IMPORTANT NOTICE

Texas Instruments (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 the specifications applicable at the time of sale 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.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage (“Critical Applications”).

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer’s applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

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.

Copyright © 1997, Texas Instruments Incorporated
About This Manual

This user’s guide serves as a reference book for developing hardware and/or software applications for the TMS320C5x digital signal processors (DSPs).

How to Use This Manual

The following table summarizes the 'C5x information contained in this user’s guide:

<table>
<thead>
<tr>
<th>If you are looking for information about:</th>
<th>Turn to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application reports</td>
<td>Appendix C, Application Reports and Designer’s Notebook Pages</td>
</tr>
<tr>
<td>Designer’s notebook pages</td>
<td>Appendix C, Application Reports and Designer’s Notebook Pages</td>
</tr>
<tr>
<td>DSP features</td>
<td>Chapter 1, Introduction</td>
</tr>
<tr>
<td>External memory interfacing</td>
<td>Chapter 3, External Memory interface</td>
</tr>
<tr>
<td>Extended-precision arithmetic subroutines</td>
<td>Chapter 2, Software Applications</td>
</tr>
<tr>
<td>Fast Fourier transform subroutine</td>
<td>Chapter 2, Software Applications</td>
</tr>
<tr>
<td>Floating-point arithmetic subroutines</td>
<td>Chapter 2, Software Applications</td>
</tr>
<tr>
<td>Hardware applications</td>
<td>Chapter 3, Analog Interface Peripherals and Applications</td>
</tr>
<tr>
<td>Infinite impulse response (IIR) filter subroutines</td>
<td>Chapter 2, Software Applications</td>
</tr>
<tr>
<td>Memory-to-memory block move subroutines</td>
<td>Chapter 2, Software Applications</td>
</tr>
<tr>
<td>Modem applications</td>
<td>Chapter 2, Software Applications</td>
</tr>
</tbody>
</table>

Chapter 4, Analog Interface Peripherals and Applications
### Notational Conventions

This document uses the following conventions:

- **Program listings and program examples are shown in a special typeface.**

  Here is a segment of a program listing:

  ```plaintext
  OUTPUT:
  LDP   #6  ;data page 6
  BLDD  #300, 20h ;move data at address 300h to 320h
  RET
  ```

- **In syntax descriptions, the instruction is in a bold typeface and parameters are in an italic typeface.** Portions of a syntax in **bold** must be entered as shown; portions of a syntax in **italics** describe the type of information that you specify. Here is an example of an instruction syntax:

  ```plaintext
  [label] BLDD src, dst
  ```

  **BLDD** is the instruction and has two parameters, *src* and *dst*. When you use **BLDD**, the first parameter must be an actual data memory source address and *dst* a destination address. A comma and a space (optional) must separate the two addresses.
The term OR is used in the assembly language instructions to denote a Boolean operation. The term or is used to indicate selection. Here is an example of an instruction with OR and or:

lk OR (src) → src or [, dst]

This instruction ORs the value of lk with the contents of src. Then, it stores the result in src or dst, depending on the syntax of the instruction.

Square brackets, [ and ], identify an optional parameter. If you use an optional parameter, specify the information within the brackets; do not type the brackets themselves. In the example above, instead of typing [label], you specify a name for the label. When you specify more than one optional parameter from a list, you separate them with a comma and a space.

Braces, { and }, indicate a list. Unless the list is enclosed in square brackets, you must choose one item from the list; do not type the braces themselves. Here’s an example of a list that provides seven choices:

ind: { * *+ *– *0+ *0– *BRO+ *BRO–}

The term ‘C5x refers to the TMS320C5x.

Related Documentation from Texas Instruments

The following books describe the 'C5x and related support tools. To obtain a copy of any of these TI documents, call the Texas Instruments Literature Response Center at (800) 477–8924. When ordering, please identify the book by its title and literature number.

TMS320C5x User’s Guide (literature number SPRU056) describes the 'C5x 16-bit, fixed-point, general-purpose digital signal processors. Covered are its architecture, internal register structure, instruction set, pipeline, specifications, DMA, I/O ports, and on-chip peripherals.

TMS320C5x, TMS320LC5x Digital Signal Processors (literature number SPRS030) data sheet contains the electrical and timing specifications for these devices, as well as signal descriptions and pinouts for all of the available packages.

Calculation of TMS320C5x Power Dissipation (literature number SPRA030). This application report describes techniques for analyzing system and device conditions to determine operating current levels. From this analysis, power dissipation for the device can be determined. Knowledge of power dissipation can, in turn, be used to determine thermal management requirements for the device.
Telecommunications Applications With the TMS320C5x DSPs (literature number SPRA033). This application book is a collection of DSP applications related to the field of telecommunications and implemented on the TMS320C5x. Topics covered are digital cellular systems, speech synthesis, error-correction coding, baseband modulation and demodulation, equalization and channel estimation, speech and character recognition algorithms, and system design considerations.

PCMCIA TMS320 DSP MediaCard (literature number SPRA052). This application report describes the DSP MediaCard, version 1.0, how it operates, and how to use it. The DSP MediaCard is a card for sound and fax/modem applications, and it uses a TMS320 DSP and on-board stereo codec.

Use of the TMS320C5x Internal Oscillator With External Crystals or Ceramic Resonators (literature number SPRA054). This application report provides information about crystal and ceramic resonators, their frequency characteristics, a general background on oscillators, and the type of oscillator circuit used on the TMS320C5x. Covered are design aspects of the 'C5x oscillator including appropriate configuration of the external components, measured parameters for the on-board portion of the circuitry, use of the oscillator with overtone crystals, and general design considerations for choosing the external components for the oscillator. This report presents some design solutions for common frequencies.

Enhanced Control of an Alternating Current Motor Using Fuzzy Logic and a TMS320 Digital Signal Processor (literature number SPRA057). This application report describes how the use of a digital signal processor with a specialized fuzzy logic software kernel provides the required computing performance for a control system design while maintaining a low cost. This report presents a fuzzy logic design that enhances the system's ability to handle the abrupt momentum changes of an ac motor controller and the software technology used to implement the fuzzy logic design.

Improving 32-Channel DTMF Decoders Using the TMS320C5x (literature number SPRA085). This application report discusses improvements that you can make to a multichannel dual-tone multifrequency (DTMF) decoder by using a TMS320C5x. PBX systems use multiple DTMF chips to encode or decode tones. PBX systems also perform other functions, such as voice compression or expansion and voice mail. By using a 'C5x, you can increase the performance and flexibility of the PBX systems, while decreasing the cost of the systems.
Digital Signal Processing Applications with the TMS320 Family, Volumes 1, 2, and 3 (literature numbers SPRA012, SPRA016, SPRA017) Volumes 1 and 2 cover applications using the 'C10 and 'C20 families of fixed-point processors. Volume 3 documents applications using both fixed-point processors, as well as the 'C30 floating-point processor.

TMS320C1x/C2x/C2xx/C5x Code Generation Tools Getting Started Guide (literature number SPRU121) describes how to install the TMS320C1x, TMS320C2x, TMS320C2xx, and TMS320C5x assembly language tools and the C compiler for the 'C1x, 'C2x, 'C2xx, and 'C5x devices. The installation for MS-DOS™, OS/2™, SunOS™, and Solaris™ systems is covered.

TMS320C1x/C2x/C2xx/C5x Assembly Language Tools User’s Guide (literature number SPRU018) 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 'C1x, 'C2x, 'C2xx, and 'C5x generations of devices.

TMS320C2x/C2xx/C5x Optimizing C Compiler User’s Guide (literature number SPRU024) describes the 'C2x/C2xx/C5x C compiler. This C compiler accepts ANSI standard C source code and produces TMS320 assembly language source code for the 'C2x, 'C2xx, and 'C5x generations of devices.

TMS320C5x C Source Debugger User’s Guide (literature number SPRU055) tells you how to invoke the 'C5x 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. It also includes a tutorial that introduces basic debugger functionality.

TMS320C5x Evaluation Module Technical Reference (literature number SPRU087) describes the 'C5x evaluation module, its features, design details and external interfaces.

TMS320C5x Evaluation Module Getting Started Guide (literature number SPRU126) tells you how to install the MS-DOS™, PC-DOS™, and Windows™ versions of the 'C5x evaluation module.

XDS51x Emulator Installation Guide (literature number SPNU070) describes the installation of the XDS510™, XDS510PP™, and XDS510WS™ emulator controllers. The installation of the XDS511™ emulator is also described.
**JTAG/MPSD Emulation Technical Reference** (literature number SPDU079) provides the design requirements of the XDS510™ emulator controller, discusses JTAG designs (based on the IEEE 1149.1 standard), and modular port scan device (MPSD) designs.

**TMS320 DSP Development Support Reference Guide** (literature number SPRU011) describes the TMS320 family of digital signal processors and the tools that support these devices. Included are code-generation tools (compilers, assemblers, linkers, etc.) and system integration and debug tools (simulators, emulators, evaluation modules, etc.). Also covered are available documentation, seminars, the university program, and factory repair and exchange.

**TMS320 Third-Party Support Reference Guide** (literature number SPRU052) alphabetically lists over 100 third parties that provide various products that serve the family of TMS320 digital signal processors. A myriad of products and applications are offered—software and hardware development tools, speech recognition, image processing, noise cancellation, modems, etc.

**Related Documents and Technical Articles**

If you are an assembly language programmer and would like more information about C or C expressions, you may find this book useful:


A wide variety of related documentation is available on DSPs. These references fall into one of the following application categories:

- General-purpose DSP
- Graphics/imagery
- Speech/voice
- Control
- Multimedia
- Military
- Telecommunications
- Automotive
- Consumer
- Medical
- Development support
In the following list, references appear in alphabetical order according to author. The documents contain beneficial information regarding designs, operations, and applications for signal-processing systems; all of the documents provide additional references.

**General-Purpose DSP:**


**Graphics/Imagery:**


**Speech/Voice:**


**Control:**


**Multimedia:**


**Military:**

**Telecommunications:**


Automotive:

Consumer:

Medical:

Development Support:
Trademarks

DuPont Electronics is a registered trademark of E.I. DuPont Corporation.

HP-UX is a trademark of Hewlett-Packard Company.

IBM, OS/2, and PC-DOS are trademarks of International Business Machines Corporation.

MS and Windows are registered trademarks of Microsoft Corporation.

Solaris and SunOS are trademarks of Sun Microsystems, Inc.

SPARC is a trademark of SPARC International, Inc., but licensed exclusively to Sun Microsystems, Inc.

320 Hotline On-line, TI, XDS510, XDS510PP, XDS510WS, and XDS511 are trademarks of Texas Instruments Incorporated.

VAX and VMS are trademarks of Digital Equipment Corp.
## If You Need Assistance . . .

### World-Wide Web Sites

- **TI Online**
  - [http://www.ti.com](http://www.ti.com)
- **Semiconductor Product Information Center (PIC)**
  - [http://www.ti.com/sc/docs/pic/home.htm](http://www.ti.com/sc/docs/pic/home.htm)
- **DSP Solutions**
  - [http://www.ti.com/dsps](http://www.ti.com/dsps)
- **320 Hotline On-line™**
  - [http://www.ti.com/sc/docs/dsps/support.htm](http://www.ti.com/sc/docs/dsps/support.htm)

### North America, South America, Central America

- **Product Information Center (PIC)**
  - (972) 644-5580
- **TI Literature Response Center U.S.A.**
  - (800) 477-8924
- **Software Registration/Upgrades**
  - (214) 638-0333
  - Fax: (214) 638-7742
- **U.S.A. Factory Repair/Hardware Upgrades**
  - (281) 274-2285
- **U.S. Technical Training Organization**
  - (972) 644-5580
- **DSP Hotline**
  - (281) 274-2320
  - Fax: (281) 274-2324
  - Email: dsph@ti.com
- **DSP Modem BBS**
  - (281) 274-2323
- **DSP Internet BBS via anonymous ftp to ftp://ftp.ti.com/pub/tms320bbs**

### Europe, Middle East, Africa

- **European Product Information Center (EPIC) Hotlines:**
  - Multi-Language Support: +33 1 30 70 11 69
  - Deutsch: +49 8161 80 33 11 or +33 1 30 70 11 68
  - English: +33 1 30 70 11 65
  - Francais: +33 1 30 70 11 64
  - Italiano: +33 1 30 70 11 67
- **EPIC Modem BBS**
  - +33 1 30 70 11 99
- **European Factory Repair**
  - +33 4 93 22 25 40
- **Europe Customer Training Helpline**
  - Fax: +49 81 61 80 40 10

### Asia-Pacific

- **Literature Response Center**
  - +852 2 956 7288
  - Fax: +852 2 956 2200
- **Hong Kong DSP Hotline**
  - +852 2 956 7268
  - Fax: +852 2 956 1002
- **Korea DSP Hotline**
  - +82 2 551 2804
  - Fax: +82 2 551 2828
- **Korea DSP Modem BBS**
  - +82 2 551 2914
- **Singapore DSP Hotline**
  - +65 390 7179
- **Taiwan DSP Hotline**
  - +886 2 377 1450
  - Fax: +886 2 377 2718
- **Taiwan DSP Modem BBS**
  - +886 2 376 2592
- **Taiwan DSP Internet BBS via anonymous ftp to ftp://dsp.ee.tit.edu.tw/pub/TI/**

### Japan

- **Product Information Center**
  - +0120-81-0026 (in Japan)
  - +03-3457-0972 or (INTL) 813-3457-0972
  - Fax: +0120-81-0036 (in Japan)
  - +03-3457-1259 or (INTL) 813-3457-1259
- **DSP Hotline**
  - +03-3769-8735 or (INTL) 813-3769-8735
  - Fax: +03-3457-7071 or (INTL) 813-3457-7071
- **DSP BBS via Nifty-Serve**
  - Type “Go TIASP”

### Documentation

When making suggestions or reporting errors in documentation, please include the following information that is on the title page: the full title of the book, the publication date, and the literature number.

- **Mail:** Texas Instruments Incorporated
  - Technical Documentation Services, MS 702
  - P.O. Box 1443
  - Houston, Texas 77251-1443

- **Email:** comments@books.sc.ti.com

---

**Note:** When calling a Literature Response Center to order documentation, please specify the literature number of the book.
1 Introduction ................................................................................................................. 1-1
Summarizes the features of the TMS320 family of products and presents typical applications.
Describes the TMS320C5x DSP and lists its key features.
1.1 TMS320 Family Overview ......................................................................................... 1-2
1.1.1 History, Development, and Advantages of TMS320 DSPs ................................. 1-2
1.1.2 TMS320 Typical Applications ............................................................................. 1-4
1.2 TMS320C5x Overview .............................................................................................. 1-5
1.3 TMS320C5x Key Features ......................................................................................... 1-7
2 Software Applications .................................................................................................. 2-1
Describes the use of the TMS320C5x instruction set with particular emphasis on its new
features and special applications.
2.1 Processor Initialization .............................................................................................. 2-2
2.2 Interrupts .................................................................................................................. 2-5
2.3 Software Stack .......................................................................................................... 2-8
2.4 Logical and Arithmetic Operations ............................................................................ 2-9
2.4.1 Parallel Logic Unit (PLU) ................................................................................... 2-9
2.4.2 Multiconditional Instructions ............................................................................ 2-12
2.4.3 Search Algorithm Using CRGT .......................................................................... 2-14
2.4.4 Matrix Multiplication Using Nested Loops ......................................................... 2-14
2.5 Circular Buffers ........................................................................................................ 2-17
2.6 Single-Instruction Repeat (RPT) Loops ..................................................................... 2-20
2.7 Subroutines .............................................................................................................. 2-24
2.8 Extended-Precision Arithmetic ................................................................................ 2-26
2.8.1 Addition .............................................................................................................. 2-26
2.8.2 Subtraction ......................................................................................................... 2-29
2.8.3 Multiplication ...................................................................................................... 2-30
2.8.4 Division .............................................................................................................. 2-35
2.9 Floating-Point Arithmetic ......................................................................................... 2-39
2.10 Application-Oriented Operations ............................................................................ 2-46
2.10.1 Modem Application .......................................................................................... 2-46
2.10.2 Adaptive Filtering ............................................................................................ 2-49
2.10.3 Infinite Impulse Response (IIR) Filters .............................................................. 2-52
2.10.4 Dynamic Programming .................................................................................... 2-55
2.11 Fast Fourier Transforms ......................................................................................... 2-57
3 External Memory Interface 3-1

Describes the external interface to program memory, local data memory, and I/O space. Also described is the direct memory access (DMA) in a portable computer configuration.

3.1 External Interface to Program Memory 3-2
3.2 External Interface to Local Data Memory 3-4
3.3 External Interface to Global Data Memory 3-6
3.4 External Interface to I/O Space 3-6
3.5 Direct Memory Access (DMA) in a Personal Computer Configuration 3-7

4 Analog Interface Peripherals and Applications 4-1

Describes the use of the TMS320 DSP in a variety of applications.

4.1 Multimedia Applications 4-2
4.1.1 System Design Considerations 4-2
4.1.2 Multimedia-Related Devices 4-4
4.2 Telecommunications Applications 4-5
4.2.1 System Design Considerations 4-6
4.2.2 Telecommunications-Related Devices 4-8
4.3 Dedicated Speech Synthesis Applications 4-10
4.3.1 System Design Considerations 4-10
4.3.2 Speech Synthesis-Related Devices 4-11
4.4 Servo Control/Disk Drive Applications 4-12
4.4.1 System Design Considerations 4-12
4.4.2 Servo Control/Disk Drive-Related Devices 4-14
4.5 Modem Applications 4-15
4.6 Advanced Digital Electronics Applications for Consumers 4-18
4.6.1 Advanced Television System Design Considerations 4-18
4.6.2 Advanced Digital Electronics-Related Devices 4-20

A Design Considerations for Using XDS510 Emulator A-1

Describes the JTAG emulator cable and how to construct a 14-pin connector on your target system and how to connect the target system to the emulator.

A.1 Cable Header and Signals A-2
A.2 Bus Protocol A-3
A.3 Emulator Cable Pod A-4
A.4 Emulator Cable Pod Signal Timings A-6
A.5 Target System Test Clock A-7
A.6 Configuring Multiple Processors A-8
A.7 Connections Between the Emulator and the Target System A-9
A.7.1 Emulation Signals Not Buffered A-9
A.7.2 Emulation Signals Buffered A-10
A.8 Emulation Timing Calculations A-11
B Development Support and Part Order Information ......................... B-1

Provides device part numbers and support tool ordering information for the TMS320C5x and development support information available from TI and third-party vendors.

