP
■ Media_Link high-speed bus expansion connector

❑ DSP-Link Peripherals

Spectrum's DSP-Link peripherals are compatible with the DSP-Link system expansion interface and can be connected to any DSP system or processor board. DSP-Link specifications are available for custom interfacing.

Following are brief descriptions of Spectrum DSP-Link peripherals:

■ 4-Channel Analog I/O Board—Four 12-bit input channels (58 kHz/channel) with quad synchronous sample-and-hold, two 12-bit output channels, third-order low-pass resistor-programmed filters on input and output, DSP-Link data transfer interface.
■ 32-Channel Analog Input Board—32 12-bit input channels (7 kHz/channel) with 4-channel synchronous sample-and-hold, 32 first-order

low-pass resistor-programmed input filters, 32 input buffer amplifiers, DSP-Link data transfer interface.

■ Pro-Audio Board—AES/EBU interface, 48-/44.1-/32-kHz clock, word sync, DSP-Link data transfer interface.

■ Pro-Audio Board—AES/EBU interface, SONY PCM interface, MIDI interface, 16×16 cascadeable RAM, 48-/44.1-/32-kHz clock, word sync, DSP-Link data transfers interface.

■ DSP-Link Prototype Module—DSP-Link slave wire-wrap interface for easy design of custom peripherals, buffered data, decoded address, R/W strobes.

■ DSP-Link Dual-Processor Communications Module—Allows two processors to communicate via DSP-Link.

6.12 Tartan Inc.

300 Oxford Dr.
Monroeville, PA 15146
(412) 856-3600
FAX: (412) 856-3636

❑ **Tartan Compilers**

Tartan, Inc. develops full-function Ada optimizing compilation systems for the TMS320C3x and TMS320C4x DSPs. The compiler targeted to the 'C30 has been validated by the U.S. Government's *Ada Compiler Validation Capability* under test suite version 1.11.

Standard components of the compilation systems are:

■ Highly optimizing compiler
■ Ada Librarian
■ Small, modular runtimes
■ Standard, predefined Ada packages
■ ARTclient package permitting access to tasking data structures and operations
■ Intrinsics package permitting access to hardware capabilities
■ Math package of elementary functions
■ Cross-reference facility
■ AdaScope debugger
■ Linker, object librarian, and utilities
■ Help facility and documentation

The Ada compiler produces fast, compact code through Ada-specific optimizations—optimizations that take advantage of the processor's architecture features, and a full range of classical optimizations. Five optimization levels permit proper optimization strategy at each point in the development cycle.

Support for Ada language features include:

■ Representation specifications for type sizes, record layout, enumeration values, object addresses, and interrupt entries
■ Unchecked deallocation and conversion
■ Insertion of routines written in machine code
■ All Ada predefined pragmas and the implementation-defined pragmas *Foreign_body* and *Linkage_name*

'C3x- and 'C40-specific features include:

■ Access to many processor-specific native instructions
■ Circular and bit-reversed addressing
■ Delayed branch functionality
■ Repeat-block and repeat-single instructions

Compiler switches permit generation of 16-bit PC-relative conditional call instructions, control of interrupt latency time using the RPTS instruction, and specification of the number of wait states for the memory in which the program is executed.

The **Tartan Ada Librarian** implements the Ada language requirements for separate compilation and dependency control. It supports multiple libraries and multiple accesses. It also permits usage of non-Ada object files within an Ada program.

The **Tartan linker** is a fast, flexible linker for embedded Ada programs. It supports precise control over placement of code, data, and constants for individual packages, modules, sections, and subprograms in memory. It eliminates unused program sections from the executable program images, including as much of the highly modularized Tartan Ada runtimes as possible. An interface to the Texas Instruments TMS320C3x cross-assembler is also provided, including conversion of the output to Tartan's object file format.

The **Tartan AdaScope debugger** provides complete window-oriented, source-level, symbolic, and assembly-level debugging for Ada programs using Ada-like commands. It operates remotely from the host system to the DSP processor, using the TI XDS500 controller, or it can be run entirely on the host using the simulator.

The Tartan Ada compilation systems can be hosted on either the Digital Equipment Corporation VAX series equipment running the VMS operating system (version 5.2 or later) or on the Sun SPARC platforms running the SunOS operations system (version 4.1.1 or later).

Available options include an interface to Spectron's SPOX-DSP vector, matrix, and filter math functions; TI simulator; facilities for customizing the runtimes; and the AdaScope retargeting kit to adapt to a different hardware configuration or communications protocol.

❑ **Ada Compiler for the TMS320C30**

Tartan's Ada compiler for the SMJ320C30, the military version of the TMS320C30, supports VAX/VMS and Sun's SPARC systems. The compiler implements Ada as defined in ANSI/MIL-STD-1815A-1983 and is validated under the latest DOD ACVC test suite 1.11.