B.1 Development Support ................................................. B-2
  B.1.1 Software and Hardware Development Tools .................. B-2
  B.1.2 Third-Party Support ............................................ B-2
  B.1.3 TMS320C5x DSP Design Workshop .......................... B-3
  B.1.4 Assistance ...................................................... B-3

B.2 Part Order Information ................................................ B-4
  B.2.1 Device and Development Support Tool Nomenclature .......... B-4
  B.2.2 Device Nomenclature ......................................... B-5
  B.2.3 Development Support Tools .................................. B-6

B.3 Hewlett-Packard E2442A Preprocessor 'C5x Interface ............. B-8
  B.3.1 Capabilities .................................................... B-8
  B.3.2 Logic Analyzers Supported .................................. B-8
  B.3.3 Pods Required ................................................ B-9
  B.3.4 Termination Adapters (TAs) ................................ B-9
  B.3.5 Availability ..................................................... B-9

C Application Reports and Designer's Notebook Pages .................. C-1

Lists the TMS320C5x application reports and the TMS320C5x designer's notebook pages (DNP) available to you.
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1–1</td>
<td>Evolution of the TMS320 Family</td>
</tr>
<tr>
<td>1–2</td>
<td>Typical Applications for the TMS320 Family</td>
</tr>
<tr>
<td>2–1</td>
<td>32-Bit Addition</td>
</tr>
<tr>
<td>2–2</td>
<td>32-Bit Subtraction</td>
</tr>
<tr>
<td>2–3</td>
<td>16-Bit Integer Multiplication Algorithm</td>
</tr>
<tr>
<td>2–4</td>
<td>32-Bit Multiplication Algorithm</td>
</tr>
<tr>
<td>2–5</td>
<td>16-Bit Integer Division</td>
</tr>
<tr>
<td>2–6</td>
<td>Nth-Order, Direct-Form, Type II, IIR Filter</td>
</tr>
<tr>
<td>2–7</td>
<td>Backtracking With Path History</td>
</tr>
<tr>
<td>2–8</td>
<td>An In-Place DIT FFT With In-Order Outputs and Bit-Reversed Inputs</td>
</tr>
<tr>
<td>2–9</td>
<td>An In-Place DIT FFT With In-Order Inputs but Bit-Reversed Outputs</td>
</tr>
<tr>
<td>3–1</td>
<td>'C5x Interfacing to External EPROM</td>
</tr>
<tr>
<td>3–2</td>
<td>'C5x Interfacing to External RAM</td>
</tr>
<tr>
<td>3–3</td>
<td>Global Memory Interface</td>
</tr>
<tr>
<td>3–4</td>
<td>Direct Memory Access in a PC Environment</td>
</tr>
<tr>
<td>4–1</td>
<td>System Block Diagram</td>
</tr>
<tr>
<td>4–2</td>
<td>Multimedia Speech Encoding and Modem Communication</td>
</tr>
<tr>
<td>4–3</td>
<td>TMS320C25 to TLC32047 Interface</td>
</tr>
<tr>
<td>4–4</td>
<td>Generic Telecom Application</td>
</tr>
<tr>
<td>4–5</td>
<td>General Telecom Applications</td>
</tr>
<tr>
<td>4–6</td>
<td>Typical DSP/Combo Interface</td>
</tr>
<tr>
<td>4–7</td>
<td>DSP/Combo Interface Timing</td>
</tr>
<tr>
<td>4–8</td>
<td>Generic Servo Control Loop</td>
</tr>
<tr>
<td>4–9</td>
<td>Disk Drive Control System Block Diagram</td>
</tr>
<tr>
<td>4–10</td>
<td>TMS320C14 to TLC32071 Interface</td>
</tr>
<tr>
<td>4–11</td>
<td>High-Speed V.32bis and Multistandard Modem With the TLC320AC01 AIC</td>
</tr>
<tr>
<td>4–12</td>
<td>Applications Performance Requirements</td>
</tr>
<tr>
<td>4–13</td>
<td>Video Signal Processing Basic System</td>
</tr>
<tr>
<td>4–14</td>
<td>Typical Digital Audio Implementation</td>
</tr>
<tr>
<td>A–1</td>
<td>Header Signals and Header Dimensions</td>
</tr>
<tr>
<td>A–2</td>
<td>Emulator Cable Pod Interface</td>
</tr>
<tr>
<td>A–3</td>
<td>Emulator Cable Pod Timings</td>
</tr>
<tr>
<td>A–4</td>
<td>Target-System Generated Test Clock</td>
</tr>
<tr>
<td>A–5</td>
<td>Multiprocessor Connections</td>
</tr>
<tr>
<td>A–6</td>
<td>Emulator Connections Without Signal Buffering</td>
</tr>
<tr>
<td>A–7</td>
<td>Buffered Signals</td>
</tr>
<tr>
<td>B–1</td>
<td>TMS320C5x Device Nomenclature</td>
</tr>
<tr>
<td>B–2</td>
<td>TMS320 Development Tool Nomenclature</td>
</tr>
</tbody>
</table>
Tables

1–1 Characteristics of the 'C5x DSPs ................................................. 1-6
2–1 Conditions for Branch, Call, and Return Instructions ......................... 2-12
2–2 Groups for Multiconditional Instructions ...................................... 2-13
2–3 Bit-Reversal Algorithm for an 8-Point Radix-2 DIT FFT ....................... 2-58
4–1 Data Converter ICs ........................................................................ 4-4
4–2 Switched-Capacitor Filter ICs ....................................................... 4-4
4–3 Telecom Devices—Codec/Filter ICs .............................................. 4-9
4–4 Telecom Devices—Transient Suppressor ICs .................................... 4-9
4–5 Voice Synthesizers ........................................................................ 4-10
4–6 Speech Memories ........................................................................... 4-11
4–7 Control Related Devices ............................................................... 4-14
4–8 Modem AFE Data Converters ........................................................ 4-15
4–9 Audio/Video Analog/Digital Interface Devices .................................... 4-20
A–1 XDS510 Header Signal Description ................................................ A-2
A–2 Emulator Cable Pod Timing Parameters ......................................... A-6
B–1 TMS320C5x Development Support Tools Part Numbers .................. B-7
C–1 TMS320C5x Application Reports .................................................. C-1
C–2 TMS320C5x Designer’s Notebook Pages ........................................ C-2
## Examples

2–1  Initialization of TMS320C5x ......................................................... 2-3  
2–2  Use of INTR Instruction .......................................................... 2-6  
2–3  Software Stack Operation ....................................................... 2-8  
2–4  Using PLU to Do Unpacking ...................................................... 2-10  
2–5  Using PLU to Do Packing .......................................................... 2-11  
2–6  Using Multiple Conditions With BCND Instruction ...................... 2-13  
2–7  Using CRGT and CRLT Instructions .......................................... 2-15  
2–8  Using Nested Loops ................................................................. 2-16  
2–9  Use of Circular Addressing ...................................................... 2-18  
2–10 Modulo-256 Addressing ........................................................... 2-19  
2–11 Memory-to-Memory Block Moves Using RPT with BLDD .............. 2-21  
2–12 Memory-to-Memory Block Moves Using RPT with BLDP .............. 2-21  
2–13 Memory-to-Memory Block Moves Using RPT with BLPD .............. 2-21  
2–14 Memory-to-Memory Block Moves Using RPT with TBLR .............. 2-22  
2–15 Memory-to-Memory Block Moves Using RPT with TBLW .............. 2-22  
2–16 Memory-to-Memory Block Moves Using RPT with SMMR .............. 2-22  
2–17 Memory-to-Memory Block Moves Using RPT with LMMR .............. 2-23  
2–18 Square Root Computation Using XC Instruction ......................... 2-24  
2–19 64-Bit Addition ................................................................. 2-28  
2–20 64-Bit Subtraction ................................................................. 2-30  
2–21 32-Bit Integer Multiplication .................................................... 2-33  
2–22 32-Bit Fractional Multiplication .............................................. 2-34  
2–23 16-Bit Integer Division Using SUBC Instruction ......................... 2-37  
2–24 16-Bit Fractional Division Using SUBC Instruction ..................... 2-38  
2–25 Floating-Point Addition Using SATL and SATH Instructions ........... 2-40  
2–26 Floating-Point Multiplication Using BSAR Instruction ................. 2-44  
2–27 V.32 Encoder Using Accumulator Buffer .................................... 2-47  
2–28 Adaptive FIR Filter Using RPT and RPTB Instructions .................. 2-50  
2–29 Nth-Order IIR Filter Using RPT and MACD Instructions ................. 2-53  
2–30 N Cascaded BiQuad IIR Filter Using LTD and MPYA Instructions ....... 2-54  
2–31 Backtracking Algorithm Using Circular Addressing ..................... 2-56  
2–32 16-Point Radix-2 Complex FFT ............................................. 2-59  
2–33 Bit-Reversed Addressing for an FFT ........................................ 2-62  
2–34 Macros for 16-Point DIT FFT .................................................. 2-63  
2–35 Initialization Routine ............................................................. 2-69
This user’s guide provides applications for the TMS320C5x generation of fixed-point digital signal processors (DSPs) in the TMS320 family. The 'C5x DSP provides improved performance over earlier 'C1x and 'C2x generations while maintaining upward compatibility of source code between the devices. The 'C5x central processing unit (CPU) is based on the 'C25 CPU and incorporates additional architectural enhancements that allow the device to run twice as fast as 'C2x devices. Future expansion and enhancements are expected to heighten the performance and range of applications of the 'C5x DSPs.

The 'C5x generation of static CMOS DSPs consists of the following devices:

<table>
<thead>
<tr>
<th>Device</th>
<th>On-Chip RAM</th>
<th>On-Chip ROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMS320C50/LC50</td>
<td>10K words</td>
<td>2K words</td>
</tr>
<tr>
<td>TMS320C51/LC51</td>
<td>2K words</td>
<td>8K words</td>
</tr>
<tr>
<td>TMS320C52/LC52</td>
<td>1K words</td>
<td>4K words</td>
</tr>
<tr>
<td>TMS320C53/LC53</td>
<td>4K words</td>
<td>16K words</td>
</tr>
<tr>
<td>TMS320C53S/LC53S</td>
<td>4K words</td>
<td>16K words</td>
</tr>
<tr>
<td>TMS320LC56</td>
<td>7K words</td>
<td>32K words</td>
</tr>
<tr>
<td>TMS320C57S</td>
<td>7K words</td>
<td>2K words</td>
</tr>
<tr>
<td>TMS320LC57</td>
<td>7K words</td>
<td>32K words</td>
</tr>
</tbody>
</table>

**Topic** | Page
--- | ---
1.1 TMS320 Family Overview | 1-2
1.2 TMS320C5x Overview | 1-5
1.3 TMS320C5x Key Features | 1-7
1.1 TMS320 Family Overview

The TMS320 family consists of two types of single-chip DSPs: 16-bit fixed-point and 32-bit floating-point. These DSPs possess the operational flexibility of high-speed controllers and the numerical capability of array processors. Combining these two qualities, the TMS320 processors are inexpensive alternatives to custom-fabricated very large scale integration (VLSI) and multichip bit-slice processors. Refer to subsection 1.1.2, TMS320 Typical Applications, for a detailed list of applications of the TMS320 family. The following characteristics make this family the ideal choice for a wide range of processing applications:

- Very flexible instruction set
- Inherent operational flexibility
- High-speed performance
- Innovative, parallel architectural design
- Cost-effectiveness

1.1.1 History, Development, and Advantages of TMS320 DSPs

In 1982, Texas Instruments introduced the TMS32010 — the first fixed-point DSP in the TMS320 family. Before the end of the year, Electronic Products magazine awarded the TMS32010 the title “Product of the Year”. The TMS32010 became the model for future TMS320 generations.

Today, the TMS320 family consists of these generations (Figure 1–1): ’C1x, ’C2x, ’C2xx, ’C5x, ’C54x, and ’C6x fixed-point DSPs; ’C3x and ’C4x floating-point DSPs; and ’C8x multiprocessor DSPs. Figure 1–1 illustrates the performance gains that the TMS320 family has made over time with successive generations. Source code is upwardly compatible from one fixed-point generation to the next fixed-point generation (except for the ’C54x), and from one floating-point generation to the next floating-point generation. Upward compatibility preserves the software generation of your investment, thereby providing a convenient and cost-efficient means to a higher-performance, more versatile DSP system.

Each generation of TMS320 devices has a CPU and a variety of on-chip memory and peripheral configurations for developing spin-off devices. These spin-off devices satisfy a wide range of needs in the worldwide electronics market. When memory and peripherals are integrated into one processor, the overall system cost is greatly reduced, and circuit board space is saved.
Figure 1–1. Evolution of the TMS320 Family
1.1.2 TMS320 Typical Applications

The TMS320 family of DSPs offers better, more adaptable approaches to traditional signal-processing problems, such as vocoding, filtering, and error coding. Furthermore, the TMS320 family supports complex applications that often require multiple operations to be performed simultaneously. Figure 1–2 shows many of the typical applications of the TMS320 family.

Figure 1–2. Typical Applications for the TMS320 Family

<table>
<thead>
<tr>
<th>Automotive</th>
<th>Consumer</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptive ride control</td>
<td>Digital radios/TVs</td>
<td>Disk drive control</td>
</tr>
<tr>
<td>Antiskid brakes</td>
<td>Educational toys</td>
<td>Engine control</td>
</tr>
<tr>
<td>Cellular telephones</td>
<td>Music synthesizers</td>
<td>Laser printer control</td>
</tr>
<tr>
<td>Digital radios</td>
<td>Power tools</td>
<td>Motor control</td>
</tr>
<tr>
<td>Engine control</td>
<td>Radar detectors</td>
<td>Robotics control</td>
</tr>
<tr>
<td>Global positioning</td>
<td>Solid-state answering machines</td>
<td>Servo control</td>
</tr>
<tr>
<td>Navigation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vibration analysis</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Voice commands</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>General-Purpose</th>
<th>Graphics/Imaging</th>
<th>Industrial</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptive filtering</td>
<td>3-D rotation</td>
<td>Numeric control</td>
</tr>
<tr>
<td>Convolution</td>
<td>Animation/digital map</td>
<td>Power-line monitoring</td>
</tr>
<tr>
<td>Correlation</td>
<td>Homomorphic processing</td>
<td>Robotics</td>
</tr>
<tr>
<td>Digital filtering</td>
<td>Pattern recognition</td>
<td>Security access</td>
</tr>
<tr>
<td>Fast Fourier transforms</td>
<td>Image enhancement</td>
<td></td>
</tr>
<tr>
<td>Hilbert transforms</td>
<td>Image compression/transmission</td>
<td></td>
</tr>
<tr>
<td>Waveform generation</td>
<td>Robot vision</td>
<td></td>
</tr>
<tr>
<td>Windowing</td>
<td>Workstations</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instrumentation</th>
<th>Medical</th>
<th>Military</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital filtering</td>
<td>Diagnostic equipment</td>
<td>Image processing</td>
</tr>
<tr>
<td>Function generation</td>
<td>Fetal monitoring</td>
<td>Missile guidance</td>
</tr>
<tr>
<td>Pattern matching</td>
<td>Hearing aids</td>
<td>Navigation</td>
</tr>
<tr>
<td>Phase-locked loops</td>
<td>Patient monitoring</td>
<td>Radar processing</td>
</tr>
<tr>
<td>Seismic processing</td>
<td>Prosthetics</td>
<td>Radio frequency modems</td>
</tr>
<tr>
<td>Spectrum analysis</td>
<td>Ultrasound equipment</td>
<td>Secure communications</td>
</tr>
<tr>
<td>Transient analysis</td>
<td></td>
<td>Sonar processing</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Telecommunications</th>
<th>Voice/Speech</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200- to 19200-bps modems</td>
<td>DTMF encoding/decoding</td>
</tr>
<tr>
<td>Adaptive equalizers</td>
<td>Echo cancellation</td>
</tr>
<tr>
<td>ADPCM transcoders</td>
<td>Fax</td>
</tr>
<tr>
<td>Cellular telephones</td>
<td>Line repeaters</td>
</tr>
<tr>
<td>Channel multiplexing</td>
<td>Speaker phones</td>
</tr>
<tr>
<td>Data encryption</td>
<td>Spread spectrum communications</td>
</tr>
<tr>
<td>Digital PBXs</td>
<td>Video conferencing</td>
</tr>
<tr>
<td>Digital speech interpolation (DSI)</td>
<td>X.25 Packet Switching</td>
</tr>
<tr>
<td>Personal digital assistants (PDA)</td>
<td>Personal communications systems (PCS)</td>
</tr>
</tbody>
</table>
1.2 TMS320C5x Overview

The 'C5x generation consists of the 'C50, 'C51, 'C52, 'C53, 'C53S, 'C56, 'C57, and 'C57S DSPs, which are fabricated by CMOS integrated-circuit technology. Their architectural design is based on the 'C25. The operational flexibility and speed of the 'C5x are the result of combining an advanced Harvard architecture (which has separate buses for program memory and data memory), a CPU with application-specific hardware logic, on-chip peripherals, on-chip memory, and a highly specialized instruction set. The 'C5x is designed to execute up to 50 million instructions per second (MIPS). Spin-off devices that combine the 'C5x CPU with customized on-chip memory and peripheral configurations may be developed for special applications in the worldwide electronics market.

The 'C5x devices offer these advantages:

- Enhanced TMS320 architectural design for increased performance and versatility
- Modular architectural design for fast development of spin-off devices
- Advanced integrated-circuit processing technology for increased performance and low power consumption
- Source code compatibility with 'C1x, 'C2x, and 'C2xx DSPs for fast and easy performance upgrades
- Enhanced instruction set for faster algorithms and for optimized high-level language operation
- Reduced power consumption and increased radiation hardness because of new static design techniques

Table 1–1 lists the major characteristics of the 'C5x DSPs. The table shows the capacity of on-chip RAM and ROM, number of serial and parallel input/output (I/O) ports, power supply requirements, execution time of one machine cycle, and package types available with total pin count. Use Table 1–1 for guidance in choosing the best 'C5x DSP for your application.
### Table 1–1. Characteristics of the 'C5x DSPs

<table>
<thead>
<tr>
<th>TMS320 Device</th>
<th>ID</th>
<th>On-Chip Memory (16-bit words)</th>
<th>I/O Ports</th>
<th>Power Supply (V)</th>
<th>Cycle Time (ns)</th>
<th>Package Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>'C50</td>
<td>PQ</td>
<td>1056 9K 2K§ 2$ 64K</td>
<td>5</td>
<td>50/35/25</td>
<td>132 pin BQFP$</td>
<td></td>
</tr>
<tr>
<td>'LC50</td>
<td>PQ</td>
<td>1056 9K 2K§ 2$ 64K</td>
<td>3.3</td>
<td>50/35/25</td>
<td>132 pin BQFP$</td>
<td></td>
</tr>
<tr>
<td>'C51</td>
<td>PZ</td>
<td>1056 1K 8K§ 2$ 64K</td>
<td>5</td>
<td>50/35/25/20</td>
<td>132 pin BQFP$</td>
<td></td>
</tr>
<tr>
<td>'LC51</td>
<td>PZ</td>
<td>1056 1K 8K§ 2$ 64K</td>
<td>3.3</td>
<td>50/35/25</td>
<td>132 pin BQFP$</td>
<td></td>
</tr>
<tr>
<td>'C52</td>
<td>PJ</td>
<td>1056 — 4K§ 1 64K</td>
<td>5</td>
<td>50/35/25/20</td>
<td>100 pin QFP$</td>
<td></td>
</tr>
<tr>
<td>'LC52</td>
<td>PJ</td>
<td>1056 — 4K§ 1 64K</td>
<td>3.3</td>
<td>50/35/25</td>
<td>100 pin QFP$</td>
<td></td>
</tr>
<tr>
<td>'C53</td>
<td>PQ</td>
<td>1056 3K 16K§ 2$ 64K</td>
<td>5</td>
<td>50/35/25</td>
<td>132 pin BQFP$</td>
<td></td>
</tr>
<tr>
<td>'LC53</td>
<td>PQ</td>
<td>1056 3K 16K§ 2$ 64K</td>
<td>3.3</td>
<td>50/35/25</td>
<td>132 pin BQFP$</td>
<td></td>
</tr>
<tr>
<td>'LC53S</td>
<td>PZ</td>
<td>1056 3K 16K§ 2$ 64K</td>
<td>3.3</td>
<td>50/35/25</td>
<td>100 pin TQFP*</td>
<td></td>
</tr>
<tr>
<td>'C56</td>
<td>PZ</td>
<td>1056 6K 32K 2$ 64K</td>
<td>3.3</td>
<td>50/35/25</td>
<td>100 pin TQFP*</td>
<td></td>
</tr>
<tr>
<td>'C57S</td>
<td>PGE</td>
<td>1056 6K 2K§ 2$ 64K</td>
<td>5</td>
<td>50/35/25</td>
<td>144 pin TQFPΔ</td>
<td></td>
</tr>
<tr>
<td>'LC57</td>
<td>PBK</td>
<td>1056 6K 32K 2$ 64K</td>
<td>3.3</td>
<td>50/35/25</td>
<td>128 pin TQFP*</td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**
- † Dual-access RAM (DARAM)
- ‡ Single-access RAM (SARAM)
- § ROM bootloader available
- $ Includes time-division multiplexed (TDM) serial port
- $ Includes buffered serial port (BSP)
- || Includes host port interface (HPI)
- $ 20 × 20 × 3.8 mm bumpered quad flat-pack (BQFP) package
- $ 14 × 14 × 1.4 mm thin quad flat-pack (TQFP) package
- $ 14 × 20 × 2.7 mm quad flat-pack (QFP) package
- $ 20 × 20 × 1.4 mm thin quad flat-pack (TQFP) package
- Sixteen of the 64K parallel I/O ports are memory mapped.
1.3 TMS320C5x Key Features

Key features of the 'C5x DSPs are listed below. Where a feature is exclusive to a particular device, the device’s name is enclosed within parentheses and noted after that feature.