Tartan Ada C30-targeted compilation systems produce highly optimized application code that runs on the TMS320C30 processors. The compilation system consists of:

■ Full-function optimizing Ada compiler
■ Tartan Ada Library that implements the Ada language requirements for separate compilation and dependency control
■ Tartan Ada Runtime System, including precompiled standard Ada packages for I/O and other facilities and precompiled C30-specific packages
■ Tartan cross-reference facility, TXREF
■ Tartan Ada Runtime Client Package, ARTClient, allowing on-site customizing of the runtime
■ Library of elementary math and trigonometric functions that fully meets the specification of the SIGAda Numerics Working Group and the Ada-Europe Numerics Working Group
■ AdaScope, the Tartan Ada source-level, symbolic debugger
■ Tartan Tool Set, consisting of the Tartan Ada linker, object file librarian, file conversions, and other utilities
■ Online help files for the compiler and library interfaces and AdaScope commands

The Ada compiler produces fast, compact code through Ada-specific optimizations, optimizations that take advantage of 'C30 architecture features, and a full range of classical optimizations. Five optimization levels permit proper optimization strategy at each point in the development cycle.

Code size is further reduced by Tartan's compact, modular runtimes that include only the runtime functionality needed by the application in the executable image. The Tartan linker reduces code size still further by eliminating unused program sections from the executable image.

❏ **'C30-Specific Features**

■ Access to many 'C30 native instructions
■ Circular addressing
■ Bit-reversed addressing
■ 'C30 delayed branch functionality
■ Repeat-block and repeat-single instructions

Compiler switches permit generation of 16-bit PC-relative conditional call instructions, control of interrupt latency time using the RPTS instruction, and specification of the number of wait states for the memory in which the program code is executed.

❏ **Ada Language Features**

■ Representation specifications
■ Unchecked deallocation and conversion
■ Insertion of routines written in machine code

Available options include an interface to the Spectron SPOX-DSP vector, matrix, and filter math functions; TI simulator; facilities for customizing the runtimes; and the AdaScope hardware interface.

Figure 6–6. AdaScope Debugger Screen

*Insert photo
M2*

Get this photo from the TMS320 3rd Party
Support Reference Guide (job # 61119);
page 3–257; Fig.3–66.
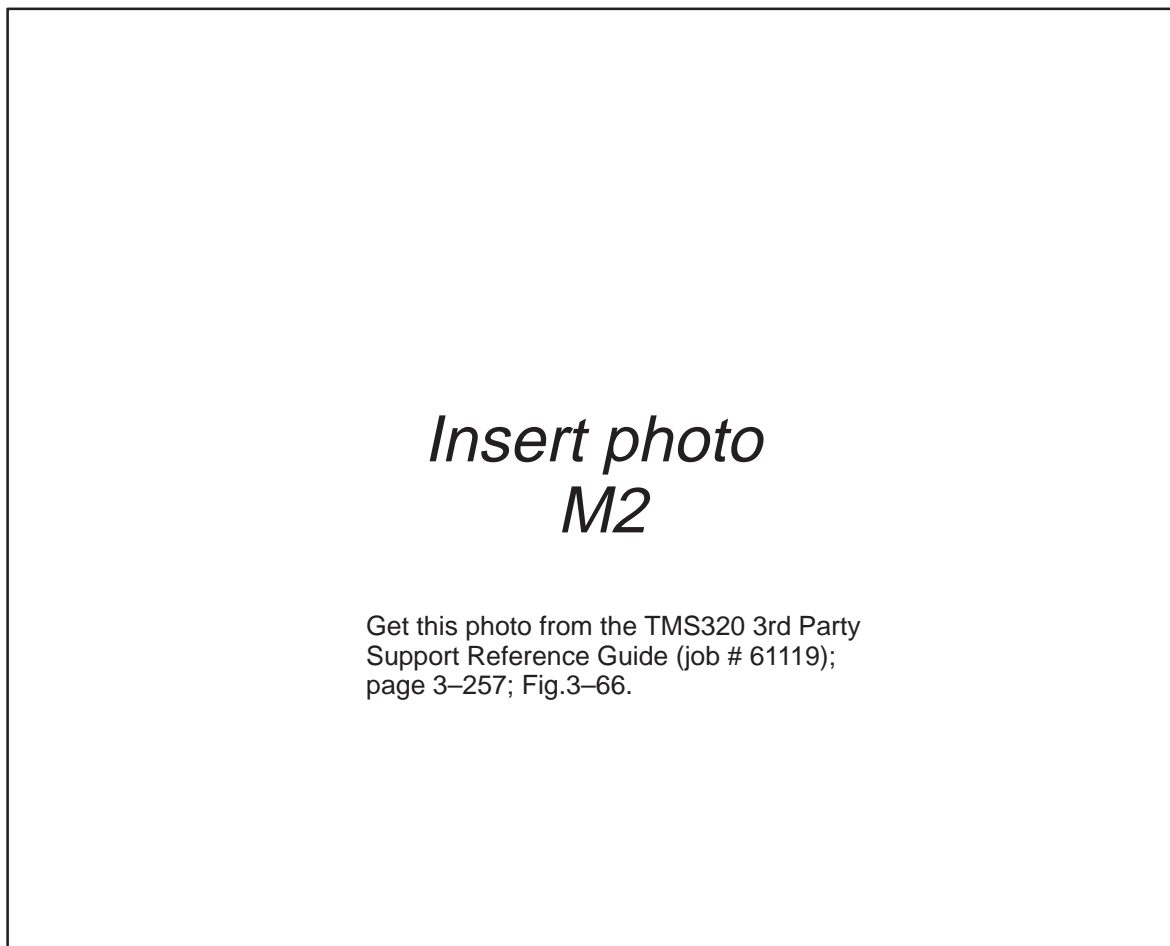
6.13 Tektronix
P.O. Box 500
Beaverton, OR 97077
(800) 835-9433
(503) 627-7111