- **Compatibility**: Source-code compatible with 'C1x, 'C2x, and 'C2xx devices

- **Speed**: 20-/25-/35-/50-ns single-cycle fixed-point instruction execution time (50/40/28.6/20 MIPS)

- **Power**
  - 3.3-V and 5-V static CMOS technology with two power-down modes
  - Power consumption control with IDLE1 and IDLE2 instructions for power-down modes

- **Memory**
  - 224K-word × 16-bit maximum addressable external memory space
    (64K-word program, 64K-word data, 64K-word I/O, and 32K-word global memory)
  - 1056-word × 16-bit dual-access on-chip data RAM
  - 9K-word × 16-bit single-access on-chip program/data RAM ('C50)
  - 2K-word × 16-bit single-access on-chip boot ROM ('C50, 'C57S)
  - 1K-word × 16-bit single-access on-chip program/data RAM ('C51)
  - 8K-word × 16-bit single-access on-chip program ROM ('C51)
  - 4K-word × 16-bit single-access on-chip program ROM ('C52)
  - 3K-word × 16-bit single-access on-chip program/data RAM ('C53, 'C53S)
  - 16K-word × 16-bit single-access on-chip program ROM ('C53, 'C53S)
  - 6K-word × 16-bit single-access on-chip program/data RAM ('LC56, 'C57S, 'LC57)
  - 32K-word × 16-bit single-access on-chip program ROM ('LC56, 'LC57)
Central processing unit (CPU)
- Central arithmetic logic unit (CALU) consisting of the following:
  - 32-bit arithmetic logic unit (ALU), 32-bit accumulator (ACC), and 32-bit accumulator buffer (ACCB)
  - 16-bit × 16-bit parallel multiplier with a 32-bit product capability
  - 0- to 16-bit left and right data barrel-shifters and a 64-bit incremental data shifter
- 16-bit parallel logic unit (PLU)
- Dedicated auxiliary register arithmetic unit (ARAU) for indirect addressing
- Eight auxiliary registers

Program control
- 8-level hardware stack
- 4-deep pipelined operation for delayed branch, call, and return instructions
- Eleven shadow registers for storing strategic CPU-controlled registers during an interrupt service routine (ISR)
- Extended hold operation for concurrent external direct memory access (DMA) of external memory or on-chip RAM
- Two indirectly addressed circular buffers for circular addressing

Instruction set
- Single-cycle multiply/accumulate instructions
- Single-instruction repeat and block repeat operations
- Block memory move instructions for better program and data management
- Memory-mapped register load and store instructions
- Conditional branch and call instructions
- Delayed execution of branch and call instructions
- Fast return from interrupt instructions
- Index-addressing mode
- Bit-reversed index-addressing mode for radix-2 fast Fourier transforms (FFTs)
TMS320C5x Key Features

- **On-chip peripherals**
  - 64K parallel I/O ports (16 I/O ports are memory mapped)
  - Sixteen software-programmable wait-state generators for program, data, and I/O memory spaces
  - Interval timer with period, control, and counter registers for software stop, start, and reset
  - Phase-locked loop (PLL) clock generator with internal oscillator or external clock source
  - Multiple PLL clocking option (x1, x2, x3, x4, x5, x9, depending on the device)
  - Full-duplex synchronous serial port interface for direct communication between the 'C5x and another serial device
  - Time-division multiplexed (TDM) serial port ('C50, 'C51, 'C53)
  - Buffered serial port (BSP) ('LC56, 'C57S, 'LC57)
  - 8-bit parallel host port interface (HPI) ('C57, 'C57S)

- **Test/emulation**
  - On-chip scan-based emulation logic
  - IEEE JTAG Standard 1149.1 boundary scan logic ('C50, 'C51, 'C53, 'C57S)

- **Packages**
  - 100-pin quad flat-pack (QFP) package ('C52)
  - 100-pin thin quad flat-pack (TQFP) package ('C51, 'C52, 'C53S, 'LC56)
  - 128-pin TQFP package ('LC57)
  - 132-pin bumpered quad flat-pack (BQFP) package ('C50, 'C51, 'C53)
  - 144-pin TQFP package ('C57S)
The 'C5x devices maintain source-code compatibility with 'C1x and 'C2x generations and have architectural enhancements that improve performance and versatility. An orthogonal instruction set is augmented by new instructions that support additional hardware and handle data movement and memory-mapped registers. Other features include an independent parallel logic unit (PLU) for performing Boolean operations, a 32-bit accumulator buffer (ACCB), and a set of registers that provide zero-latency context-switching capabilities to interrupt service routines. The on-chip dual-access RAM and memory-mapped register set are enhanced.

This chapter explains the use of the 'C5x instruction set with particular emphasis on its new features and special applications. For a complete discussion of the assembler directives used in this chapter’s examples, consult the TMS320C1x/C2x/C2xx/C5x Assembly Language Tools User’s Guide.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Processor Initialization</td>
<td>2-2</td>
</tr>
<tr>
<td>2.2 Interrupts</td>
<td>2-5</td>
</tr>
<tr>
<td>2.3 Software Stack</td>
<td>2-8</td>
</tr>
<tr>
<td>2.4 Logical and Arithmetic Operations</td>
<td>2-9</td>
</tr>
<tr>
<td>2.5 Circular Buffers</td>
<td>2-17</td>
</tr>
<tr>
<td>2.6 Single-Instruction Repeat (RPT) Loops</td>
<td>2-20</td>
</tr>
<tr>
<td>2.7 Subroutines</td>
<td>2-24</td>
</tr>
<tr>
<td>2.8 Extended-Precision Arithmetic</td>
<td>2-26</td>
</tr>
<tr>
<td>2.9 Floating-Point Arithmetic</td>
<td>2-39</td>
</tr>
<tr>
<td>2.10 Application-Oriented Operations</td>
<td>2-46</td>
</tr>
<tr>
<td>2.11 Fast Fourier Transforms</td>
<td>2-57</td>
</tr>
</tbody>
</table>
2.1 Processor Initialization

Before executing a 'C5x algorithm, it is necessary to initialize the processor. Generally, initialization takes place anytime the processor is reset. The processor is reset by applying a low level to RS input; the interrupt vector pointer (IPTR) bits of the processor mode status register (PMST) are all cleared, thus mapping the vectors to page 0 in program memory space. This means that the reset vector always resides at program memory location 0. This location normally contains a branch instruction to direct program execution to the system initialization routine. A hardware reset clears all pending interrupt flags and sets the interrupt mode (INTM) bit in ST0, thereby disabling all interrupts. A hardware reset also initializes various status bits and peripheral registers.

To configure the processor after the reset, the following internal functions must be initialized:

- Memory-mapped core processor and peripheral control registers
- Interrupt structure (INTM bit)
- Mode control (OVM, SXM, PM, AVIS, NDX, TRM bits)
- Memory control (RAM, OVL, CNF bits)
- Auxiliary registers and the auxiliary register pointer (ARP)
- Data memory page pointer (DP)

The OVM (overflow mode), TC (test/control flag), IMR (interrupt mask register), auxiliary register pointer (ARP), auxiliary register buffer (ARB), and data memory page pointer (DP) are not initialized by reset.

Example 2–1 shows coding for initializing the 'C5x to the following machine state and for the initialization performed after hardware reset:

- Internal single-access RAM configured as program memory
- Interrupt vector table loaded in internal program memory
- Interrupt vector table pointer (IPTR)
- Internal dual-access RAM blocks filled with 0s
- Interrupts enabled
Example 2–1. Initialization of TMS320C5x

```assembly
.title 'PROCESSOR INITIALIZATION'
.mmregs
.ref ISR0,ISR1,ISR2,ISR3,ISR4,TIME
.ref RCV,XMT,TRX,TXMT,TRP,NMISR
*;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
* Processor initialization for TMS320C50.
*
* The memory mapping of S/A RAM in program space and data space is different for
* the ’C5x devices. Therefore, the memory location pointed to by address 0800h
* in data space is mapped to a different address in program space for different
* ’C5x devices. Hence, the IPTR should be loaded with the corresponding value
* to allocate the vector table to the correct program space.
*
*       |   C50   |   C51   |   C53   |   C56   |   C57
*–––––––+–––––––––+–––––––––+–––––––––+–––––––––+––––––––
* PMST  |  0081E  |  0201E  |  0401E  |  0801E  |  0801E
*
*;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
V_TBL .sect "vectors"
RESET B INIT ;This section will be loaded in program
            ;memory address 0h.
INT0 B ISR0 ;INT0 – begins processing here
INT1 B ISR1 ;INT1 – begins processing here
INT2 B ISR2 ;INT2 – begins processing here
INT3 B ISR3 ;INT3 – begins processing here
TINT B TIME ;Timer interrupt processing
RINT B RCV ;Serial port receive interrupt
XINT B XMT ;Serial port transmit interrupt
TRNT B TRX ;TDM port receive interrupt
TXNT B TXMT ;TDM port transmit interrupt
INT4 B ISR4 ;INT4 – begins processing here
            .space 14*16 ;14 words
TRAP B TRP
NMI B NMISR
.text
INIT LDP #0 ;Initialize data pointer
OPL #20h,PMST ;Configure S/A RAM in data memory
LAR AR7,#0800h ;Data space address for vector table
MAR *,AR7
RPT #39
BLPD #V_TBL,** ;Load vector table at 0800h
SPLK #0081h,PMST ;Now configure S/A RAM in program space
            ;and initialize vector table pointer
SPLK #01FFh,IMR ;Clear interrupt mask register
CLRC OVM ;Disable overflow saturation mode
LAR AR7,#60h ;Initialize B2 block
RPTZ #31
SACL *+
```
Example 2–1.  *Initialization of TMS320C5x (Continued)*

```
LAR  AR7,#100h ;Initialize B0 block
RPTZ #511
SACL *+
LAR  AR7,#300h ;Initialize B1 block
RPTZ #511
SACL *+
CLRC INTM ;Globally enable interrupts
B    MAIN_PRG ;Return
```
2.2 Interrupts

The 'C5x devices have four external, maskable, user interrupts (INT1–INT4) and one nonmaskable interrupt (NMI) available for external devices. Internal interrupts are generated by the serial ports, the timer, and by the software interrupt instructions (INTR, TRAP, and NMI). The interrupt structure is described in the TMS320C5x User’s Guide.

The 'C5x devices are capable of generating software interrupts using the INTR instruction. This allows any of the 32 interrupt service routines (ISRs) to be executed from your software. The first 20 ISRs are reserved for external interrupts, peripheral interrupts, and future implementations. The remaining 12 locations in the interrupt vector table are user-definable. The INTR instruction can invoke any of the 32 interrupts available on the 'C5x devices.

When an interrupt is executed, certain key CPU registers are saved automatically. The PC is saved on an 8-deep hardware stack, which is also used for subroutine calls. Therefore, the CPU supports subroutine calls within an ISR as long as the 8-level stack is not exceeded. Also, there is a 1-deep stack (or shadow register) for each of the following registers:

- Accumulator (ACC)
- Accumulator buffer (ACCB)
- Auxiliary register compare register (ARCR)
- Index register (INDX)
- Processor mode status register (PMST)
- Product register (PREG)
- Status register 0 (ST0)
- Status register 1 (ST1)
- Temporary register 0 (TREG0) for multiplier
- Temporary register 1 (TREG1) for shift count
- Temporary register 2 (TREG2) for bit test

When the interrupt trap is taken, the contents of all these registers are pushed onto a 1-level stack, with the exception of the the INTM bit in ST0 and the XF bit in ST1. On an interrupt, the INTM bit is always set to disable interrupts. The values in the registers at the time of the interrupt trap are still available to the ISR but are also protected in the shadow registers. The shadow registers are copied back to the CPU registers when the RETI or RETE instruction is executed. This function allows the CPU to be used for the ISR without requiring the overhead of context save and restore in the ISR.
Interrupts

Example 2–2 illustrates the use of the INTR instruction. The foreground program sets up auxiliary registers and invokes user-defined interrupt number 20. Since the context is saved automatically, the ISR is free to use any of the saved registers without destroying the calling program’s variables. The routine shown here uses the CRGT instruction to find the maximum value of 16 executions of the equation \( Y = aX^2 + bX + c \). AR1 points to the X values, AR2 points to the coefficients, and AR3 points to the Y results. To return the result to the calling routine, all the registers are restored by executing an RETI instruction. The computed value is placed in the accumulator, and a standard return is executed because the stack is already popped.

Example 2–2. Use of INTR Instruction

* Foreground Program

.mnregs
TEMP .set 63h ;Temporary storage.
X .set 64h
Y .set 65h
COEFF .set 66h
V_TBL .sect "vectors"
RESET B INIT ;This section will be loaded in program
;memory address 0h.
.space 38*16 ;Skip the next 38 locations to interrupt #20
IN20 B ISR20 :Interrupt #20 – begins processing here
.text
INIT LDP #0 ;Initialize data pointer
LAR AR1,#X ;AR1 points to X values
LAR AR2,#COEFF ;AR2 points to coefficients b,a,c in that order
LAR AR3,#Y ;AR3 points to Y results
INTR 20 ;Invoke software interrupt #20
B $ ;Finish the program

* This routine uses the block repeat feature of the ’C5x to find the maximum
* value of 16 executions of the equation \( Y = aX^2 + bX + c \). The X values are pointed
* at by AR1. The Y results are pointed at by AR3. The coefficients are pointed
* at by AR2. At the completion of the routine, ACC contains the maximum value.
* AR1, AR2, and AR3 are modified. All other registers are unaffected.
*
ISR20 LDP #0 ;Use page 0 of data memory.
LACC #08000h ;Initialize AccB with min. possible value
MAR *,AR1 ;ARP <- AR1
* Load Block repeat count register with 15.
SPLK #0Fh,BRCR
*
Example 2–2.  Use of INTR Instruction (Continued)

* Repeat Block.
  RPTB  END_LOOP-1 ;For i=0; i<=15; i++.
  ZAP ;ACC = PREG = 0
  SQR A *+,AR2 ;TREG0 = X  PREG = X^2
  SPL TEMP ;Save X^2.
  MPY *+ ;PREG = b*X
  LTA TEMP ;TREG = X^2  ACC = b*X
  MPY *+ ;PREG = a*X^2
  APAC ;ACC = a*X^2 + b*X
  ADD *+,0,AR3 ;ACC = A*X^2 + b*X + c
  SA CL *+,0,AR1 ;Save Y.
  CR GT ;Save maximum Y.
END LOOP
SA CL TEMP ;Save the result temporarily
L ACC #RE_ENTER
P US H ;Push re-entry address onto stack
R ETI ;Pop all registers
RE_ENTER
L ACC TEMP ;Load ACC with the max. value
R ET ;Return to interrupted code
2.3 Software Stack

The 'C5x has an internal 8-deep hardware stack that is used to save and restore return addresses for the subroutines and the ISRs. Provisions have been made on the 'C5x to extend the hardware stack into the data memory. The PUSH and POP instructions can access the hardware stack via the accumulator. Two additional instructions, PSHD and POPD, are included in the instruction set so that the stack may be directly stored to and recovered from the data memory.

A software stack can be implemented by using the POPD instruction at the beginning of each subroutine to save the PC in data memory. Then, before returning, a PSHD is used to put the proper value back onto the top of the stack.

When the stack has seven values stored on it, and two or more values are to be put on the stack before any other values are popped off, a subroutine that expands the stack is needed. A routine to expand the stack is shown in Example 2–3. In this example, the main program stores the stack, stores the starting memory location in AR2, and indicates to the subroutine whether to push the data from memory onto the stack or pop data from the stack to memory. If a 0 is loaded into the accumulator before calling the subroutine, the subroutine pushes data from memory to the stack. If the accumulator contains a nonzero value, the subroutine pops data from the stack to memory.

Because the CALL instruction uses the stack to save the program counter, the subroutine pops this value into the accumulator and uses the BACC instruction to return to the main program. This prevents the program counter from being stored into a memory location. The subroutine in Example 2–3 uses the BCND (delayed conditional branch) instruction to determine whether a save or restore operation is to be performed.

Example 2–3. Software Stack Operation

```
* This routine expands the stack while letting the main program determine where
* to store the stack contents, or from where to restore them. Entry Conditions:
* ACC = 0 (restore stack); 1 (save stack)
* AR2 -> Top of software stack in data memory
*
STACK: BCND DR, #0 ;Delayed branch if POPD required
       POP, #0 ;Use AR2 as stack pointer
       POP ;Get return address
       RPT #6 ;Repeat 7 times
       PSHD #0 ;Put memory in stack
       BACC ;Return to main program

POP: MAR #0 ;Align AR2
     RPT #6 ;Repeat 7 times
     POPD #0 ;Put stack in memory
     MAR #0 ;Realign stack pointer
     BACC ;Return to main program
```
2.4 Logical and Arithmetic Operations

The following subsections provide examples of logical and arithmetic operations.

2.4.1 Parallel Logic Unit (PLU)

The PLU provides a direct logical path to data memory values without affecting the contents of the accumulator or product register. The PLU allows direct manipulation of bits in any location in data memory space. The source operand can be either a long immediate value or the dynamic bit manipulation register (DBMR). The use of a long immediate value is particularly effective in initializing data memory locations, including the memory-mapped registers. The use of the DBMR as the source operand allows run-time computation of operands. It also reduces instruction execution time to one cycle, which may be important for time-critical routines.

Example 2–4 on page 2-10 and Example 2–5 on page 2-11 illustrate the use of the PLU for initialization and logical operation. The UNPACK subroutine (Example 2–4) extracts individual bits from a single word and stores them separately in an array. The PACK subroutine (Example 2–5) does the opposite of UNPACK by getting bits from different locations and packing them in a single word. In Example 2–5, notice that a NOP instruction is inserted in the repeat-block loop. A repeat-block loop must be at least three words long on 'C5x devices.
Example 2–4. Using PLU to Do Unpacking

.title 'Routine to extract bits from a single word'

* PCKD
* |Bn ---- B0|

* UNPCKD
* |0 -- 0|Bn|
* |0 -- 0|Bn-1|
* ... |
* |0 -- 0|B0|

.mmregs
NO_BITS .set 16 ;Number of packed bits in the word
PCKD .set 60h ;Input word
UNPCKD .set 61h ;Output buffer. Each word will have
; one bit in LSB location.

.text
UNPACK LDP #0 ;DP=0
MAR *,AR0
LAR AR0,#UNPCKD+NO_BITS-1;End of table address
SPLK #NO_BITS-1,BRCR ;Initialize the count register
SPLK #1,DBMR ;Load mask in DBMR register
LACC PCKD ;Packed bits -> Acc
RPTB LOOP–1 ;Begin looping
SACL * ;Save remaining packed bits
APL *- ;Keep the LSB only
SFR ;Shift right to eliminate unpacked bit
LOOP RET ;Return back
Example 2–5. Using PLU to Do Packing

*.title ’Routine to pack input bits in a single word’
.*
.* PCKD
.* ---------------------
.* |Bn  -----  B0|
.* ---------------------
.*
.* UNPCKD
.* ---------------------
.* |0  --  0 |Bn|
.* ---------------------
.* |0  --  0|Bn–1|
.* ---------------------
.* . .
.* ---------------------
.* |0  --  0|B0|
.* ---------------------
.*
.* .data
NO_BITS .set 16 ;Number of bits to be packed
PCKD .set 60h ;Packed word
UNPCKD .set 61h ;Array of unpacked bits
.* .text
PACK  LAR  AR0,#UNPCKD ;AR0 points to start of UNPACKED array
  MAR  *+,AR0
  LDP  #0 ;DP=0
  SPLK  #NO_BITS–2,BRCR ;Loop NO_BITS–1 times
  LACC  *+ ;Get the MSB
  RPTB LOOP–1 ;Begin looping
  SFL ;Make space for next bit
  ADD  *+ ;Put next bit
  NOP
  LOOP
  SACL  PCKD ;Store the result
  RET ;Return back
Logical and Arithmetic Operations

2.4.2 Multiconditional Instructions

The 'C5x includes instructions that test multiple conditions before passing control to another section of the program. These instructions are: BCND, BCNDD, CC, CCD, RETC, RETCD, and XC. These instructions can test the conditions listed in Table 2–1 individually or in combination with other conditions.

Table 2–1. Conditions for Branch, Call, and Return Instructions

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQ</td>
<td>ACC = 0</td>
<td>Accumulator equal to 0</td>
</tr>
<tr>
<td>NEQ</td>
<td>ACC ≠ 0</td>
<td>Accumulator not equal to 0</td>
</tr>
<tr>
<td>LT</td>
<td>ACC &lt; 0</td>
<td>Accumulator less than 0</td>
</tr>
<tr>
<td>LEQ</td>
<td>ACC ≤ 0</td>
<td>Accumulator less than or equal to 0</td>
</tr>
<tr>
<td>GT</td>
<td>ACC &gt; 0</td>
<td>Accumulator greater than 0</td>
</tr>
<tr>
<td>GEQ</td>
<td>ACC ≥ 0</td>
<td>Accumulator greater than or equal to 0</td>
</tr>
<tr>
<td>NC</td>
<td>C = 0</td>
<td>Carry bit cleared</td>
</tr>
<tr>
<td>C</td>
<td>C = 1</td>
<td>Carry bit set</td>
</tr>
<tr>
<td>NOV</td>
<td>OV = 0</td>
<td>No accumulator overflow detected</td>
</tr>
<tr>
<td>OV</td>
<td>OV = 1</td>
<td>Accumulator overflow detected</td>
</tr>
<tr>
<td>BIO</td>
<td>BIO is low</td>
<td>BIO signal is low</td>
</tr>
<tr>
<td>NTC</td>
<td>TC = 0</td>
<td>Test/control flag cleared</td>
</tr>
<tr>
<td>TC</td>
<td>TC = 1</td>
<td>Test/control flag set</td>
</tr>
<tr>
<td>UNC</td>
<td>none</td>
<td>Unconditional operation</td>
</tr>
</tbody>
</table>

You can combine conditions from four groups (Table 2–2). Up to four conditions can be selected; however, each of these conditions must be from different groups. You cannot have two conditions from the same group. For example, you can test EQ and TC at the same time but not NEQ and GEQ. For example:

```plaintext
BCND BRANCH,LT,NOV,TC ; If ACC < 0, no overflow
; and TC bit set.
```

In this example, LT (ACC < 0), NOV (OV = 0), and TC (TC = 1) conditions must be met for the branch to be taken.
Table 2–2. Groups for Multiconditional Instructions

<table>
<thead>
<tr>
<th>Group 1</th>
<th>Group 2</th>
<th>Group 3</th>
<th>Group 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQ</td>
<td>OV</td>
<td>C</td>
<td>TC</td>
</tr>
<tr>
<td>NEQ</td>
<td>NOV</td>
<td>NC</td>
<td>NTC</td>
</tr>
<tr>
<td>GT</td>
<td></td>
<td></td>
<td>BIO</td>
</tr>
<tr>
<td>LT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GEQ</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEQ</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Testing the status of the TC flag is mutually exclusive to testing the BIO pin. The code in Example 2–6 simultaneously tests the carry (C) flag and the sign bit of the accumulator to locate a zero bit (beginning from MSB) in a 64-bit word, consisting of ACC and ACCB with ACC having the higher part. This 64-bit word could be the serial port output where the first 0 indicates the start bit.

**Example 2–6. Using Multiple Conditions With BCND Instruction**

```
LDP    #0
SPLK   #63,BRCR ;No. of iterations - 1
.
.
;Code to get 64-bit input word and
;load it in ACC and ACCB
.
LAR    AR0,#0 ;Initialize the bit counter
RPTB   ENDLOOP–1 ;For I=0,I<=63,I++
SFLB
*      ;Shift left ACC+ACCB, MSB is shifted
    ;out in Carry flag
    ;Increment bit counter
MAR    *+ ;Exit if carry=0 and current MSB=1
BCND   ENDLOOP,NC,LT
ENDLOOP:
    ;ACC+ACCB contains aligned data now
APL    #0FFFEh,PMST ;Clear BRAF flag
```
2.4.3 Search Algorithm Using CRGT

Example 2–7 on page 2-15 shows how the CRGT and RPTB instructions find the maximum value and its location by searching through a block of data. Loop overhead is minimized by using the block-repeat function. The accumulator is initialized with the minimum possible value (8000h) before the main search loop is entered.

To find the minimum value, CRGT instruction may be replaced by CRLT, and the accumulator is loaded with the maximum possible value (7FFFh) instead of the smallest. The rest of the code remains the same.

2.4.4 Matrix Multiplication Using Nested Loops

The 'C5x provides three different types of instructions to implement code loops. The RPT (single-instruction repeat) instruction allows the following instruction to be executed N times. The RPTB (repeat block) instruction repeatedly executes a block of instructions with the loop count determined by the block repeat counter register (BRCR). The BANZ (branch if AR not 0) instruction is another way of implementing for-next loops with the count specified by an auxiliary register.

Three-level-deep nested loops can be efficiently implemented by these three instructions with each instruction controlling one loop. Example 2–8 on page 2-16 shows this nested code structure to do N-by-N matrix multiplication. Note the use of the BANZD (delayed BANZ) instruction to avoid flushing the instruction pipeline. Also, note the use of the MADS (multiply-accumulate using BMAR) instruction to dynamically switch between the rows of matrix A to compute the elements of the product matrix C.
Example 2–7. Using CRGT and CRLT Instructions

* This routine searches through a block of data in the data memory to store
* the maximum value and the address of that value in memory locations MAXVAL
* and MAXADR, respectively. The data block could be of any size defined by
* the Block Repeat Counter Register (BRCR).
*  
* KEY ’C5X instructions:
*  
* RPTB Repeat a block of code as defined by repeat counter BRCR.
* CRGT Compare ACC to ACCB. Store larger value in both ACC and ACCB,
* set CARRY bit if value larger than previously larger value found.
* XC Execute conditionally (1 or 2 words) if CARRY bit is set.
*  
MAXADR .set 60h
MAXVAL .set 61h

.mmregs
.text
LDP #0 ;Point to data page 0
LAR AR0,#0300h ;AR= data memory addr
SETC SXM ;Set sign extension mode
LACC #08000h ;Load minimum value

* Use #07FFFh (largest possible) to check for minimum value
SACB ;Store into ACCB
SPLK #9,BRCR ;Rpt cont = 9 for 10 data values
RPTB endb –1 ;Repeat block from here to endb–1
startb:
LACC * ;Load data from <(AR0)> into ACC
CRGT ;Set carry if ACC > previous largest value

* Use CRLT to find minimum value
SACL MAXVAL ;Save new largest which is in ACC & ACCB
XC #1,C ;Save addr if current value > previous largest
SAR AR0,MMAXADR
MAR *+  
endb: RET

* At the end of routine, following registers contain:
  * ACC = 32050
  * ACCB = 32050
  * (MAXVAL) = 32050
  * (MAXADR) = 0307h

.data ;Data is expected to be in data RAM
.word 5000 ;Start address = 0300h
.word 10000
.word 320
.word 3200
.word -5600
.word -2105
.word 2100
.word 32050
.word 1000
.word -1
.end
Example 2–8. Using Nested Loops

.title "NxN Matrix Multiply Routine"
.mmregs

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
* This routine performs multiplication of two NxN matrices.
* A x B = C where A, B, and C are NxN in size.
* Entry Conditions:
  * AR1 -> element (0,0) of A (in program space)
  * AR2 -> element (0,0) of B (in data space)
  * AR3 -> element (0,0) of C (in data space)
  * DP = 0, NDX = 1
  * ARP = 2
* Storage of matrix elements in memory (beginning from low memory):
  * M(0,0),...,M(0,N-1),M(1,0),...,M(N-1,N-1)
*;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;:

LDP #0
SPLK #3Eh,PMST
SPLK #2000h,AR1
SPLK #0810h,AR2
SPLK #0820h,AR3
MAR *,AR2

MTRX_MPY:
LAR AR0,#(N–1) ;Set up loop count
SPLK #N,INDX ;Row size
SAR AR2,AR4 ;Save addr of B

* ;For i=0,i<N,++i
LOOP1: SMMR AR1,BMAR ;BMAR -> A(i,0)
SPLK #(N–1),BRCR ;Setup loop2 count
SAR AR4,AR5 ;AR5 -> B(0,0)
LOOP2: RPTB ELOOP2 ;For j=0,j<N,++j
SAR AR5,AR2 ;AR2 -> B(0,j)
ELOOP3: MADS *0+ ;Acc=A(i,k)xB(k,j)
APAC ;Final accumulation
MAR *,AR5 ;ARp = AR5
MAR ++,AR3 ;AR5 -> B(0,j+1)
ELOOP2: SACL ++,AR2 ;Save C(i,j)
MAR *,AR0 ;Loop back if
BANZD LOOP1,*,AR1 ;Count != N
ADRK N ;AR1 -> A(i+1,0)
ELOOP1: MAR *,AR2 ;ARp = AR2
2.5 Circular Buffers

Circular addressing is an important feature of the 'C5x instruction set. Algorithms like convolution, correlation, and finite impulse response (FIR) filters can make use of circular buffers in memory. The 'C5x supports two concurrent buffers operating via the auxiliary registers. Five memory-mapped registers control the circular buffer operation: CBSR1, CBSR2, CBER1, CBER2, and CBCR.

The start and end addresses must be loaded in the corresponding buffer registers (CBSRx and CBERx) before the circular buffer is enabled. Also, the auxiliary register that acts as a pointer to the buffer must be initialized with the proper value.

Example 2–9 on page 2-18 shows the use of a circular buffer to generate a digital sine wave. A 256-word sine-wave table is loaded in the DARAM B1 block of internal data memory from external program memory. Accessing the internal DARAM requires only one machine cycle. The block move address register (BMAR) is loaded with the ROM address of the table. The block-move instruction moves 256 samples of the sine wave to internal data memory, which is then set up as a circular buffer.

The start and end addresses of this circular buffer are loaded into the corresponding registers (CBSR1 and CBER1). The auxiliary register AR7 is also initialized to the beginning of the sine-wave table. Note the use of the SAMM instruction to update AR7 because all auxiliary registers are memory-mapped at data page 0. Finally, circular buffer #1 is enabled and AR7 is mapped to that buffer. The other circular buffer is disabled.

Whenever the next sample is to be pulled off from the table, postincrement indirect addressing may be used with AR7 as the pointer. This ensures that the pointer wraps around to the beginning of the table if the previous sample was the last one on the table.
Example 2–9. Use of Circular Addressing

.title 'Digital Sine-Wave Generator'
.mmregs

* This routine illustrates the circular addressing capability of C5x devices.
* A digital sine-wave generator is implemented as circular buffer #1 with AR7
* as its pointer. XSINTBL is the location in external program memory where this
* table is stored. It is moved to internal data memory block B1 where it is
* setup as a circular buffer.
*
.XSINTBL .set 03000h ;Program space address of sine table

.text
SINTBL  LDP #0
   LAR  AR0,#0300h ;Address of B1 block
   MAR  *,AR0
   LACC #XSINTBL ;Get sine table address in
   *;external program memory
   SAMM BMAR ;Load source register
   *
   RPT  #255 ;Move 256-word
   BLPD BMAR,/*+ ;Load table from external program
   *;memory to internal data memory
   SAMM CBSR1 ;Start address of buffer=300h
   SAMM AR7 ;AR7 points to start of buffer
   ADD  #255
   SAMM CBER1 ;End address of buffer=3FFh
   SPLK #0Fh,CBCR ;Enable CB#1, disable CB#2
   *. ;pointer for CB#1 is AR7
   *
   NXTSMP MAR  *,AR7
   LACC  **+ ;Get next sample from table
   *. ;AR7 is updated to next valid sample
   *
   DISABLE APL  #0FFF7h,CBCR ;Disable CB#1
   *. 
   RET
Circular Buffers

If the step size must be greater than 1, check to see if an update to the auxiliary register generates an address outside the range of the circular buffer. This may happen if the same sine table is used to generate sine waves of different frequencies by changing the step size. Modulo addressing can avoid such problems. A simple way to perform modulo addressing on 'C5x devices is to use the APL and OPL instructions. For example, to implement the modulo-256 counter, first load the dynamic bit manipulation register (DBMR) with 255 (the maximum value allowed); when the auxiliary register is updated (by any amount), it is ANDed with the DBMR and ORed with the start address of the buffer. The start address of the modulo-2^k buffer must have 0s in the k LSBs. Hence, for modulo-256 addressing, the first eight LSBs of the start register must be 0 (see Example 2–10).

Example 2–10. Modulo-256 Addressing

```
START .set 04000h ;Start address of the buffer
LDP #0
LACL #0FFh
SAMM DBMR ;Max value = 255
.
.
MAR *0+ ;Increment AR7 by some amount
APL AR7 ;Extract lower 8 bits
OPL #START,AR7 ;Add the start address
.
.
```

Software Applications 2-19
2.6 Single-Instruction Repeat (RPT) Loops

The 'C5x provides two different types of repeat instructions. The repeat block (RPTB) instruction implements code loops that can be 3 to 65,536 words in size. These loops do not require any additional cycles to jump from the end-of-block to the start-of-block address at the end of each iteration. In addition, these zero-overhead loops are interruptible so that they can be used in background processing without affecting the latency of time-critical tasks.

On the other hand, the single-instruction repeat (RPT) pipelines the execution of the next instruction to provide a high-speed repeat mode. A 16-bit repeat counter register (RPTC) allows execution of a single instruction 65,536 times. When this repeat feature is used, the instruction being repeated is fetched only once. As a result, many multicycle instructions, such as MAC/MACD, BLDD/BLDP, or TBLR/TBLW, become single-cycle when repeated.

Some of 'C5x instructions behave differently in the single-instruction repeat mode to efficiently use the 'C5x multiple-bus architecture. The following instructions fall in this category:

BLDD, BLDP, BLPD, IN, OUT, MAC, MACD, MADS, MADD, TBLR, TBLW, LMMR, SMMR

Because the instruction is fetched and internally latched when in single-instruction repeat mode, the program bus is used by these instructions to read or write a second operand in parallel to the operations being done using the data bus. With the instruction latched for repeated execution, the program counter is loaded with the second operand address (which may be in data, program, or I/O space) and incremented on succeeding executions to read/write in successive memory locations. As an example, the MAC instruction fetches the multiplicand from the program memory via the program bus. Simultaneously with the program bus fetch, the second multiplicand is fetched from data memory via the data bus. In addition to these data fetches, preparation is made for accesses in the following cycle by incrementing the program counter and by indexing the auxiliary register. The IN instruction is another example of an instruction that benefits from simultaneous transfers of data on both the program and data buses. In this case, data values from successive locations in I/O space may be read and transferred to data memory. For complete details of how the above-listed instructions behave in repeat mode, see the individual description of each instruction in the TMS320C5x User’s Guide.

Example 2–11 through Example 2–17 demonstrate the implementation of memory-to-memory block moves on the 'C5x using single-instruction repeat (RPT) loops. There is no single instruction to move data from memory to memory.
Example 2–11. Memory-to-Memory Block Moves Using RPT with BLDD

* This routine uses the BLDD instruction to move external data memory to internal data memory.

MOVEDD:
LACC #4000h
Samm BMAR ;BMAR -> source in data memory.
LAR AR7,#100h ;AR7 -> destination in data memory
MAR *,AR7 ;LARP = AR7.
RPT #1023 ;Move 1024 value to blocks B0 and B1
BLDD BMAR,++
RET

Example 2–12. Memory-to-Memory Block Moves Using RPT with BLDP

* This routine uses the BLDP instruction to move external data memory to internal program memory. This instruction could be used to boot load a program to the 8K on-chip program memory from external data memory.

MOVEDP:
LACC #800h
Samm BMAR ;BMAR -> destination in program memory ('C50)
LAR AR7,#0E000h ;AR7 -> source in data memory.
RPT #8191 ;Move 8K to program memory space.
BLDP *++
RET

Example 2–13. Memory-to-Memory Block Moves Using RPT with BLPD

* This routine uses the BLPD instruction to move external program memory to internal data memory. This routine is useful for loading a coefficient table stored in external program memory to data memory when no external data memory is available.

MOVEPD:
LAR AR7,#100h ;AR7 -> destination in data memory.
RPT #127 ;Move 128 values from external program
BLPD #3800h,++ ;to internal data memory B0.
RET
Example 2–14. Memory-to-Memory Block Moves Using RPT with TBLR

* This routine uses the TBLR instruction to move external program memory to internal data memory. This differs from the BLPD instruction in that the accumulator contains the source program memory address from which to transfer. This allows for a calculated, rather than predetermined, location in program memory to be specified. The calling routine must contain the source program memory address in the accumulator.

TABLER:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAR*,AR3</td>
<td>AR3 -&gt; destination in data memory.</td>
</tr>
<tr>
<td>LAR AR3,#300h</td>
<td></td>
</tr>
<tr>
<td>RPT #127</td>
<td>Move 128 items to data memory block B1</td>
</tr>
<tr>
<td>TBLR +</td>
<td></td>
</tr>
<tr>
<td>RET</td>
<td></td>
</tr>
</tbody>
</table>

Example 2–15. Memory-to-Memory Block Moves Using RPT with TBLW

* This routine uses the TBLW instruction to move data memory to program memory. This differs from the BLDP instruction in that the accumulator contains the destination program memory address to which to transfer. This allows for a calculated, rather than predetermined, location in program memory to be specified. The calling routine must contain the destination program memory address in the accumulator.

TABLEW:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAR*,AR4</td>
<td>ARP = AR4.</td>
</tr>
<tr>
<td>LAR AR4,#380h</td>
<td>AR4 -&gt; source address in data memory.</td>
</tr>
<tr>
<td>RPT #127</td>
<td>Move 128 items from data memory to program memory.</td>
</tr>
<tr>
<td>TBLW +</td>
<td></td>
</tr>
<tr>
<td>RET</td>
<td></td>
</tr>
</tbody>
</table>

Example 2–16. Memory-to-Memory Block Moves Using RPT with SMMR

* This routine uses the SMMR instruction to move data from a memory-mapped I/O port to local data memory. Note that 16 I/O ports are mapped in data page 0 of the 'C5x memory map.

INPUT:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDP #0</td>
<td></td>
</tr>
<tr>
<td>RPT #511</td>
<td>Input 512 values from port 51h to table beginning</td>
</tr>
<tr>
<td>SMMR 51h,800h</td>
<td>at 800h in data memory.</td>
</tr>
<tr>
<td>RET</td>
<td></td>
</tr>
</tbody>
</table>
Example 2–17. Memory-to-Memory Block Moves Using RPT with LMMR

* This routine uses the LMMR instruction to move data from local data memory
to a memory-mapped I/O port. Note that 16 I/O ports are mapped in data
page 0 of the 'C5x memory map.

OUTPUT:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDP #0</td>
<td>data page 0</td>
</tr>
<tr>
<td>RPT #63</td>
<td>Output 64 values from a table beginning at 800h</td>
</tr>
<tr>
<td>LMMR 50h,800h</td>
<td>in data memory to port 50h.</td>
</tr>
</tbody>
</table>

RET
2.7 Subroutines

Example 2–18 shows the use of a subroutine to determine the square root of a 16-bit number. The main routine executes to the point where the square root of a number is needed. At this point, a delayed call (CALLD) is made to the subroutine, transferring control to that section of the program memory for execution and then returning to the calling routine via the delayed return (RETD) instruction when execution has completed.

Example 2–18 shows several features of the 'C5x instruction set. In particular, note the use of the delayed call (CALLD), delayed return (RETD), and conditional execute (XC) instructions. Due to the four-level-deep pipeline on 'C5x devices, normal branch instructions require four cycles to execute. Using delayed branches, only two cycles are required for execution. The XC instruction is useful where only one or two instructions are to be executed conditionally. In this example, notice how XC is used to avoid an extra cycle due to the branch instruction. Use of the XC instruction also helps in keeping the execution time of a routine constant, regardless of input conditions. This is because XC executes NOPs in place of instructions if conditions are not met.

Note that the restore is done with the LST instruction to prevent the ARP from being overwritten. If indirect addressing is used, the order is reversed.