Tektronix offers realtime, symbolic debugging support for TMS320 development with their comprehensive line of logic analyzers, including the DAS9200 and PRISM 300. Tektronix logic analyzers provide powerful fault-triggering capabilities coupled with comprehensive mnemonic disassembly support, including performance, state, timing and analog analysis for hardware, software, and integration applications. It is ideal for the testing and debugging of algorithms on TMS320 hardware. See Figure 6–7.

❑ **DAS9200**

■ Realtime symbolic debugging
■ Support of up to 5000 symbols from your compiler/assembler with LA-LINK
■ Four disassembly display modes
■ 8K, 32K, 128K trace buffers
■ Automatic fetch prediction
■ 200-MHz state analysis
■ 2-GHz timing analysis
■ 100-MHz pattern generation
■ Time correlation of up to ten DSPs
■ Hard disk for storage

❑ **PRISM 3000**

- ■ Realtime symbolic debugging
- ■ Support of up to 1500 symbols from your compiler/assembly with LA-LINK
- ■ Realtime performance analysis
- ■ Four disassembly display modes
- ■ Automatic fetch prediction
- ■ 200-MHz timing analysis
- ■ Time correlation of up to four DSPs
- ■ Choice of lab or field-portable units
- ■ Integrated digital scope module
- ■ Hard disk for storage

Figure 6–7. Logic Analyzer Family

*Insert photo
O2*

Get this photo from the TMS320 3rd–Party Support Reference Guide (job # 61119)
page 3–266; Figure 3–71.

❏ **1240/1241 Logic Analyzer**

Tektronix supports TMS320 development on their 1240/1241 Logic Analyzer. The 1240/1241 Logic Analyzer provides complete state and timing analysis support for hardware, software, and integration applications. It is ideal for the testing and debugging of algorithms on TMS320 hardware. Powerful triggering, dual timebase, and mnemonic disassembly make the 1240/1241 a valuable tool for developing processor-based products.

6.14 Wintriss
4715 Viewridge, #200
San Diego, CA 92123
(800) 733–8089

❑ EVB Evaluation Board

The WECO EVB is a complete, low-cost, PC/AT TMS320 evaluation board. Models are available for the 'C31.

The EVB contains a wire wrap area for system prototyping purposes and full access by standard PC I/O functions. Dual-ported memory provides for convenient communications. Full debug monitor software is included for dynamic debugging.

EVB features include

- 1-M static RAM
- Wire wrap area
- Dual-port memory
- Dynamic debug software
- C compiler
- Up to 40-MHz operation

TMS320 DSP Family

Digital signal processors are programmable microprocessors designed for speed and flexibility. While they provide functionality similar to traditional microprocessors, they are distinguished by architectural differences which optimize their ability to quickly process complex mathematical formulas.

This appendix describes the evolution of the DSP market and the role of TI in this market. The TMS320 roadmap and a description of each generation of devices are also presented.

A.1 The DSP Market

Over the last decade, DSP technology has made new products possible and many applications affordable. In the early 1980s, DSPs provided an off-the-shelf alternative to custom chips and bit-slice processors. They quickly won acceptance in high-performance applications such as military systems. High-volume applications such as modems soon followed, as the cost of TI DSPs declined dramatically. A processor costing $500 in 1982 now costs $5 (quantity 1)—and as little as $3 in volume. Similar price reductions will transform former niche applications such as multimedia into a widespread standard in the near future.

In addition to lower prices, improvements in ease-of-use and increased system integration have enabled DSPs to displace traditional microcontrollers in many applications. As systems become more numeric intensive, the DSP alternative is increasingly attractive. Evidence of this trend can be seen in semiconductor manufacturers' attempts to incorporate DSP-like functionality into traditional controllers.

DSPs are clearly moving into the mainstream. The evidence suggests that DSPs will be to the 1990s what general-purpose microprocessors were to the 1970s and 1980s.

A.2 The TI Role in the DSP Industry

Advanced technology products and extensive development support have made Texas Instruments a dominant force in the DSP industry.

TI has played a vital role in educating new users and has made a substantial investment in new product development since patenting their first digital signal processor in 1982. In a dedicated effort to train upcoming designers in DSP technology, TI provided students and professors at more than 200 universities with resources to study the technology and offer suggestions for improvements and new applications. University work, along with efforts of third-party developers, helped define new applications far beyond the niche markets of the early 80s.

A broad application base led to significant cost reductions by 1987, because the higher volume enabled more efficiencies through mass production. Continous advances in fabrication process technology contributed to low-cost mass production and enabled TI to incorporate numerous functions on a single DSP. As the number of functions performed by a single processor increased, products could be designed to be lightweight and portable, which made the DSP appeal to a growing number of consumer OEMs. Texas Instruments world-class development support led to shorter design cycles and contributed to the progress in customer product technologies. The market exploded.

Today more than 10,000 designers have gained the benefits that TMS320 DSPs bring to applications. More than 100 independent software and hardware third parties support the development of products incorporating TI DSPs. TI also offers seminars and workshops on product applications and assists potential customers who want to incorporate DSPs in their products.

TI is firmly committed to the future of DSP, and will continue to develop new devices and applications that will drive technology into the next century.

A.3 The TMS320 Product Roadmap

The TMS320 family of 16-/32-bit single-chip digital signal processors combines the flexibility of a high-speed controller with the numerical capability of an array processor, offering an inexpensive alternative to microcontrollers, custom VLSI, and bit-slice processors.

The combination of the TMS320's high degree of parallelism and its specialized digital signal processing instruction set provide speed and flexibility to produce a CMOS microprocessor family that is capable of executing up to 50 MFLOPS or 275 MOPS. The TMS320 family optimizes speed by implementing functions in hardware that other processors implement through software or microcode. This hardware-intensive approach provides the design engineer with power previously unavailable on a single chip. The newest TI generation of floating-point DSPs — TMS320C4x — is designed for high-performance, parallel-processing applications.

The TMS320 family consists of five generations (three fixed-point and two floating-point) of digital signal processors. The fixed-point devices are members of the TMS320C1x, TMS320C2x, or TMS320C5x generation, and the floating-point devices belong to the TMS320C3x or TMS320C4x generation. Figure A–1 shows the TMS320 family. Table A–1 provides a tabulated overview of each member's memory capacity, number of I/O ports (by type), cycle time, package type, technology, and availability.

Many features are common among these TMS320 processors. When the term TMS320 is used, it refers to all five generations of DSP devices. When referring to a specific member of the TMS320 family (e.g.,TMS320C15), the name also implies enhanced-speed in MHz (-14, -25, etc.), erasable/programmable (TMS320E15), low-power (TMS320LC15), and one-time-programmable (TMS320P15) versions. Specific features are added to each processor to provide different cost/performance alternatives. Software compatibility is maintained throughout the family to protect your investment. Each processor has code-generation, system integration, and debug tools to facilitate the design process.