**Example 2–18. Square Root Computation Using XC Instruction**

* Autocorrelation
* This routine performs a correlation of two vectors and then calls a Square Root subroutine that will determine the RMS amplitude of the waveform.
* AUTOC
  
  CALLD SQRT ;Call square root subroutine after executing next two instructions
  SST #0,ST0 ;Get the value to be passed to SQRT subroutine
  SST #1,ST1

* Square Root Computation
* This routine computes the square root of a number that is located in the higher half of accumulator. The number is in Q15 format.
* BRCR .set 09h ;DP=0
ST0 .set 60h ;Internal RAM block B2
ST1 .set 61h

2-24
Example 2–18. Square Root Computation Using XC Instruction (Continued)

NUMBER .set 62h
TEMPR .set 63h
GUESS .set 64h
.text
SQR T MAR *,AR0
LACC *
LD P #0
SE T C SXM ;Set SXM=1
SPM 1 ;Set PM mode for fractional arithmetic
SA CL NUMBER ;Save the number
LA CL #0
SA CB ;Clear accumulator buffer
SPL K #11,BRC R ;Initialize for 12 iterations
SPL K #800h,GUESS ;Set initial guess
LACC NUMBER
S U B #200h
BC ND D LOOP,LT ;If NUMBER<200h then begin looping
SPL K #800h,TEMPR
LACC #4000h ;Otherwise set initial guess
SA CL GUESS ;and temporary root to 4000h
SA CL TEM PR
SPL K #14,BRC R ;and increase iterations to 15
LOOP RPT B EN DL P–1 ;Repeat block
SQR A TEM PR ;Square temporary root
LACC NUMBER,16
SPAC ;Acc=NUMBER–TEMPR**2
NOP ;Dead cycle for XC
X C 2,GT ;If NUMBER>TEMPR**2 skip next 2 instr.
LACC TEM PR,16
SAC B ;Otherwise ROOT <- TEM PR
LACC GUESS,15
SAC H GUESS ;GUESS <- GUESS/2
ADD B
SACH TEM PR ;TEMPR <- GUESS+ROOT
EN DL P LAC B ;High Acc contains square root of NUMBER
RE TD
LST #1,ST1
LST #0,ST0 ;Restore context
2.8 Extended-Precision Arithmetic

Numerical analysis, floating-point computations, or other operations may require arithmetic to be executed with more than 32 bits of precision. Since the 'C5x devices are 16/32-bit fixed-point processors, software is required for the extended precision of arithmetic operations. Subroutines that perform the extended-precision arithmetic functions for the 'C5x are provided in the examples of this section. The technique consists of performing the arithmetic by parts, similar to the way in which longhand arithmetic is done.

The 'C5x has several features that help make extended-precision calculations more efficient. One of the features is the carry bit. The carry bit is affected by all arithmetic operations of the accumulator, including addition and subtraction with the accumulator buffer. This allows 32-bit-long arithmetic operations using the accumulator buffer as the second operand.

The carry bit is also affected by the rotate and shift accumulator instructions. It may also be explicitly modified by the load status register ST1 and the set/reset control bit instructions. For proper operation, the overflow mode bit should be reset (OVM = 0) so that the accumulator result is not loaded with the saturation value.

2.8.1 Addition

The carry bit is set (C = 1) whenever the input scaling shifter, the product register (PREG), or the accumulator buffer value added to the accumulator contents generates a carry out from bit 31. Otherwise, the carry bit is reset (C = 0) because the carry out from bit 31 is a 0. One exception to this case is the addition to the accumulator with a shift of 16 instruction (ADD dma,16), which can only set the carry bit. This allows the ALU to generate a proper single carry when the addition either to the lower or the upper half of the accumulator actually causes the carry. Figure 2–1 demonstrates the significance of the carry bit for additions.

Example 2–19 on page 2-28 shows an implementation of two 64-bit numbers added to each other to obtain a 64-bit result.
Figure 2–1. 32-Bit Addition

\[
\begin{array}{c|c|c}
\text{C MSB} & \text{LSB} & \text{C MSB} \\
\text{X} & F F F F F F F F (ACC) & X \\
+ & 1 & +F F F F F F F F \\
1 & 0 0 0 0 0 0 0 0 & 1 & F F F F F F F E \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{C MSB} & \text{LSB} & \text{C MSB} \\
\text{X} & 7 F F F F F F F F (ACC) & X \\
+ & 1 & +F F F F F F F F \\
0 & 8 0 0 0 0 0 0 0 & 1 & 7 F F F F F F F E \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{C MSB} & \text{LSB} & \text{C MSB} \\
\text{X} & 8 0 0 0 0 0 0 0 (ACC) & 1 \\
+ & 1 & +F F F F F F F F \\
0 & 8 0 0 0 0 0 0 1 & 1 & 7 F F F F F F F E \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{C MSB} & \text{LSB} & \text{C MSB} \\
1 & 0 0 0 0 0 0 0 0 (ACC) & 1 \\
+ & 0 (ADDC) & +F F F F F F F F \\
0 & 0 0 0 0 0 0 0 1 & 1 & 0 0 0 0 0 0 0 0 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{C MSB} & \text{LSB} & \text{C MSB} \\
1 & 8 0 0 0 F F F F (ACC) & 1 \\
+0 0 0 1 0 0 0 0 (ADD dma,16) & +7 F F F 0 0 0 0 (ADD dma,16) & +F F F F F F F F \\
1 & 8 0 0 0 F F F F & 1 & F F F F F F F F \\
\end{array}
\]
Example 2–19. 64-Bit Addition

* Two 64-bit numbers are added to each other producing a 64-bit result.
* The number X (X₃, X₂, X₁, X₀) and Y (Y₃, Y₂, Y₁, Y₀) are added resulting in
  W (W₃, W₂, W₁, W₀). If the result is required in 64-bit ACC/ACCB pair,
  replace the instructions as indicated in the comments below.

```
*  X₃ X₂ X₁ X₀
+  Y₃ Y₂ Y₁ Y₀
---
  W₃ W₂ W₁ W₀

ADD64  LACC  X₁,16    ;ACC = X₁ 00
       ADDS  X0       ;ACC = X₁ X₀
       ADDS  Y0       ;ACC = X₁ X₀ + 00 Y₀
       ADD   Y1,16    ;ACC = X₁ X₀ + Y₁ Y₀
       SACL  W0       ;THESE 2 INSTR ARE REPLACED BY
       SACH  W1       ;"SACB" IF RESULT IS DESIRED IN (ACC ACCB)
LACC  X₃,16    ;ACC = X₃ 00
ADDC  X₂       ;ACC = X₃ X₂ + C
ADDS  Y2       ;ACC = X₃ X₂ + 00 Y₂ + C
ADD   Y₃,16    ;ACC = X₃ X₂ + Y₃ Y₂ + C
SACL  W2       ;THESE 2 INSTR ARE NOT REQUIRED IF
SACH  W3       ;THE RESULT IS DESIRED IN (ACC ACCB)
RET
```
### 2.8.2 Subtraction

The carry bit is reset ($C = 0$) whenever the input scaling shifter, the PREG, or the accumulator buffer value subtracted from the accumulator contents generates a borrow into bit 31. Otherwise, the carry bit is set ($C = 1$) because no borrow into bit 31 is required. One exception to this case is the SUB dma,16 instruction, which can only reset the carry bit. This allows the ALU to generate a proper single carry when the subtraction either from the lower or the upper half of the accumulator actually causes the borrow. Figure 2–2 demonstrates the significance of the carry bit for subtraction.

Example 2–20 on page 2-30 shows an implementation of two 64-bit numbers subtracted from each other. A borrow is generated within the accumulator for each of the 16-bit parts of the subtraction operation.

---

#### Figure 2–2. 32-Bit Subtraction

<table>
<thead>
<tr>
<th>C</th>
<th>MSB</th>
<th>LSB</th>
<th>C</th>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>0 0 0 0 0 0 0 0 (ACC)</td>
<td>-</td>
<td>1</td>
<td>F F F F F F F F</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>F F F F F F F F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C</th>
<th>MSB</th>
<th>LSB</th>
<th>C</th>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>7 F F F F F F F (ACC)</td>
<td>-</td>
<td>1</td>
<td>F F F F F F F F</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7 F F F F F F F E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C</th>
<th>MSB</th>
<th>LSB</th>
<th>C</th>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>8 0 0 0 0 0 0 0 (ACC)</td>
<td>-</td>
<td>1</td>
<td>F F F F F F F F</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7 F F F F F F F F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C</th>
<th>MSB</th>
<th>LSB</th>
<th>C</th>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0 0 0 0 0 0 0 (ACC)</td>
<td>-</td>
<td>0</td>
<td>F F F F F F F F</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>F F F F F F F F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C</th>
<th>MSB</th>
<th>LSB</th>
<th>C</th>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8 0 0 0 F F F F (ACC)</td>
<td>-</td>
<td>0 0 1 0 0 0 0 0 (SUB dma,16)</td>
<td>F F F F F F F F</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>7 F F F F F F F F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C</th>
<th>MSB</th>
<th>LSB</th>
<th>C</th>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8 0 0 0 F F F F (ACC)</td>
<td>-</td>
<td>0 0 0 1 0 0 0 0 0 (SUB dma,16)</td>
<td>F F F F F F F F</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>8 0 0 1 F F F F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example 2–20. 64-Bit Subtraction

* Two 64-bit numbers are subtracted, producing a 64-bit result.
* The number Y (Y3, Y2, Y1, Y0) is subtracted from X (X3, X2, X1, X0) resulting
* in W (W3, W2, W1, W0). If the result is required in 64-bit ACC/ACCB pair,
* replace the instructions as indicated in the comments below.

* X3 X2 X1 X0
* – Y3 Y2 Y1 Y0
* -----------
* W3 W2 W1 W0 –OR– ACC ACCB

SUB64 LACC X1,16 ; ACC = X1 00
ADDS X0 ; ACC = X1 X0
SUBS Y0 ; ACC = X1 X0 – 00 Y0
SUB Y1,16 ; ACC = X1 X0 – Y1 Y0
SACL W0 ; THESE 2 INSTR ARE REPLACED BY
SACH W1 ; ”SACB” IF RESULT IS DESIRED IN (ACC ACCB)
LACL X2 ; ACC = 00 X2
SUBB Y2 ; ACC = 00 X2 – 00 Y2 – C
ADD X3,16 ; ACC = X3 X2 – 00 Y2 – C
SUB Y3,16 ; ACC = X3 X2 – Y3 Y2 – C
SACL W2 ; THESE 2 INSTR ARE NOT REQUIRED IF
SACH W3 ; THE RESULT IS DESIRED IN (ACC ACCB)
RET

2.8.3 Multiplication

Another important feature that aids in extended-precision calculations is the
MPYU (unsigned multiply) instruction. The MPYU instruction allows two
unsigned 16-bit numbers to be multiplied and the 32-bit result placed in the
PREG in a single cycle. Efficiency is gained by generating partial products
from the 16-bit portions of a 32-bit or larger value, instead of having to split the
value into 15-bit or smaller parts.

Further efficiency is gained by using the accumulator buffer to hold partial
results, instead of using a temporary location in data memory. The ability of the
'C5x devices to barrel shift the accumulator by 1 to 16 bits in only one cycle
is also useful for scaling and justifying operands.

For 16-bit integer multiplication, in which one operand is a 2s-complement
signed integer and the other is an unsigned integer, the algorithm shown in
Figure 2–3 can be used.
Figure 2–3. 16-Bit Integer Multiplication Algorithm

Steps required:

1) Multiply two operands X and Y as if they are signed integers.

2) If MSB of the unsigned integer Y is 1, add X to the upper half of the 32-bit signed product.

The correction factor must be added to the signed multiplication result because the bit weight of the MSB of any 16-bit unsigned integer is $2^{15}$.

Consider the following representation of a signed integer X and an unsigned integer Y:

$$X = -2^{15}x_{15} + 2^{14}x_{14} + 2^{13}x_{13} + ... + 2^1x_1 + 2^0x_0$$

$$Y = 2^{15}y_{15} + 2^{14}y_{14} + 2^{13}y_{13} + ... + 2^1y_1 + 2^0y_0$$

Multiplication of X and Y yields:

$$X \times Y = X \times (2^{15}y_{15} + 2^{14}y_{14} + 2^{13}y_{13} + ... + 2^1y_1 + 2^0y_0)$$

$$= 2^{15}y_{15}X + 2^{14}y_{14}X + 2^{13}y_{13}X + ... + 2^1y_1X + 2^0y_0X \quad \text{(1)}$$

However, if X and Y are considered signed integers, their multiplication yields:

$$X \times Y = X \times (-2^{15}y_{15} + 2^{14}y_{14} + 2^{13}y_{13} + ... + 2^1y_1 + 2^0y_0)$$

$$= -2^{15}y_{15}X + 2^{14}y_{14}X + 2^{13}y_{13}X + ... + 2^1y_1X + 2^0y_0X \quad \text{(2)}$$
The difference between equations (1) and (2) is in the first term on the right-hand side of the two equations.

Hence, if we add the correction term, $2^{16}y_{15}X$, to equation (2), the result would be identical to that of equation (1) and is the correct result.

This method of multiplying a signed integer with an unsigned integer can be used to implement extended-precision multiplication on the 'C5x. Figure 2–4 shows a 32-bit multiplication algorithm based on this method. Example 2–21 on page 2-33 implements this algorithm. The product is a 64-bit integer number. Note the use of BSAR and XC instructions.

Example 2–22 on page 2-34 performs fractional multiplication. The operands are in Q31 format, while the product is in Q30 format.

Figure 2–4. 32-Bit Multiplication Algorithm
Example 2–21. 32-Bit Integer Multiplication

.title "32–bit Optimized Integer Multiplication"
def MPY32
*;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
* This routine multiplies two 32-bit signed integers resulting in a 64-bit
* product. The operands are fetched from data memory and the result is
* written back to data memory.
* Data Storage:
* X1,X0 32-bit operand
* Y1,Y0 32-bit operand
* W3,W2,W1,W0 64-bit product
* Entry Conditions:
* DP = 6, SXM = 1
* OVM = 0
*;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
X1 .set 300h ;DP=6
X0 .set 301h ;DP=6
Y1 .set 302h ;DP=6
Y0 .set 303h ;DP=6
W3 .set 304h ;DP=6
W2 .set 305h ;DP=6
W1 .set 306h ;DP=6
W0 .set 307h ;DP=6
.text
MPY32:  
BIT X0,0 ;TC = X0 bit#15 
LT X0 ;T = X0 
MPYU Y0 ;P = X0Y0 
SPL W0 ;Save W0 
SPH W1 ;Save partial W1 
MPY Y1 ;P = X0Y1 
LTP X1 ;ACC = X0Y1, T = X1 
MPY Y0 ;P = X1Y0 
MPPA Y1 ;ACC = X0Y1+X1Y0, P=X1Y1 
ADDS W1 ;ACC = X0Y1+X1Y0+X0Y02^-16 
SACL W1 ;Save final W1 
BSAR 16 ;Shift ACC right by 16 
XC 1,TC ;If MSB of X0 is 1 
ADD Y1 ;Add Y1 
BIT Y0,0 ;TC = Y0 bit#15 
APAC ;ACC = X1Y1 + (X0Y1+X1Y0)2^-16 
XC 1,TC ;If MSB of Y0 is 1 
ADD X1 ;Add X1 
SACL W2 ;Save W2 
SACH W3 ;Save W3
Example 2–22. 32-Bit Fractional Multiplication

```
.title "32-bit Fractional Multiplication"
*;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;*
* This routine multiplies two Q31 signed integers resulting in a Q30 product.  *
* The operands are fetched from data memory and the result is written back *
* to data memory.  *
* Data Storage:  *
*  X1,X0      Q31 operand  *
*  Y1,Y0      Q31 operand  *
*  W1,W0      Q30 product  *
* Entry Conditions:  *
*  DP = 6, SXM = 1  *
*  OVM = 0  *
*
*;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;*
X1 .set 300h ;DP=6
X0 .set 301h ;DP=6
Y1 .set 302h ;DP=6
Y0 .set 303h ;DP=6
W1 .set 304h ;DP=6
W0 .set 305h ;DP=6
.text
BIT X0,0 ;TC = X0 bit#15
LT X0 ;TREG0 = X0
MPY Y1 ;P = X0*Y0
LTP X1 ;ACC = X0*Y0
MPY Y0 ;P = X1*Y0
MPYA Y1 ;ACC = X0*Y0 + X1*Y0
BSAR 16 ;Throw away low 16 bits
XC 1,TC ;If MSB of X0 is 1
ADD Y1 ;then add Y1
BIT Y0,0 ;TC = Y0 bit#15
APAC ;ACC = ACC + X1*Y1
XC 1,TC ;If MSB of Y0 is 1
ADD X1 ;then add X1
SACL W0 ;Save lower product
SACH W1 ;Save upper product
```
2.8.4 Division

Integer and fractional division is implemented on the 'C5x by repeated subtractions executed with SUBC, a special conditional subtract instruction. Given a 16-bit positive dividend and divisor, the repetition of the SUBC command 16 times produces a 16-bit quotient in the low accumulator and a 16-bit remainder in the high accumulator.

SUBC implements binary division in the same manner as long division is done (Figure 2–5). The dividend is shifted until subtracting the divisor no longer produces a negative result. For each subtract that does not produce a negative answer, a 1 is put in the LSB of the quotient and then shifted. The shifting of the remainder and quotient after each subtract produces the separation of the quotient and remainder in the low and high halves of the accumulator, respectively.

Both the dividend and the divisor must be positive when using the SUBC command. Thus, the sign of the quotient must be determined and the quotient computed by using the absolute value of the dividend and divisor.

Integer division can be implemented with the SUBC instruction, as shown in Example 2–23 on page 2-37. For integer division, the absolute value of the numerator must be greater than the absolute value of the denominator.

Fractional division can also be implemented with the SUBC instruction as shown in Example 2–24 on page 2-38. When implementing a division algorithm, it is important to know if the quotient can be represented as a fraction and the degree of accuracy to which the quotient is to be computed. For fractional division, the absolute value of the numerator must be less than the absolute value of the denominator. Note that the dividend is loaded into the high accumulator and that only N–1 iterations are required for an N-bit fraction.
Extended-Precision Arithmetic

Figure 2–5. 16-Bit Integer Division

LONG DIVISION:

\[
\begin{array}{c}
\text{0000 0000 0000 0101 000 0000 0010 0001} \\
\text{0000 0000 0000 0110} \\
\hline
\text{110} \\
\text{101} \\
\hline
\text{11 REMAINDER}
\end{array}
\]

SUBC METHOD:

<table>
<thead>
<tr>
<th>32</th>
<th>HIGH ACC</th>
<th>16</th>
<th>15</th>
<th>LOW ACC</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 0000 0000 0000</td>
<td>0000 0000 0010 0001</td>
<td>-10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000 0000 0000 0000</td>
<td>0000 0000 0100 0010</td>
<td>-10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000 0000 0000 0000</td>
<td>0000 0000 0111 1111</td>
<td>-10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Dividend is loaded into ACC. The divisor is left-shifted 15 and subtracted from ACC. The result is negative, so discard the result, shift ACC left one bit, and replace LSB with 0.</td>
</tr>
<tr>
<td>2</td>
<td>Second SUBC command. The result is negative, so discard the result, shift ACC (dividend) left one bit, and replace LSB with 0.</td>
</tr>
<tr>
<td>14</td>
<td>14th SUBC command. The result is positive. Shift result left one bit and replace LSB with 1.</td>
</tr>
<tr>
<td>15</td>
<td>15th SUBC command. The result is again positive. Shift result left one bit and replace LSB with 1.</td>
</tr>
<tr>
<td>16</td>
<td>16th SUBC command. The result is negative, so discard the result, shift ACC left one bit, and replace LSB with 0. Answer reached after 16 SUBC commands stored in ACC.</td>
</tr>
</tbody>
</table>
Example 2–23. 16-Bit Integer Division Using SUBC Instruction

* This routine implements integer division with the SUBC instruction. For this
* integer division routine, the absolute value of the numerator must be greater
* than the absolute value of the denominator. In addition, the calling routine
* must check to verify that the divisor does not equal 0.
*
* The 16-bit dividend is placed in the low accumulator, and the high accumulator
* is zeroed. The divisor is in data memory. At the completion of the last SUBC,
* the quotient of the division is in the lower-order 16-bits of the accumulator.
* The remainder is in the higher-order 16-bits.
*
* Key C5x Instruction:
* RETCD return if conditions true – after executing next 2-word instruction or
* two single-word instructions
*
DENOM .set 60h
NUMERA .set 61h
QUOT .set 62h
REM .set 63h
TEMSGN .set 64h
*
INTDIV LDP #0
LT NUMERA ;Determine sign of quotient.
MPY DENOM
*
SPH TEMSGN ;Save the sign
LAACL DENOM
ABS ;Make denominator and numerator positive.
SAACL DENOM ;Save absolute value of denominator
LAACL NUMERA
ABS
*
* If divisor and dividend are aligned, division can start here.
*
RPT #15 ;16 cycle division. Low accumulator contains
* the quotient and high accumulator contains
* the remainder at the end of the loop.