Figure A–1. TMS320 Device Evolution
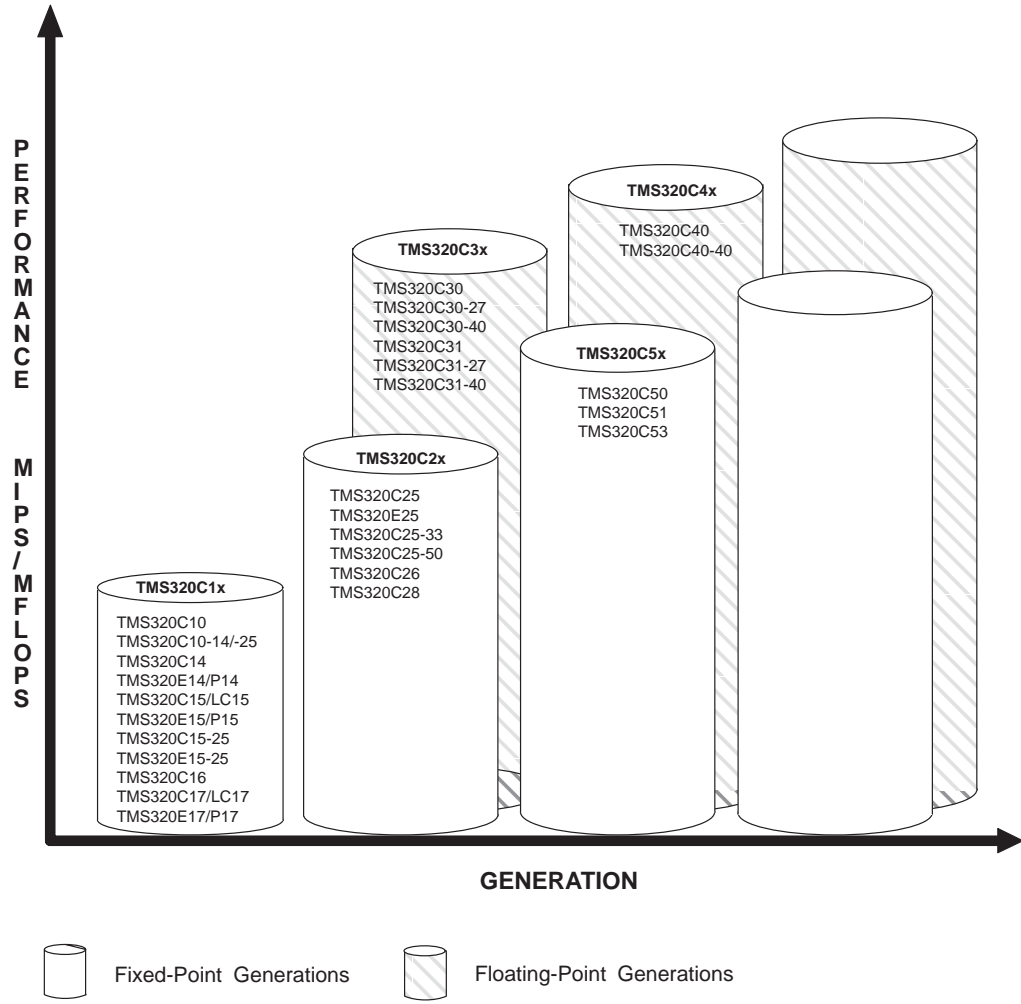
Table A–1. TMS320 Family Overview

Data Type	Device	Memory (words)				I/O‡				On-Chip Timer	Cycle Time (ns)	Package Type
		On-Chip			Off-Chip	Ser	Par	DMA	Com			
		RAM	ROM	EPROM	Dat / Pro							
Fixed-Point (16-bit word size)	TMS320C10†	144	1.5K	–	– / 4K	–	8×16	–	–	–	200	DIP/PLCC
	TMS320C10-14	144	1.5K	–	– / 4K	–	8×16	–	–	–	285	DIP/PLCC
	TMS320C10-25†	144	1.5K	–	– / 4K	–	8×16	–	–	–	160	DIP/PLCC
	TMS320C14	256	4K	–	– / 4K	1	7×16	–	–	4	160	PLCC
	TMS320E14†	256	–	4K	– / 4K	1	7×16	–	–	4	160	CERQUAD
	TMS320P14	256	–	4K	– / 4K	1	7×16	–	–	4	160	PLCC
	TMS320C15†	256	4K	–	– / 4K	–	8×16	–	–	–	200	DIP/PLCC
	TMS320C15-25†	256	4K	–	– / 4K	–	8×16	–	–	–	160	DIP/PLCC
	TMS320E15†	256	–	4K	– / 4K	–	8×16	–	–	–	200	DIP/CER-QUAD
	TMS320E15-25	256	–	4K	– / 4K	–	8×16	–	–	–	160	DIP/CER-QUAD
	TMS320LC15	256	4K	–	– / 4K	–	8×16	–	–	–	250	DIP/PLCC
	TMS320P15	256	–	4K	– / 4K	–	8×16	–	–	–	200	DIP/PLCC
	TMS320C16	256	8K	–	– / 64K	–	8×16	–	–	–	114	PQFP
	TMS320LC16	256	8K	–	– / 64K	–	8×16	–	–	–	250	PQFP
	TMS320C17	256	4K	–	– / –	2	6×16	–	–	1	200	DIP/PLCC
	TMS320E17	256	–	4K	– / –	2	6×16	–	–	1	200	DIP/CER-QUAD
	TMS320LC17	256	4K	–	– / –	2	6×16	–	–	1	278	DIP/PLCC
	TMS320P17	256	–	4K	– / –	2	6×16	–	–	1	200	DIP/PLCC

† military version available/planned; contact nearest TI Field Sales Office for availability
‡ Ser = serial; Par = parallel; DMA = direct memory access (Int = internal; Ext = external); Com = parallel communication ports

Table A–1. TMS320 Family Overview (Concluded)