BIT TEMSGN,0 ;Test sign of quotient.
RETC5 NTC ;Return if sign positive, else continue.
SAACL QUOT ;Store quotient and remainder during delayed
SACH REM ;return.
*
LAACL #0 ;If sign negative, negate quotient and return
RETD
SUB QUOT
SAACL QUOT
Example 2–24. 16-Bit Fractional Division Using SUBC Instruction

* This routine implements fractional division with the SUBC instruction. For this division routine, the absolute value of the denominator must be greater than the absolute value of the numerator. In addition, the calling routine must check to verify that the divisor does not equal 0.*

* The 16-bit dividend is placed in the high accumulator, and the low accumulator is zeroed. The divisor is in data memory.

DENOM .set 60h
NUMERA .set 61h
QUOT .set 62h
REM .set 63h
TEMSGN .set 64h

FRACDIV LDP #0
LT NUMERA ;Determine sign of quotient.

MPY DENOM
SPH TEMSGN
LACL DENOM
ABS ;Make denominator and numerator positive.
SACL DENOM
LACC NUMERA,16 ;Load high accumulator, zero low accumulator.
ABS

* If divisor and dividend are aligned, division can start here.

RPT #14 ;15-cycle division. Low accumulator contains the quotient and high accumulator contains the remainder at the end of the loop.

SUBC DENOM

BIT TEMSGN,0 ;Test sign of quotient.
RETCN NTC ;Return if sign positive, else continue.
SACL QUOT ;Store quotient and remainder during delayed return.*
SACH REM
LAACL #0 ;If sign negative, negate quotient and return
RETD
SUB QUOT
SAACL QUOT
2.9 Floating-Point Arithmetic

To implement floating-point arithmetic on the 'C5x, operands must be converted to fixed point for arithmetic operations and then converted back to floating point. Conversion to floating-point notation is performed by normalizing the input data.

To multiply two floating-point numbers, the mantissas are multiplied and the exponents added. The resulting mantissa must be renormalized. Floating-point addition or subtraction requires shifting the mantissa so that the exponents of the two operands match. The difference between the exponents is used to left shift the lower power operand before adding. Then, the output of the add must be renormalized.

The 'C5x instructions used in floating-point operations are NORM, SATL, SATH, and XC. NORM may be used to convert fixed-point numbers to floating-point numbers. SATL in combination with SATH provides a 2-cycle 0 through 31-bit right shift. XC helps avoid extra cycles caused by branch instructions.

Example 2–25 on page 2-40 and Example 2–26 on page 2-44 show how to implement floating-point arithmetic on the 'C5x. Floating-point numbers are generally represented by mantissa and exponent values. Single-precision IEEE floating-point numbers are represented by a 24-bit mantissa, an 8-bit exponent, and a sign bit. In order to simplify the routines, a format slightly different from the IEEE format is used. Four words are occupied by each floating-point number. One sign word, one word for the exponent, and two words for the mantissa are reserved in memory as shown in the examples.
Example 2–25. Floating-Point Addition Using SATL and SATH Instructions

```
.title 'Floating Point Addition Algorithm'
.def FL_ADD

* THIS SUBROUTINE ADDS TWO FLOATING-POINT NUMBERS PRODUCING A NORMALIZED
* FLOATING-POINT PRODUCT. THE FORMAT OF FLOATING-POINT NUMBERS IS
* SPECIFIED BELOW.
*
* INPUT / OUTPUT FORMAT
* -----------------------
* | SIGN WORD |
* |EXPONENT |
* |HIGH PART OF MANTISSA|
* |LOW PART OF MANTISSA|
*
* Key C5x Instructions:
* SANN save the accumulator contents in a memory-mapped register
* LAOC accumulator is loaded with contents of accumulator buffer
* SACC contents of accumulator are copied in accumulator buffer
* SATL accumulator is barrel-shifted right by the value specified
* in the 4 LSBs of TREG1
* SATH accumulator is barrel-shifted right by 16 bits if bit 4 of
* TREG1 is a one.
* SPLK store immediate long constant in data memory
* CPL compare long immediate value (or DBMR) with data memory
* TC=1 if two values are same
* TC=0 otherwise
*
```

TREG1 .set 0dh
ASIGN .set 60h ; Sign, exponent, high and low part of mantissa
AEXP .set 61h ; of input number A
AHI .set 62h
ALO .set 63h
BSIGN .set 64h ; Sign, exponent, high and low part of mantissa
BEXP .set 65h ; of input number B
BHI .set 66h
BLO .set 67h
Example 2–25. Floating-Point Addition Using SATL and SATH Instructions (Continued)

CSIGN .set 68h ;Sign, exponent, high and low part of mantissa
CEXP .set 69h ;of the resulting floating point number C
CHI .set 6Ah
CLO .set 6Bh
DIFFEXP .set 6Ch

.text
FL_ADD LDP #0 ;Initialization
    SETC SXM ;Set sign extension mode
    MAR *,AR0 ;ARP <- AR0
    LAR AR0,#0 ;AR0 is used by NORM instruction
    CMPEXP LACL BLO ;Load low Acc with BLO
    ADD BHI,16 ;Add BHI to high Acc
    SACB ;AccB = BHIBLO
    LACC AEXP
    SUB BEXP ;Acc = AEXP+BEXP
    SACL DIFFEXP ;Save the difference
    BCND AEQB,EQ ;If |A| == |B|
    BCND ALTB,LT ;If |A| < |B|
    AGTB LACC DIFFEXP ;If |A| > |B|
    SAMM TREG1 ;Load TREG1 with # of right shifts reqd.
    SUB #32
    BCND AGRT32,GEQ ;If difference > 32
    LACB ;Acc = BHIBLO
    SATL
    SATH ;Right justify BHIBLO
    SACB ;Store the result back in AccB
    AEQB LACC ASIGN ;Copy sign and exponent values of
    SACL CSIGN ;A in C (i.e. the result)
    LACC AEXP
    SACL CEXP
    CHKSGN LACC ASIGN ;Acc=ASIGN-BSIGN
    SUB BSIGN
    CLRC TC ;Clear TC flag
    XC 1,LT ;If A<0 and B>0
    SETC TC ;Set TC flag
    BCNDD ADNOW,EQ ;If both A and B have same sign
    LAACL ALO
    ADD AHI,16 ;Acc = AHIALO
    SBB ;Acc=A-B
    XC 1,TC ;If A<0 and B>0
    SETC TC ;Set TC flag
    BCND CZERO,EQ ;If A-B == 0
    XC 2,LT ;If A-B < 0
    SPLK #0FFFFh,CSIGN;then CSIGN=-1
    XC 2,GT ;If A-B > 0
    SPLK #0,CSIGN;then CSIGN=0
    XC 1,LT ;If A-B<0
    ABS ;then Acc=|A-B|
    BD NORMAL ;delayed branch
    SACH CHI ;Save the result
    SAACL CLO
Example 2–25. Floating-Point Addition Using SATL and SATH Instructions (Continued)

```assembly
CZERO LACL #0 ;If A–B == 0
SACL CEXP ;then result is zero
SACL CSIGN ;Make sign positive
RETD ;Return delayed
SACL CHI
SACL CLO ;Clear CHICLO
ADDB ;If signs are same
BCNDD OVFLOW,OV ;then add two numbers
SACH CHI
SACL CLO ;Save it in CHICLO
BCND CZERO,EQ ;If CHICLO is zero, goto CZERO
NORMAL CPL #0,CHI ;Compare CHI with 0
NOP ;Dead cycle for XC
XC 2,TC ;If CHI is 0
LACC CLO,16 ;then normalize only the CLO part
LAR AR0,#16 ;AR0 has exponent value
XC 2,NTC ;If CHI != 0
LACC CHI,16 ;Acc=CHICLO
ADDS CLO
CLRC SXM ;Disable sign extension mode
XC 2,LT ;If MSB of CLO is 1
SBRK 1 ;then shift right once
SFR ;and decrement exponent.
SETC SXM ;Enable sign extension mode
RPT #13 ;Repeat 14 times
NORM *+ ;Normalize
OUTPUT SACH CHI ;Store high part
SACL CLO ;Store low part of the result
LACC CEXP
SAR AR0,CEXP ;Save exponent
RETD ;Return delayed
SUB CEXP
SACL CEXP ;CEXP=CEXP-ARG
OVFLOW CLRC SXM ;Disable sign extension mode
SFR ;Shift Acc right
SACH CHI
SACL CLO ;Save the result
LACC CEXP
ADD #1 ;Increment exponent by one
SACL CEXP ;Save it
ALTB LACC BSIGN ;Copy sign of B in C
SACL CSIGN
LACC BEXP ;Copy exponent of B in C
SACL CEXP
LACC DIFFEXP
NEG ;since A–B < 0 here
SAMM TREG1 ;No. of shifts reqd. for right-justification
SUB #32
BCND BGR32,GEQ ;difference in exponent >= 32
LACL ALO
ADD AHI,16 ;Acc=AHIALO
```
Example 2–25. Floating-Point Addition Using SATL and SATH Instructions (Continued)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SATL</td>
<td>Right-justify ALOAHI</td>
</tr>
<tr>
<td>SATH</td>
<td>Jump back after next two instructions</td>
</tr>
<tr>
<td>BD CHKSGN</td>
<td>Jump back after next two instructions</td>
</tr>
<tr>
<td>SACL ALO</td>
<td>Save normalized value</td>
</tr>
<tr>
<td>SACH AHI</td>
<td>in ALO and AHI</td>
</tr>
<tr>
<td>BGRT32</td>
<td>If exponent of B &gt; 32</td>
</tr>
<tr>
<td>LACC BHI</td>
<td>then C &lt;- B.</td>
</tr>
<tr>
<td>RETD</td>
<td>Return after</td>
</tr>
<tr>
<td>LACC BLO</td>
<td>saving CHI and CLO</td>
</tr>
<tr>
<td>SACL CLO</td>
<td></td>
</tr>
<tr>
<td>AGRT32</td>
<td>If exponent of A &gt; 32</td>
</tr>
<tr>
<td>LACC AHI</td>
<td>then C &lt;- A.</td>
</tr>
<tr>
<td>SACL CHI</td>
<td></td>
</tr>
<tr>
<td>LACC ALO</td>
<td>Copy ALO to CLO</td>
</tr>
<tr>
<td>SACL ASIGN</td>
<td>Copy ASIGN to CSIGN</td>
</tr>
<tr>
<td>RETD</td>
<td>Return after</td>
</tr>
<tr>
<td>LACC AEXP</td>
<td>Copying AEXP to CEXP</td>
</tr>
<tr>
<td>SACL CEXP</td>
<td></td>
</tr>
</tbody>
</table>
Example 2–26. Floating-Point Multiplication Using BSAR Instruction

```
/title 'Floating Point Multiplication Routine'
*;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
* THIS SUBROUTINE MULTIPLIES TWO FLOATING-POINT NUMBERS PRODUCING
* A NORMALIZED FLOATING-POINT PRODUCT. THE FORMAT OF FLOATING-
* POINT NUMBERS IS SPECIFIED BELOW.
* INPUT / OUTPUT FORMAT
* =====================
* | ALL 0 OR 1 | SIGN WORD
* -------------------
* | 16 BITS | EXPONENT
* -------------------
* | 15 BITS | HIGH PART OF MANTISSA
* -------------------
* | 16 BITS | LOW PART OF MANTISSA
* -------------------
* NOTE THAT EVEN IF THE PRODUCT IS ZERO, SIGN OF THE PRODUCT MAY
* EITHER BE POSITIVE OR NEGATIVE DEPENDING ON THE INPUTS.
* Key C5x Instructions:
* BSAR 1-16 bit right barrel arithmetic shift in one cycle
* CLRC reset control bit
* SETC set control bit
* BD branch after executing next two one-word instructions
* or one two-word instruction
* ASIGN .set 60h ;Sign, exponent, high and low parts of mantissa
* AEXP .set 61h ;of input number A
* AHI .set 62h
* ALO .set 63h
* BSIGN .set 64h ;Sign, exponent, high and low parts of mantissa
* BEXP .set 65h ;of input number B
* BHI .set 66h
* BLO .set 67h
* CSIGN .set 68h ;Sign, exponent, high and low parts of mantissa
* CEXP .set 69h ;of the resulting floating point number C
* CHI .set 6ah
* CLO .set 6bh
```
Example 2–26. Floating-Point Multiplication Using BSAR Instruction (Continued)

```
.text
MULT
    LDP #0
    MAR *,AR0 ;ARP <- AR0
    LAR AR0,#0 ;Reset exponent counter
    SPM 0 ;No left shift of P register
    LACC AEXP
    ADD BEXP
    SAACL CEXP ;CEXP = AEXP + BEXP
    CLR CXM ;for barrel shift, disable sign extension
    LT ALO ;T = ALO
    MPYU BHI ;P = ALO*BHI
    LTP AHI ;Acc=ALO*BHI, T=AHI
    MPYU BLO ;P=AHI*BLO
    MPYA BHI ;Acc=ALO*BHI + AHI*BLO, P=AHI*BHI
    BSAR 16 ;Retain upper 16 bits plus 1 additional
    APAC ;bit due to zero MSBs of BLO & ALO
    BCND NZERO,NEQ ;If the product is not zero
    SACH CHI ;If the product is zero
    BD SIGN ;then clear CHI,CLO and CEXP
    SAACL CLO ;and jump to SIGN
    SAACL CEXP
NZERO
    SFL ;Discard additional sign bit (Q63)
    NORM *+ ;Remove leading zero if any
    SACH CHI ;Save product
    SAACL CLO
    SETC CXM ;Enable sign extension mode
    LACC CEXP
    SAR AR0,CEXP ;CEXP<-AR0
    SUB CEXP
    SAACL CEXP ;CEXP=CEXP-AR0
SIGN
    LACL ASIGN ;If signs are same then product is +ve
    RETD ;Return after next two instructions
    XOR BSIGN ;otherwise it is -ve.
    SAACL CSIGN
```
2.10 Application-Oriented Operations

The following subsections provide application-oriented operations for:

- modem applications
- adaptive filtering
- infinite impulse response (IIR) filters
- dynamic programming

2.10.1 Modem Application

Digital signal processors are especially appropriate for modem applications. The 'C5x devices with their enhanced instruction set and reduced instruction cycle time are particularly effective in implementing encoding and decoding algorithms. Features like circular addressing, repeat block, and single-cycle barrel shift reduce the execution time of such routines.

Example 2–27 on page 2-47 shows a differential and convolutional encoder for a 9600-bit/second V.32 modem. This encoder uses trellis coding with 32 carrier states. The data stream to be transmitted is divided into groups of four consecutive data bits. The first two bits in time Q1_n and Q2_n in each group are differentially encoded into Y1_n and Y2_n according to the following equations:

\[
Y1_n = Q1_n \oplus Y1_{n-1}
\]

\[
Y2_n = (Q1_n \cdot Y1_{n-1}) \oplus Y2_{n-1} \oplus Q2_n
\]

This is done by a subroutine called DIFF. The two differentially encoded bits Y1_n and Y2_n are used as inputs to a convolutional encoder subroutine ENCODE, which generates a redundant bit Y0_n. These five bits are packed into a single word by the PACK subroutine.
Example 2–27. V.32 Encoder Using Accumulator Buffer

```
.title 'Convolutional Encoding for a V.32 Modem'
.mmregs

STATMEM .set 60h ; (60h - 62h) Delay States S1, S2, S3
INPUT .set 64h ; (64h - 67h) Four input bits
YPAST .set 68h ; (68h - 69h) Past values of Y1 and Y2
OUTPUT .set 63h ; Y0, the redundant bit
LOCATE .set 6ah ; Temporary storage for current input word
PCKD_IP .set 1000h ; Input buffer (4 bits packed per word)
PCKD_OP .set 2000h ; Output buffer (5 bits packed per word)
COUNT .set 50 ; # of input data words

.text

INIT LAR AR1, #PCKD_IP
LAR AR2, #PCKD_OP
LAR AR3, #COUNT-1 ; COUNT contains # of input words
LDP #0

START MAR *, AR1
LACC *, 0, AR0
SACL LOCATE ; Temporary storage for current input word
LAR AR0, #INPUT+3
LA CL #3 ; Loop 4 times
SAMM BRCR
LA CL #1
SAMM DBMR ; Load DBMR with the mask for LSB
LACC LOCATE ; Acc = packed input bits
RPTB LOOP1-1 ; for I=0, I<=3, I++
SA CL * ; Save it
APL *- ; Mask off all bits except LSB
SFR ; Shift right to get next bit

LOOP1
CALL DIFF ; Call differential encoder
CALL ENCODE ; Call convolutional encoder

PACK LAR AR0, #INPUT
LA CL #3 ; Loop 4 times only
SAMM BRCR
LACC ++ ; Get first bit (MSB)
RPTB LOOP2-1 ; for I=0, I<=2, I++
SFL ; Make space by left-shifting once
ADD ++ ; Pack next bit by left-shifting other
NOP

LOOP2
MAR *, AR2 ; ARP <- AR2
SA CL ++, 0, AR3 ; Save it in packed form
BANZ START ; Loop if COUNT is not zero
RET ; Return
```
Example 2–27. V.32 Encoder Using Accumulator Buffer (Continued)

; This subroutine differentially encodes Q1n and Q2n (INPUT buffer)
; according to previous output values Y1n-1 and Y2n-1 (YPAST buffer).
; The resulting values Y1n and Y2n overwrite previous Q1n and Q2n.

DIFF   LACC  YPAST ;Acc=Y1n-1
   AND  INPUT ;Q1n & Y1n-1
   XOR  INPUT+1 ;(Q1n & Y1n-1) xor Q2n
   XOR  YPAST+1 ;(Q1n & Y1n-1) xor Q2n xor Y2n-1
  SACL  INPUT+1
  SACL  YPAST+1 ;Save Y2n
  LACC  YPAST
  XOR  INPUT ;Q1n xor Y1n-1
  RETD ;Delayed return
  SACL  INPUT ;Save Y1n
  SACL  YPAST ;save Y1n-1

; This subroutine generates a redundant bit Y0n by convolutional encoding,
; taking Y1n and Y2n as input. Three delay states S1, S2 and S3 are
; located in STATMEM buffer.
ENCODER LACC  STATMEM
SACL  OUTPUT ;Y0 <- S1
LACC  INPUT+1
XOR  STATMEM+1 ;Y2 xor S2
SACB ;Save in AccB
LACC  OUTPUT
AND  INPUT ;Y0 & Y1
XORB ;(Y0 & Y1) xor (Y2 xor S2)
SACL  STATMEM ;Save it in S1
LACC  OUTPUT
ANDB ;Y0 & (Y2 xor S2)
SACB
LACC  INPUT
XOR  INPUT+1 ;Y1 xor Y2
XOR  STATMEM+2 ;(Y1 xor Y2) xor S3
XORB ;((Y1 xor Y2) xor S3) xor (Y0 & (Y2 xor S2))
SACL  STATMEM+1 ;Update S2
RETD ;Delayed return
LACC  OUTPUT
SACL  STATMEM+2 ;Update S3
2.10.2 Adaptive Filtering

There are many practical applications of adaptive filtering; one example is in the adapting or updating of coefficients. This can become computationally expensive and time-consuming. The MPYA, ZALR, and RPTB instructions on the 'C5x can reduce execution time.

A means of adapting the coefficients on the 'C5x is the least-mean-square algorithm given by the following equation:

\[ b_k(i + 1) = b_k(i) + 2B_e(i)x(i-k) \]

where \( e(i) = x(i) - y(i) \)

and

\[ y(i) = \sum_{k=0}^{N-1} b_k x(i-k) \]

Quantization errors in the updated coefficients can be minimized if the result is obtained by rounding rather than truncating. For each coefficient in the filter at a given point in time, the factor \( 2B_e(i) \) is a constant. This factor can then be computed once and stored in the TREG0 for each of the updates.

MPYA and ZALR instructions help in reducing the number of instructions in the main adaptation loop. Furthermore, the RPTB (repeat block) instruction allows the block of instructions to be repeated without any penalty for looping.

Example 2–28 on page 2-50 shows a routine that implements a 128-tap finite impulse response (FIR) filter and an LMS adaptation of its coefficients. The SARAM of the 'C5x can be mapped in both the program and data spaces at the same time by setting the OVLY and RAM control flags to 1. This feature can be used to locate the coefficient table in SARAM so that the table can be accessed by the MACD and MPY instructions without modifying the RAM configuration. Note that the MACD instruction requires one of its operands to be in program space.

If the address of the coefficient table is to be determined in runtime, load the BMAR (block move address register) with the address computed dynamically and replace the instruction MACD COEFFP,*− with MADD *− .
Example 2–28. Adaptive FIR Filter Using RPT and RPTB Instructions

```plaintext
.title 'Adaptive Filter'
def ADPFIR
def X,Y
.mmregs

* This 128–tap adaptive FIR filter uses on–chip memory block B0 for
* coefficients and block B1 for data samples. The newest input should be in
* memory location X when called. The output will be in memory location Y
* when returned.
*
* OVLY =1 , RAM =1 when this routine is called.
*
COEFFP .set 02000h ;Program memory address of the coeff. in S/A RAM
COEFFD .set 02000h ;Data memory address of the coeff. in S/A RAM
* For 'C51,'C53,'C56,'C57, COEFFD is 0800h instead of 02000h
ONE .set 7Ah ;Constant one. (DP=0).
BETA .set 7Bh ;Adaptation constant. (DP=0).
ERR .set 7Ch ;Signal error. (DP=0).
ERRF .set 7Dh ;Error function. (DP=0).
Y .set 7Eh ;Filter output. (DP=0).
X .set 037Fh ;Newest data sample.
FRSTAP .set 0380h ;Next newest data sample.
LASTAP .set 03FFh ;Oldest data sample.
*
* Finite impulse response (FIR) filter.
*
ADPFIR ZPR ;Clear P register.
LACC #1,14 ;Load output rounding bit.
MAR *,AR3
LAR AR3,#LASTAP ;Point to oldest sample.
FIR RPT #127
MACD COEFFP,*– ;128–tap FIR filter.
APAC
SACH Y,1 ;Store the filter output.
NEG ;Acc = –y(n)
LAR AR3,#X
ADD *,15 ;Add the newest input sample.
SACH ERR,1 ;err(n) = x(n) – y(n)
DMOV * ;Include newest sample
*
* LMS Adaption of Filter Coefficients.
*
LT ERR ;T = err
MPY BETA ;P = beta*err(i)
PAC ;errf(i) = beta * err(i)
ADD ONE,14 ;Round the results.
SACH ERRF,1 ;Save errf(i)
LACC #126
SAMM BRCR ;127 coefficients to update in the loop.
LAR AR2,#COEFFD ;Point to the coefficients.
LAR AR3,#LASTAP+1;Point to the data samples.
```
Example 2–28. Adaptive FIR Filter Using RPT and RPTB Instructions (Continued)

```
LT ERRF
  MPY *,AR2 ; P = 2*beta*err(i)*x(i-255)
  *
  RPTB LOOP-1 ; For I=0, I<=126, I++
    ADAPT ZALR *,AR3 ; Load ACC with ak(i).
      MPYA *,AR2 ; P = 2*beta*err(i)*x(i-k-1)
      * Acc = ak(i) + 2*beta*err(i)*x(i-k)
      SACH ++ ; Store ak(i+1)
      *
  LOOP ZALR *,AR3 ; Finally update last coeff. a0(i)
    RETD ; Delayed return
    APAC ; Acc = a0(i) + 2*beta*err(i)*x(i)
    SACH ++ ; Save a0(i+1)
```
2.10.3 Infinite Impulse Response (IIR) Filters

Infinite impulse response (IIR) filters are widely used in digital signal processing applications. The transfer function of an IIR filter is given by:

\[
H(z) = \frac{b_0 + b_1 z^{-1} + \ldots + b_M z^{-M}}{1 + a_1 z^{-1} + \ldots + a_N z^{-N}} = \frac{Y(z)}{X(z)}
\]

Figure 2–6 shows a block diagram of an Nth-order, direct-form, type II, IIR filter. In the time domain, an Nth-order IIR filter is represented by the following two difference equations:

At time interval \(n\):

\(x(n)\) is the current input sample
\(y(n)\) is the output of the IIR filter
\(d(n) = x(n) - d(n-1)a_1 - \ldots - d(n-N+1)a_{N-1}\)
\(y(n) = d(n)b_0 + d(n-1)b_1 + \ldots + d(n-N+1)b_{N-1}\)

The two equations above can easily be implemented on the 'C5x using the multiply-accumulate instructions (MAC, MACD, MADS, MADD). Note that the second equation also requires a data-move operation to update the state variable sequence \(d(n)\). Example 2–29 on page 2-53 implements an Nth-order IIR filter using single-instruction repeat (RPT) and multiply-accumulate (MAC, MACD) instructions.

Figure 2–6. Nth-Order, Direct-Form, Type II, IIR Filter
Example 2–29. Nth-Order IIR Filter Using RPT and MACD Instructions

Application-Oriented Operations

Due to the recursive nature of an IIR filter, quantization of filter coefficients may cause significant variation from the desired frequency response. To avoid this problem, the desired filter transfer function can be broken up into lower order sections that are cascaded with each other. Example 2–30 on page 2-54 shows an implementation of N cascaded second-order IIR sections (also called biquad sections). The filter coefficients and the state variables are stored in data memory. Note the use of LTD and MPYA instructions to perform multiply-accumulate and data-move operations.
Example 2–30. N Cascaded BiQuad IIR Filter Using LTD and MPYA Instructions

\[ \text{title "N Cascaded BiQuad IIR Filters"} \]
\[ \text{.mmregs} \]

* This routine implements \( N \) cascaded blocks of biquad IIR canonic type II filters. Each biquad requires 3 data memory locations \( d(n), d(n-1), d(n-2) \), and 5 coefficients \(-a_1, a_2, b_0, b_1, b_2\).

* For each block: \[
\begin{align*}
d(n) &= x(n) - d(n-1)a_1 - d(n-2)a_2 \\
y(n) &= d(n)b_0 + d(n-1)b_1 + d(n-2)b_2
\end{align*}
\]

* Coefficients Storage (low to high data memory):
  - \(-a_2, -a_1, b_2, b_1, b_0\)
  - 1st biquad \( \rightarrow \) \(N\)th biquad

* State Variables (low to high data memory):
  - \(d(n), d(n-1), d(n-2)\)
  - \(N\)th biquad \( \rightarrow \) 1st biquad

* Entry Conditions:
  - AR1 \( \rightarrow d(n-2) \) of 1st biquad
  - AR2 \( \rightarrow -a_2 \) of 1st biquad
  - AR3 \( \rightarrow \) input sample (Q15 number)
  - AR4 \( \rightarrow \) output sample (Q15 number)
  - DP = 0, PM = 0, ARP = 3

*;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;*

BIQUAD: ;Setup variables
  ZPR ;Clear P register
  LACC *,15,AR1 ;Get Q15 input
  SPLK #2,INDX ;Setup index register
  SPLK #N–1,BRCR ;Setup count

;Begin computation;
  RPTB ELOOP–1 ;Repeat for \( N \) biquads

LOOP:
  LT *–,AR2 ;\( T = d(n-2) \)
  MPYA *+,AR1 ;Acc = \( x(n) \), \( P = -d(n-2)a_2 \)
  LTA *–,AR2 ;Acc += -d(n-2)a_2, \( T = d(n-1) \)
  MPY ** ;\( P = -d(n-1)a_1 \)
  LTA *–,AR1 ;Acc += -d(n-1)a_1, \( T = b_2 \)
  SACH *0+,1 ;Save \( d(n) \)
  MPY *– ;\( P = d(n-2)b_2 \)
  LACL #0 ;Acc = 0
  LTD *–,AR2 ;\( T = d(n-1), d(n-2) = d(n-1) \)
  MPY **,AR1 ;Acc += d(n-2)b_2, \( P = d(n-1)b_1 \)
  LTD *–,AR2 ;\( T = d(n), d(n-1) = d(n) \)
  MPY **,AR1 ;Acc += d(n-1)b_1, \( P = d(n)b_0 \)

ELOOP:
  LTA *+,AR4 ;Final accumulation
  SACH *,1 ;Save output in Q15 format

2-54
2.10.4 Dynamic Programming

Dynamic programming techniques are widely used in optimal search algorithms. Applications such as speech recognition, telecommunications, and robotics use dynamic programming algorithms. The 'C5x devices have an enhanced instruction set for efficient implementation of dynamic programming methods.

Most real-time search algorithms use the basic dynamic programming principle that the final optimal path from the start state to the goal state always passes through an optimal path from the start state to an intermediate state. Identifying intermediate paths reduces a long, time-consuming search to the final goal. An integral part of any optimal search scheme based on the dynamic programming principle is the backtracking operation. The backtracking is necessary to retrace the optimal path when the goal state is reached.

Example 2–31 on page 2-56 shows an implementation of the backtracking algorithm in which the path history consists of four independent path traces for N time periods. This path history is stored in a circular buffer. After each backtracking operation, the path history is updated by a search algorithm (not shown) for the next time period. The path history buffer is shown in Figure 2–7 for N equal to 4. Each group of four consecutive memory locations in the buffer corresponds to the expansion of the four paths by one node (or by one time period). Each element of a group corresponds to one of the four states in that time period. In addition, each element of a group points to an element in the previous time period that belongs to that path.

Using the path history buffer shown in Figure 2–7, the element corresponding to state #0 at the current time period contains a 1. This points to the second element of the previous time period that contains a 0. In this way, beginning from the current time period and using pointers to step back in time, this path is traced back as 1–0–2–1. Note that this simplified backtracking approach is taken here to illustrate 'C5x programming techniques. Most real applications would require more complex backtracking algorithms.

* Backtracking Example
* This program back-tracks the optimal path expanded by a dynamic programming
* algorithm. The path history consists of four paths expanded N times.
* It is set up as a circular buffer of length N*4. Note that decrement
* type circular buffer is used. The start and end address of the circular
* buffer are initialized this way because of two reasons:
  1- to avoid skipping the end-address of circ buffer
  2- to ensure that wrap-around is complete before next iteration.

```
LAR AR0,#BUFFER ;Get buffer address
LMMR INDEX,PATH ;Get the selected path [0..3]
SPLK #N-1,BRCR ;Trace back N time periods
* init. AR0 as pointer to circular buffer#1; length=N*4 words
SPLK #BUFFER+(N-1)*4,CBSR1
SPLK #BUFFER-3,CBER1
SPLK #08h,CBCR

* RPTB TLOOP-1 ;For i=0,i<N,i++
MAR *0+ ;Offset by state#
LACC *0- ;Get next pointer & reset to state#0
SAMM INDEX ;Save next state#
SBRK 3 ;Decrement AR0 to avoid skipping CBER1
SBRK 1 ;Now AR0 is correctly positioned 1 time
TLOOP:
  ;period back (circular addressing)
```

Figure 2–7. Backtracking With Path History

```
<table>
<thead>
<tr>
<th>Buffer</th>
<th>Buffer + 15</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
```

Path history circular buffer (N = 4)
2.11 Fast Fourier Transforms

Fourier transforms are an important tool often used in digital signal processing systems. The purpose of the transform is to convert information from the time domain to the frequency domain. The inverse Fourier transform converts information back to the time domain from the frequency domain. Computationally efficient implementations of the Fourier transforms are known as fast Fourier transforms (FFT).

The 'C5x reduces the execution time of all FFTs by virtue of its 50-ns instruction cycle time. Also, the bit-reversed addressing mode helps reduce execution time for radix-2 FFTs. As demonstrated in Figure 2–8 and Figure 2–9, the inputs or outputs of an FFT are not in sequential order. This scrambling of data locations is a direct result of the radix-2 FFT derivation. Observation of the figures and the relationship of the input and output addressing reveal that the address indexing is in bit-reversed order, as shown in Table 2–3. As a result, either the input data sequence or the output data sequence must be scrambled in association with the execution of the FFT. In Example 2–32 on page 2-59, the input data is scrambled before the execution of the FFT algorithm so that the output is in order.

Figure 2–8. An In-Place DIT FFT With In-Order Outputs and Bit-Reversed Inputs

![Diagram of an In-Place DIT FFT]

Legend for twiddle factor: \( W_0 = \frac{1}{\sqrt{8}} \) \( W_1 = \frac{\sqrt{3}}{2} \) \( W_2 = \frac{1}{\sqrt{8}} \) \( W_3 = \frac{\sqrt{3}}{2} \)
Fast Fourier Transforms

Figure 2–9. An In-Place DIT FFT With In-Order Inputs but Bit-Reversed Outputs

![Diagram of an In-Place DIT FFT](image)

Legend for twiddle factor: \( W_0 = W_8^0, W_1 = W_8^1, W_2 = W_8^2, W_3 = W_8^3 \)

Table 2–3. Bit-Reversal Algorithm for an 8-Point Radix-2 DIT FFT

<table>
<thead>
<tr>
<th>Index</th>
<th>Bit Pattern</th>
<th>Bit-Reversed Pattern</th>
<th>Bit-Reversed Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>000</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
<td>010</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>011</td>
<td>110</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>001</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>101</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>011</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td>111</td>
<td>7</td>
</tr>
</tbody>
</table>
Example 2–32. 16-Point Radix-2 Complex FFT

```
.include "c5cx0016.asm"
.title "0016 point DIT Radix–2, Complex FFT"
.width 120
N .set 16 ;NUMBER OF POINTS FOR FFT
.mmregs
.pmstmask .set 0110b ;ndx=trm=1

******************************************************************************
*                                                                            *
*     16 – POINT COMPLEX, RADIX–2 DIF FFT WITH THE TMS320C5x / LOOPED CODE   *
*     –––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––  *
******************************************************************************

* THE PROGRAM IS BASED ON THE BOOK 'DIGITAL SIGNAL PROCESSING APPLICATIONS' *
* FROM TEXAS INSTRUMENTS P. 69. IT IS OPTIMIZED FOR THE TMS320C5x INCLUDING *
* BIT REVERSAL ADDRESSING MODE.                                           *

******************************************************************************
*                                                                            *
*    USED REGISTERS: INDX,AR1,AR2,AR3,AR4,AR5,ACCU,PREG,TREG0, PMST, BRCR    *
*                    2 Stacklevel, Block B2 for temp variables               *
*                                                                            *
*    PROGRAM MEMORY: 164 WORDS ('END' – 'FFT') WITHOUT INITIALIZATION       *
*                                                                            *
*    COEFFICIENTS : 16 BITS (Q15 Format) SCALING: 1/2^4                     *
*                                                                            *
*    PROGRAM SEQUENCE:0. INITIALIZATION FOR FFT/COEFF  ADD: 240H – 20BH *
*                     1. INPUT NEW DATA INTO 'INPUT'  ADD: 220H – 23FH *
*                     2. CALL SUBROUTINE FFT      ADD: 600H – 6A3H *
*                     2.1. BITREVERSAL FROM INPUT TO DATA ADD: 200H – 21FH *
*                     2.2. FFT WITH WORK SPACE DATA ADD: 200H – 21FH *
*                     3. OUTPUT THE RESULTS FROM DATA ADD: 200H – 21FH *
*                                                                            *
*    INPUT DATA AT ADDRESS 0220h–023fh:                                      *
*                                                                            *
*    THE DATA IS STORED IN 'INPUT' AS THE SEQUENCE: X(0),X(1),...,X(15)      *
*    Y(0),Y(1),...,Y(15)                                                    *
*                                                                            *
*    OUTPUT DATA AT ADDRESS 0200h–021fh:                                     *
*                                                                            *
*    THE DATA IS STORED IN 'DATA' AS THE SEQUENCE:                           *
*    X(0),Y(0),X(1),Y(1),... ... ,X(15),Y(15)                                *
*                                                                            *
******************************************************************************
*                                                                            *
*    THIS PROGRAM INCLUDES FOLLOWING FILE:                                   *
*                                                                            *
*    THE FILE 'TWIDDLES.Q15' CONSISTS OF TWIDDLE FACTORS IN Q15 FORMAT       *
*    THE FILE 'C5CXRAD2.MAC' macro files                                     *
*    THE FILE 'INIT–FFT.ASM' for initialization                            *
******************************************************************************
```
Example 2–32. 16-Point Radix-2 Complex FFT (Continued)

```
.include C5CXRAD2.MAC
.def TWIDLEN,FFTLEN,TEMP,WAIT,cos45
.def INIT,FFT,TWIDSTRT,TWIDEND
.def STAGE1,STAGE3,STAGE4,INPUT,DATA,TWID
;
.sect "twiddles"
; table of twiddle factors for the FFT
TWIDSTRT .set $
.include twiddles.q15
TWIDEND .set $
TWIDLEN .set TWIDEND–TWIDSTRT
*
INPUT .usect "input",N*2 ;input data array
DATA .usect "data",N*2 ;working data array
TWID .usect "twid",N*2 ;reserve space for twiddles
*
.include init-fft.asm
*
.sect "fftprogram"
*
* FFT CODE WITH BIT-REVERSED INPUT SAMPLES / ARP=AR3
*
FFT:  LAR AR3,DATAADD ;TRANSFER 32 WORDS FROM ‘input’ to ‘data’
LACC NN
SAMM INDX ;indexregister=7
RPT NN2 ;N TIMES
BLDD #INPUT,*BR0+
*
* FFT CODE for STAGES 1 and 2
*
STAGE1:  SPLK #7,INDX ;index register = 7
LAR AR1,DATAADD ;pointer to DATA r1,i1
LAR AR2,#DATA+2 ;pointer to DATA + 2 r2,i2
LAR AR3,#DATA+4 ;pointer to DATA + 4 r3,i3
LAR AR4,#DATA+6 ;pointer to DATA + 6 r4,i4
COMBOSX 4 ;repeat 4 times
*
* FFT CODE FOR STAGE 3 / ARP=AR2
*
STAGE3:  SPLK #9,INDX ;index register = 9
LAR AR1,DATAADD ;ar1 –> DATA
LAR AR2,#DATA+8 ;ar2 –> DATA+8
stage3 2 ;repeat 2 times
*
* FFT CODE FOR STAGE 4 / ARP=ARP
*
```
Example 2–32. 16-Point Radix-2 Complex FFT (Continued)

The bit-reversed addressing mode is part of the indirect addressing implemented with the auxiliary registers and the associated arithmetic unit. In this mode, a value (index) contained in INDX is either added to or subtracted from the auxiliary register being pointed to by the ARP. However, the carry bit is not propagated in the forward direction; instead, it is propagated in the reverse direction. The result is a scrambling in the address access.