Data Type	Device	Memory (words)				I/O‡				On-Chip Timer	Cycle Time (ns)	Package Type
		On-Chip			Off-Chip	Ser	Par	DMA	Com			
		RAM	ROM	EPROM	Dat / Pro							
Fixed-Point (16-bit word size)	TMS320C25†	544	4K	–	64K / 64K	1	16×16	Ext	–	1	100	PGA/PLCC/PQFP
	TMS320C25-33	544	4K	–	64K / 64K	1	16×16	Ext	–	1	120	PLCC/PQFP
	TMS320C25-50†	544	4K	–	64K / 64K	1	16×16	Ext	–	1	80	PLCC/PQFP
	TMS320E25	544	–	4K	64K / 64K	1	16×16	Ext	–	1	100	CERQUAD
	TMS320C26†	1.5K	–	–	64K / 64K	1	16×16	Ext	–	1	100	PLCC
	TMS320C28	544	8K	–	64K / 64K	1	16×16	Ext	–	1	100	PQFP
	TMS320C50†	10K	2K	–	64K / 64K	2	64K×16¶	Ext	–	1	35/50	PQFP
	TMS320C51	2K	8K	–	64K / 64K	2	64K×16¶	Ext	–	1	35/50	PQFP
	TMS320C53	4K	16K	–	64K / 64K	2	64K×16¶	Ext	–	1	35/50	PQFP
Floating-Point (32-bit word size)	TMS320C30†	2K	4K	–	16M#	2	16M×32*	Int/Ext	–	2(6)‖	60	PGA
	TMS320C30-27	2K	4K	–	16M#	2	16M×32*	Int/Ext	–	2(6)‖	74	PGA
	TMS320C30-40	2K	4K	–	16M#	2	16M×32*	Int/Ext	–	2(6)‖	50	PGA
	TMS320C31†	2K	§	–	16M#	1	16M×32	Int/Ext	–	2(4)‖	60	PQFP
	TMS320C31-27	2K	§	–	16M#	1	16M×32	Int/Ext	–	2(4)‖	74	PQFP
	TMS320C31-40	2K	§	–	16M#	1	16M×32	Int/Ext	–	2(4)‖	50	PQFP
	TMS320C40	2K	4K§	–	4G#	–	4G×32*	Int/Ext	6	2	40	PGA
	TMS320C40-40†	2K	4K§	–	4G#	–	4G×32*	Int/Ext	6	2	50	PGA

† military version available/planned; contact nearest TI Field Sales Office for availability

‡ Ser = serial; Par = parallel; DMA = direct memory access concurrent with CPU operation (Int = internal; Ext = external); Com = parallel communication ports

¶ sixteen of these parallel I/O ports are memory-mapped

single logical memory space for program, data, and I/O; minus on-chip RAM, peripherals, and reserved spaces

‖ includes the use of serial port timers

* Dual buses

§ Contains an on-chip bootloader ROM

Note: Programmed transcoders (TMS320SS16 and TMS320SA32) are also available.

Table A–2. TMS320 Family Features and Benefits

Feature	Benefit
Five generations of more than 25 compatible devices.	DSP to meet any application need.
Cycle times as fast as 35 ns. Choice of fixed-point or floating-point devices. Hardware multiplier and barrel shifters. Modified Harvard architecture. Concurrent DMA, program cache.	Realtime DSP performance.
On-chip data RAM up to 8.5K words, program ROM/EPROM up to 4K words. Serial port, timer, multiprocessor interface, instruction cache, DMA controller. CMOS processing.	Reduced system cost, space, and power consumption.
Large memory space up to 4 gigawords.	Multiple DSP programs on a single chip.
General-purpose and DSP-specific instructions. EPROM and OTP versions. High-level language support. Operating system support. Extensive development support.	Ease of design. Fast time to market.
JTAG IEEE test bus. Serial scan path for 99% fault grading.	System reliability.

A.4 TMS320C1x

The TMS320C1x DSPs provide cost-effective solutions for many needs. TMS320C1x DSPs perform a multiply command at least 30 times faster than a general-purpose microprocessor. An on-chip hardware multiplier allows the TMS320C1x to produce results in a single instruction cycle. Instruction cycle times range from 160 to 280 ns. Higher performance is achieved through internal parallelism and a unique Harvard architecture, which allows program fetch to overlap data operations. The 'C1x generation includes DSPs optimized for specific high-performance applications such as speech synthesis, high-speed modems, and telephone systems. All TMS320C1x devices are software compatible for easy upgrade as application requirements change. TMS320C1x ROM-code versions can be used to reduce system costs. On-chip serial ports, companding hardware, and a coprocessor interface make the TMS320C17 ideal for telecommunications applications.

The TMS320C14 has been optimized for control applications such as disk drives and servo control. The 'C14 is the industry's first device to combine the high performance of a DSP with the on-chip peripherals of a microcontroller. Operating at 25.6 MHz, the TMS320C14 offers five to ten times the speed of traditional 16-bit microcontrollers and can execute advanced control algorithms (such as Kalman filters and state controllers) for analog-type performance. On-chip peripherals (such as event manager with PWM, bit I/O, watchdog timer, serial port, and baud rate generator) reduce chip count, resulting in space and cost savings.

With 4K words of on-chip EPROM, the TMS320E15, 'E17, and 'E14 support realtime code development.

A.5 TMS320C2x

The TMS320C2x DSPs offer from two to four times the performance of the 'C1x devices. Since the TMS320C2x devices are source-code compatible with TMS320C1x DSPs, they provide an ideal upgrade path for the world's largest installed base of signal processors. The TMS320C2x DSPs offer instruction cycle times as fast as 80 ns, two to four times the amount of on-chip RAM, larger external memory reach (160K), multiprocessor capabilities, and several additional application-specific instructions and addressing modes. The 'E25 offers 4K words of on-chip EPROM for realtime code development and prototyping ease. The TMS320C2x ROM versions can be used for system cost reduction. The 'C2x DSPs vary in instruction time and memory size and type. Specifically, the TMS320C25-50 supports 50-MHz (80-ns) operation. The TMS320C26 offers 1.5K words of on-chip data RAM, 256 words of on-chip ROM, and up to 128K words of data/program RAM.

A.6 TMS320C3x

The TMS320C3x DSPs incorporate floating-point arithmetic and offer the features of a super computer on a single chip, executing more than 33 MFLOPS. High performance is gained through large on-chip memories (2K words of RAM and 4K words of ROM), a concurrent DMA controller, and instruction cache (64 words). Two serial ports, two timers, a DMA controller, and large on-chip system memory are achieved by using a high-density CMOS process incorporating 700,000 transistors. This high level of on-chip integration reduces system cost, space, and power requirements. Because the 'C3x devices are floating-point DSPs, numbers no longer need to be scaled, thereby simplifying code development. Future 'C3x devices will support applications needing faster cycle times, lower cost, and extreme temperature and reliability characterization. TMS320C3x development is supported by high-level language compilers (C and Ada) and the SPOX realtime operating system. Scan-based emulation is possible through a unique on-chip serial scan path, which provides access to all chip registers.

A.7 TMS320C4x

The TMS320C4x DSPs are the world's first floating-point DSPs designed for parallel processing. The 'C4x devices include 40- and 50-MHz versions. The 'C4x CPU features a 40-/50-ns single-cycle floating-point instruction execution time with 275/225 MOPS and 320/256 Mbytes/sec, respectively. There are six communication ports for direct interprocessor or processor-I/O communications peripherals. A self-programmable six-channel DMA coprocessor maximizes sustained CPU performance. The 512-byte instruction cache memory with two independent 32-bit memory interfaces support shared memory configurations. The 'C4x 40-MHz version is designed for slower speed DSP applications that would benefit from the attributes of a lower-priced, floating-point TMS320C40 processor.

A.8 TMS320C5x

The TMS320C5x DSPs are the industry's highest-performance fixed-point DSPs. Designed to execute an instruction in 35 ns, the 'C5x is software upwardly compatible with all 'C1x and 'C2x DSPs, providing a fast performance upgrade path. Fast cycle times, large on-chip memories, a parallel logic unit (PLU), zero overhead context switching, and block repeats differentiate the TMS320C5x. The 'C5x has 2 serial ports which can operate in normal or time division multiplexed (TDM) modes. The integration of the JTAG IEEE test bus standard increases system reliability, allowing 99% fault grade testing and on-chip emulation. Spin-off devices can be developed rapidly because of the modular design of the 'C5x.

Part Ordering Information

This chapter provides the device and support tool part numbers. Table B–1 lists the part numbers for the TMS320C30 and TMS320C31, and Table B–2 gives ordering information for TMS320C3x hardware and software support tools. An explanation of the TMS320 family device and development support tool prefix and suffix designators follows the two tables to assist you in understanding the TMS320 product numbering system.

The topics covered and their page numbers include:

B.1 Part Numbers

Table B–1. TMS320C3x Digital Signal Processor Part Numbers

Device	Technology	Operating Frequency	Package Type	Typical Power Dissipation
TMS320C30GEL	1.0-µm CMOS	33 MHz	Ceramic 181-pin PGA	1.00 W
TMS320C30GEL27	1.0-µm CMOS	27 MHz	Ceramic 181-pin PGA	0.88 W
TMS320C30GEL40	1.0-µm CMOS	40 MHz	Ceramic 188-pin PGA	1.25 W
TMS320C31PQL	0.8-µm CMOS	33 MHz	Plastic 132-pin QFP	0.75 W
TMS320C31PQL27	0.8-µm CMOS	27 MHz	Plastic 132-pin QFP	0.63 W
TMS320C31PQL40	0.8-µm CMOS	40 MHz	Plastic 132-pin QFP	–
SMJ320C30GBM28 SMJ320C30HUM28 SMJ320C30HTM28	1.0-µm CMOS	28 MHz	Ceramic 181-pin PGA or Ceramic 196-pin QFP	1.00 W 1.00 W
SMJ320C30GBM25 SMJ320C30HUM25 SMJ320C30HTM25	1.0-µm CMOS	25 MHz	Ceramic 181-pin PGA or Ceramic 196-pin QFP	1.00 W 1.00 W
TMS320C31PQA	0.8-µm CMOS	33 MHz	Plastic 132-pin QFP	1.00 W

Table B–2. TMS320C3x Support Tool Part Numbers

Tool Description	Operating System	Part Number
Software		
C Compiler & Macro Assembler/ Linker	VAXVMS PC-DOS/MS-DOS SUN UNIX[†] MAC-MPW	TMDS3243255-08 TMDS3243855-02 TMDS3243555-08 TMDS3243565-01
Macro Assembler/Linker	PC-DOS/MS-DOS; OS/2	TMDS3243850-02
Simulator	VAX VMS PC-DOS/MS-DOS SUN UNIX[†]	TMDS3243251-08 TMDS3243851-02 TMDS3243551-09
SPOX OS Software for 'C3x Target Board	PC-DOS/MS-DOS	TMDS3240132

[†] Note that SUN UNIX supports TMS320C3x software tools on the 68000 family-based SUN-3 series workstations and on the SUN-4 series machines that use the SPARC processor, but not on the SUN-386i series of workstations.

Table B–2. TMS320C3x Support Tool Part Numbers (Concluded)

Tool Description	Operating System	Part Number
Hardware		
Evaluation Module (EVM)	PC-DOS/MS-DOS	TMDS3260030
HP Subsystem	PC-DOS/MS-DOS	TMDX326HP30
'C31 Adapter for HP Subsystem	PC-DOS/MS-DOS	TMDX326HP31
TMS320C3x XDS Emulator	PC/MS-DOS	TMDS3260131
'C3x Application Board With Software Demo	PC/MS-DOS	TMDS3260132

† Note that SUN UNIX supports TMS320C3x software tools on the 68000 family-based SUN-3 series worksta-
tions and on the SUN-4 series machines that use the SPARC processor, but not on the SUN-386i series of
workstations.

B.2 Device and Development Support Tool Prefix Designators

Prefixes to Texas Instruments' part numbers designate phases in the product's development stage for both devices and support tools, as shown in the following definitions:

Device Development Evolutionary Flow:

TMX   Experimental device that is not necessarily representative of the final device's electrical specifications.

TMP   Final silicon die that conforms to the device's electrical specifications but has not completed quality and reliability verification.

TMS   Fully qualified production device.

Support Tool Development Evolutionary Flow:

TMDX   Development support product that has not yet completed Texas Instruments' internal qualification testing for development systems.

TMDS   Fully qualified development support product.

TMX and TMP devices and TMDX development support tools are shipped with the following disclaimer:

"Developmental product is intended for internal evaluation purposes."

Note:

Texas Instruments recommends that prototype devices (TMX or TMP) not be used in production systems because their expected end-use failure rate is undefined but predicted to be greater than standard qualified production devices.

TMS devices and TMDS development support tools have been fully characterized, and their quality and reliability have been fully demonstrated. Texas Instruments' standard warranty applies to TMS devices and TMDS development support tools.

TMDX development support products are intended for internal evaluation purposes only. They are covered by Texas Instruments' Warranty and Update Policy for Microprocessor Development Systems products; however, they should be used by customers only with the understanding that they are developmental in nature.

B.3 Device Suffixes

The suffix indicates the package type (e.g., N, FN, or GB) and temperature range (e.g., L).

Figure B–1 presents a legend for reading the complete device name for any TMS320 family member.

Figure B–1. TMS320 Device Nomenclature

```
                    TMS    320    C    30    GB    L
```

Prefix

TMX = Experimental Device
TMP = Prototype Device
TMS = Qualified Device
SMJ = MIL-STD-883C

Device Family

320 = TMS320 Family

Technology

C   =   CMOS
E   =   CMOS EPROM
LC =   Low-power CMOS
P   =   One-time programmable

Device

1st-generation DSP:
  10
  14
  15
  16
  17
2nd-generation DSP:
  20
  25
  26
  28
3rd-generation DSP:
  30
  31
4th-generation DSP:
  40
5th-generation DSP:
  50
  51
  53

Temperature Range

H = 0 to 50°C
L = 0 to 70°C
S = -55 to 100°C
M = -55 to 125°C
A = -40 to 85°C

Package Type

N   = Plastic DIP
JD = Ceramic DIP Side-Brazed
FN = Plastic Leaded CC
GB = Ceramic PGA
FJ = Ceramic Leaded CC
FD = Leadless Ceramic CC
FZ = Ceramic Leaded CC
GE = Ceramic PGA, Glass Seal
HU = Ceramic quad flatpack
HT = Ceramic quad flatpack
       (gull wing)
PQ = Plastic quad flatpack

Index

D

E

X

XDS
  analysis subsystem, 5-31
  emulator, 5-26, 5-27
  HP 64776, 5-31
  scan-based emulators, 5-26
  system requirements, 5-29