The procedure for generating the bit-reversed address sequence is to load INDX with a value corresponding to one-half the length of the FFT and to load another auxiliary register—for example, AR1—with the base address of the data array. However, implementations of FFTs involve complex arithmetic; as a result, two data memory locations (one real and one imaginary) are associated with each data sample. For ease of addressing, the samples are stored in workspace memory in pairs with the real part in the even address locations and the imaginary part in the odd address locations. This means that the offset from the base address for any given sample is twice the sample index. If the incoming data is in the following form:

\[
XR(0), \; XR(1), \ldots, \; XR(7), \; XI(0), \; XI(1), \ldots, \; XI(7)
\]

where

- \(XR\) – real component of input sample
- \(XI\) – imaginary component of input sample

then it is easily transferred into the data memory and stored in the scrambled order:

\[
XR(0), \; XI(0), \; XR(4), \; XI(4), \; XR(2), \; XI(2), \ldots, \; XR(7), \; XI(7)
\]

by loading INDX register with the size of FFT and by using bit-reversed addressing to save each input word. Example 2–33 on page 2-62 shows the contents of auxiliary register AR1 when INDX is initialized with a value of 8.
Example 2–33. Bit-Reversed Addressing for an FFT

<table>
<thead>
<tr>
<th>MSB</th>
<th>LSB</th>
<th>FOR 8-POINT FFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>INDX</td>
<td>1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td>1 0 0 0</td>
</tr>
<tr>
<td>AR1</td>
<td>0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0</td>
<td>0 0 0 0</td>
</tr>
</tbody>
</table>

**AR1**

RPT 15
BLDD #INPUT, *BR0+

<table>
<thead>
<tr>
<th>AR1</th>
<th>BASE ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0</td>
<td>XR(0)</td>
</tr>
<tr>
<td>0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0</td>
<td>XR(4)</td>
</tr>
<tr>
<td>0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0</td>
<td>XR(2)</td>
</tr>
<tr>
<td>0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0</td>
<td>XR(6)</td>
</tr>
<tr>
<td>0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0</td>
<td>XR(1)</td>
</tr>
<tr>
<td>0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0</td>
<td>XR(5)</td>
</tr>
<tr>
<td>0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0</td>
<td>XR(3)</td>
</tr>
<tr>
<td>0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0</td>
<td>XR(7)</td>
</tr>
<tr>
<td>0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0</td>
<td>XI(0)</td>
</tr>
<tr>
<td>0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0</td>
<td>XI(4)</td>
</tr>
<tr>
<td>0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0</td>
<td>XI(2)</td>
</tr>
<tr>
<td>0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0</td>
<td>XI(6)</td>
</tr>
<tr>
<td>0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0</td>
<td>XI(1)</td>
</tr>
<tr>
<td>0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0</td>
<td>XI(5)</td>
</tr>
<tr>
<td>0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0</td>
<td>XI(3)</td>
</tr>
<tr>
<td>0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0</td>
<td>XI(7)</td>
</tr>
</tbody>
</table>

Example 2–34 on page 2-63 provides macros for a 16-point FFT. Example 2–35 on page 2-69 provides an initialization routine.
Example 2–34. Macros for 16-Point DIT FFT

********************************************************************************
* FILE: c5cxrad2.mac ––> macro file for radix 2 fft’s based on 320c5x *
* *
* COPYRIGHT TEXAS INSTRUMENTS INC. 1990 *
********************************************************************************
* *
* MACRO ’COMBO2x’ FOR THE COMPLEX, RADIX-2 DIT FFT *
* *
* ORGANIZATION OF THE INPUT DATA MEMORY: R1,I1,R2,I2,R3,I3,R4,I4 *
*
********************************************************************************
* *
* THE MACRO ’COMBO2x’ PERFORMS FOLLOWING CALCULATIONS: *
* *
* R1 := [(R1+R2)+(R3+R4)]/4 INPUT OUTPUT *
* R2 := [(R1–R2)+(I3–I4)]/4 –––––––––––––––––– –––––––––––––––––– *
* R3 := [(R1+R2)–(R3+R4)]/4 AR0 = 7 *
* R4 := [(R1–R2)–(I3–I4)]/4 AR1 –> R1,I1 AR1 –> R5,I5 *
* I1 := [(I1+I2)+(I3+I4)]/4 AR2 –> R2,I2 AR2 –> R6,I6 *
* I3 := [(I1+I2)–(I3+I4)]/4 AR4 –> R4,I4 AR4 –> R8,I8 *
* I4 := [(I1–I2)+(R3–R4)]/4 *
 *
* For a 16-point Radix 2 complex FFT the Macro ’COMBO2x’ has to be *
* repeated N/4 times (e.g. 4 times for a 16 point FFT). *
*
********************************************************************************
COMBO5x $MACRO num ; REPEAT MACRO ’COMBO5x’: N/4 times
SPLK #:num:-1,BCR ; execute ’num’ times ’COMBO5x’
*
RPTB comboend ;
*
LACC *,14,AR4 ;ACC :=(R3)/4 4 R1 R2 R3 R4 T1
SUB *,14,AR5 ;ACC :=(R3–R4)/4 5 R1 R2 R3 R4 T1
SACH *+,1,AR4 ;T1  =(R3–R4)/2 4 R1 R2 I3 R4 T2
*
ADD *+,15,AR5 ;ACC :=R3+R4)/4 5 R1 R2 R3 I4 T2
SACH *+,1,AR2 ;T2  =(R3+R4)/2 2 R1 R2 R3 I4 T2
*
ADD *,14,AR1 ;ACC :=(R2+R3+R4)/4 1 R1 R2 R3 I4 T2
ADD *,14 ;ACC :=(R1+R2+R3+R4)/4 1 R1 R2 R3 I4 T2
SACH *,0,AR5 ;R1 :=(R1+R2+R3+R4)/4 5 I1 R2 R3 I4 T2
SUB *,16,AR3 ;ACC :=(R1+R2–(R3+R4))/4 3 I1 R2 R3 I4 T2
SACH *,0,AR5 ;R3 :=(R1+R2–(R3+R4))/4 5 I1 R2 I3 I4 T2

Software Applications
Example 2–34. Macros for 16-Point DIT FFT (Continued)

```
ADD *,15,AR2 ;ACC :=(R1+R2)/4 2 I1 R2 I3 I4 T2
SUB *,15,AR3 ;ACC :=(R1–R2)/4 3 I1 R2 I3 I4 T2
ADD *,14,AR4 ;ACC :=((R1–R2)+(I3))/4 4 I1 R2 I3 I4 T2
SUB *,14,AR2 ;ACC :=((R1–R2)+(I3–I4))/4 2 I1 R2 I3 I4 T2
SACH *+,0,AR4 ;R2 :=((R1–R2)+(I3–I4))/4 4 I1 I2 I3 I4 T2
ADD *,14,AR3 ;ACC :=((R1–R2)+I3+I4)/4 3 I1 R2 I3 I4 T2
SACH *+,0,AR1 ;R4 :=((R1–R2)–(I3–I4))/4 1 I1 I2 I3 I4 T2
LACC *,14,AR2 ;ACC :=(I1)/4 2 I1 I2 I3 I4 T2
SUB *,14,AR5 ;ACC :=(I1–I2)/4 5 I1 I2 I3 I4 T2
SACH *+,0,AR3 ;T2 :=(I1–I2)/2 2 I1 I2 I3 I4 T2
ADD *,15,AR3 ;ACC :=((I1+I2))/4 4 I1 I2 I3 I4 T2
ADD *,14,AR4 ;ACC :=((I1+I2)+(I3))/4 4 I1 I2 I3 I4 T2
ADD *,14,AR1 ;ACC :=((I1+I2)+(I3+I4))/4 3 R5 I2 I3 I4 T2
SACH *0+,0,AR3 ;I1 :=((I1+I2)+(I3+I4))/4 3 R5 I2 I3 I4 T2
ADD *,15,AR4 ;ACC :=((I1+I2)–(I3+I4))/4 4 R5 I2 I3 I4 T2
SACH *0+,0,AR5 ;I3 :=((I1+I2)–(I3+I4))/4 5 R5 I2 I3 I4 T2
LACC *–,15 ;ACC :=(I1–I2)/4 5 R5 I2 R7 I4 T1
SUB *,15,AR2 ;ACC :=((I1–I2)–(R3–R4))/4 2 R5 I2 R7 I4 T1
SACH *+,0,AR5 ;I2 :=((I1–I2)–(R3–R4))/4 5 R5 R6 R7 I4 T1
ADD *,16,AR4 ;ACC :=((I1–I2)+(R3–R4))/4 4 R5 R6 R7 I4 T1
comboend:
SACH *0+,0,AR3 ;I4 :=((I1–I2)+(R3–R4))/4 3 R5 R6 R7 R8 T1
MAR *,AR2 ;ARP=AR2
$ENDM
```

*** MACRO ’ZEROI’ number of words : 10 ***

```
LACC *,15,AR1 ;ACC :=(1/2)(QR) PR QR 1
ADD *,15,AR2 ;ACC :=(1/2)(PR+QR) PR QR 1
SACH *+,0,AR2 ;PR :=(1/2)(PR+QR) PI QR 2
SUB *,16,AR4 ;ACC :=(1/2)(PR+QR)–(QR) PI QR 2
SACH *+ ;QR :=(1/2)(PR–QR) PI QI 2
```

---

2-64
Example 2–34. Macros for 16-Point DIT FFT (Continued)

* LACC *,15,AR1 ; ACC :=(1/2)(QI)   PI  QI  1
  ADD *,15 ; ACC :=(1/2)(PI+QI)   PI  QI  1
  SACH *,0,AR2 ; PI :=(1/2)(PI+QI) PR+1 QI  2
  SUB *,16 ; ACC :=(1/2)(PI+QI)-(QI) PR+1 QI  2
  SACH *+ ; QI :=(1/2)(PI-QI)    PR+1 QR+1  2
$ENDM

*********************************************************************************
* MACRO 'PBY2I' number of words: 12 *
* PR'=(PR+QI)/2 PI'=(PI-QR)/2 *
* QR'=(PR-QI)/2 QI'=(PI+QR)/2 *
*********************************************************************************

PBY2I $MACRO ; AR1 AR2 ARP
* ––– ––– –––
  LACC *,15,AR5 ; PR  QI  5
  SACH *,1,AR2 ; TMP=QR  PR  QI  2
  LACC *,15,AR1 ; ACC :=QI/2   PR  QI  1
  ADD *,15 ; ACC :=(PR+QI)/2  PR  QI  1
  SACH *,0,AR2 ; PR :=(PR+QI)/2 PI  QI  2
  SUB *,16 ; ACC :=(PR-QI)/2  PI  QR  2
  SACH *+ ,0,AR1 ; QR :=(PR-QI)/2 PI  QI  1
  LACC *,15,AR5 ; ACC :=PI/2   PI  QI  5
  SUB *,15,AR1 ; ACC :=(PI-QR)/2 PI  QI  1
  SACH *,0,AR5 ; PI :=(PI-QR)/2 PR+1 QI  5
  ADD *,16,AR2 ; ACC :=(PI+QR)/2 PR+1 QI  2
  SACH *+ ; QI :=(PI+QR)/2    PR+1 QR+1  2
$ENDM

*********************************************************************************
* MACRO 'PBY4J' number of words: 16 *
* T=SIN(45)=COS(45)=W45 *
* PR'=(PR+(W*QI+W*QR)) = PR + W * QI + W * QR (<- AR1) *
* QR'=(PR-(W*QI+W*QR)) = PR - W * QI - W * QR (<- AR2) *
* PI'=(PI+(W*QI-W*QR)) = PI + W * QI - W * QR (<- AR1+1) *
* QI'=(PI-(W*QI-W*QR)) = PI - W * QI + W * QR (<- AR1+2) *
*********************************************************************************
Example 2–34. Macros for 16-Point DIT FFT (Continued)

```
PBY4J $MACRO ;TREG =W AR5 PREG AR1 AR2 ARP
* MPY *,AR5 ;PREG =W*QR/2 – W*QR/2 PR QI 5
SPH *,AR1 ;TMP =W*QR/2 W*QR/2 W*QR/2 PR QI 1
LACC *,15,AR2 ;ACC =PR/2 W*QR/2 W*QR/2 PR QI 2
MPYS *– ;ACC = (PR-W*QR)/2 W*QR/2 W*QR/2 PR QI 2
SPAC ;ACC = (PR-W*QI-W*QR)/2 W*QR/2 W*QR/2 PR QI 2
SACH *,0,AR1 ;QR = (PR-W*QI-W*QR)/2 W*QR/2 W*QR/2 PR QI 1
SUB *,16 ;ACC = (PR-W*QI-W*QR)/2 W*QR/2 W*QR/2 PR QI 1
NEG ;ACC = (PR-W*QI-W*QR)/2 W*QR/2 W*QR/2 PR QI 1
SACH *+ ;QR = (PR+QI+QQR)/2 W*QR/2 W*QR/2 PI QI 1
*
LACC *,15,AR5 ;ACC = (PR)/2 W*QR/2 W*QR/2 PI QI 5
SPAC ;ACC = (PI-W*QI)/2 W*QR/2 – PI QI 5
ADD *,16,AR2 ;ACC = (PI-W*QI-W*QR)/2 – – PI QI 5
SACH *,0,AR1 ;PI = (PI-W*QI+W*QR)/2 – – PI QI 5
SUB *,16 ;ACC1 = (PI-W*QI+W*QR)/2 – – PI QI 5
NEG ;ACC1 = (PI-W*QI+W*QR)/2 – – PI QI 5
SACH *+ ;PI = (PI+W*QI+W*QR)/2 – – PI QI 5
$ENDM
```

*********************************************************************************
* *
* MACRO ‘P3BY4J’ number of words: 16 *
* *
* ENTRANCE IN THE MACRO: ARP=AR2 *
* AR1->PR,PI *
* AR2->QR,QI *
* TREG=W=COS(45)=SIN(45) *
* *
* PR’ = PR + (W*QI – W*QR) = PR + W * QI – W * QR (<– AR1) *
* QR’ = PR – (W*QI – W*QR) = PR – W * QI + W * QR (<– AR2) *
* PI’ = PI – (W*QI + W*QR) = PI – W * QI – W * QR (<– AR1+1) *
* QI’ = PI + (W*QI + W*QR) = PI + W * QI + W * QR (<– AR1+2) *
* *
* EXIT OF THE MACRO: ARP=AR2 *
* AR1->PR+1,PI+1 *
* AR2->QR+1,QI+1 *
* *
**********************************************************************************
Example 2–34. Macros for 16-Point DIT FFT (Continued)

* LACC *,15,AR5 ;ACC = (PI)/2  W*QR/2  W*QI/2  PI  QI  5
  APAC ;ACC = (PI+W*QI)/2  W*QR/2  –  PI  QI  5
  ADD *,16,AR2 ;ACC = (PI+W*QI+W*QR)/2  –  –  PI  QI  2
  SACH *0+,0,AR1 ;QI’ = -(PI+W*QI+W*QR)/2  –  –  PI  QR5  1
  SUB *,16 ;ACCU = (–PI+W*QI+W*QR)/2  –  –  PI  QR5  1
  NEG ;ACCU = (PI–W*QI–W*QR)/2  –  –  PI  QR5  1
  SACH *0+,0,AR2 ;PI’ = (PI–W*QI–W*QR)/2  –  –  PR5  QR5  2

* MACRO ‘stage3’ number of words: 54
  *
  stage3 $macro num
  SPLK #:num:–1,BRCR ;execute ‘num’–1 times ‘stage3’
  LT cos45
  RPTB stage3e
  ZEROI
  PBY4J
  PBY2I
  P3BY4j
  stage3e: .set $–1

* MACRO: ‘BUTTFLYI’ general butterfly radix 2 for 320C5x
  *
  THE MACRO ‘BUTTFLYI’ REQUIRES 18 WORDS
  *
  Definition: ARP -> AR2 (input) ARP -> AR2 (output)
  *
  Definition: AR1 -> QR (input) AR1 -> QR+1 (output)
  *
  Definition: AR2 -> PR (input) AR2 -> PR+1 (output)
  *
  Definition: AR3 -> Cxxx (input) AR3 -> Cxxx+1 (output) -- WR=cosine
  *
  Definition: AR4 -> Sxxx (input) AR4 -> Sxxx+1 (output) -- WI=sine
  *
  Definition: AR5 -> temporary variable (unchanged)
  *
  uses index register
  *
  PR’ = (PR+(QR*WR+QI*WI))/2  WR=cos(W) WI=sin(W)
  *
  PI’ = (PI+(QI*WR–QR*WI))/2
  *
  QI’ = (PI–(QR*WR–QI*WI))/2

*********************************************************************************
Example 2–34. Macros for 16-Point DIT FFT (Continued)

BUTTFLYI $MACRO
* (contents of register after exec.)

RPTB btflyend ; TREG :=QR
LT *+,AR3 ; TREG :=QR PR QI C S 3
MPY *,AR2 ; PREG :=QR*WR/2 QR PR QI C S 2
LTP *-,AR4 ; ACC :=QR*WR/2 QI PR QR C S 4
MPY *,AR3 ; PREG :=QI*WI/2 QI PR QR C S 3
MPYA *,AR2 ; ACC :=(QR*WR+QI*WI)/2 QR PR QR C+1 S 2
; PREG :=QI*WR
LT *,AR5 ; TREG :=QR QR PR QI C+1 S 5
SACH *,1,AR1 ; H0 :=(QR*WR+QI*WI) QR PR QR C+1 S 1

ADD *,15 ; ACC :=(PR+(QR*WR+QI*WI))/2 QR PR QR C+1 S 1
SACH *,0,AR5 ; PR :=(PR+(QR*WR+QI*WI))/2 QR PI QR C+1 S 5
SUB *,16,AR2 ; ACC :=(PR–(QR*WR+QI*WI))/2 QR PI QR C+1 S 2
SACH *,0,AR1 ; QR :=(PR–(QR*WR+QI*WI))/2 QR PI QI C+1 S 1

LACC *,15,AR4 ; ACC :=(PI/PREG=QI*WR) QI PI QI C+1 S 4
MPYS *,AR2 ; PREG :=QR*WI/2 QI PI QI C+1 S+1 2
; ACC :=(PI–QI*WR)/2
APAC ; ACC :=(PI–QI*WR–QR*WI))/2 QI PI QI C+1 S+1 2
SACH *,0,AR1 ; QI :=(PI–(QI*WR–QR*WI))/2 QI PI QR+1 C+1 S+1 1
NEG ; ACC :=(PI–(QI*WR–QR*WI))/2 QI PI QR+1 C+1 S+1 1
ADD *,16 ; ACC :=(PI+(QI*WR–QR*WI))/2 QI PI QR+1 C+1 S+1 1
btflyend:
SACH *,0,AR2 ; PI :=(PI+(QI*WR–QR*WI))/2 QI PR+1 QR+1 C+1 S+1 2
$ENDM
; end of file
Example 2–35. Initialization Routine

* file: INIT-FFT.ASM
* Initialized variables
* .bss NN,1 ;Number of fft-points
.bss NN2,1 ;2*N-1
.bss DATAADD,1 ;Start address of data
.bss cos45,1
.bss sin4,1 ;Start of sine in stage 4
.bss cos4,1 ;Start of cosine in stage 4
* Temp variables
* .bss TEMP,2 ;Used for temporary numbers
* .sect "vectors"
B INIT,*,AR0
.sect "init"
TABINIT: .word N,N-1,2*N-1,DATA
.word 5A82h ;cos(45)=sin(45)
.word TWID,TWID+4
TABEND: .set $

* INIT: LDP #0 ;Use only B2 and mmregs for direct addressing
SPM 0 ;No shift from PREG to ALU
CLRC OVM ;Disable overflowmode
SETC SXM ;Enable sign extension mode
SPLK #pmstmask,PMST ;ndx=trm=1
* INIT Block B2
* LAR AR0,#NN ;ARP is already pointing to AR0
LACC #TABINIT
RPT #TABEND-TABINIT
TBLR *+
* INIT TWIDDLE FACTORS
* LAR AR0,#TWID ;ARP is already pointing to AR0
LACC #TWIDSTRT
RPT #TWIDLEN
TBLR *+
* EXECUTE THE FFT
* LAR AR5,#TEMP ;Pointer to 2 temp register
CALL FFT,*,AR3 ;ARP=AR3 FOR MACRO COMBO
* WAIT RET ;Return
Chapter 3

External Memory Interface

This chapter provides a general description of the external interface to memory. Also included is a description of the direct memory access (DMA) in a portable computer configuration.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 External Interface to Program Memory</td>
<td>3-2</td>
</tr>
<tr>
<td>3.2 External Interface to Local Data Memory</td>
<td>3-4</td>
</tr>
<tr>
<td>3.3 External Interface to Global Data Memory</td>
<td>3-6</td>
</tr>
<tr>
<td>3.4 External Interface to I/O Space</td>
<td>3-6</td>
</tr>
<tr>
<td>3.5 Direct Memory Access (DMA) in a Personal Computer</td>
<td>3-7</td>
</tr>
<tr>
<td>Configuration</td>
<td></td>
</tr>
</tbody>
</table>
3.1 External Interface to Program Memory

The 'C5x devices can address up to 64K words of program memory off-chip. The following key signals interface to off-chip memory:

- **A0–A15**: 16-bit bidirectional address bus
- **D0–D15**: 16-bit bidirectional data bus
- **PS**: Program memory select
- **STRB**: External memory access active strobe
- **RD**: Read select (external device output enable)
- **WE**: Write enable
- **IACK**: Interrupt acknowledge
- **READY**: Memory ready to complete cycle
- **HOLD**: Request for control of memory interface
- **HOLDA**: Acknowledge HOLD request
- **BR**: Bus request
- **IAQ**: Acknowledge bus request (when HOLDA is low)

In the example of an external EPROM interface shown in Figure 3–1, the 'C5x device interfaces to an external 8K-byte \( \times 8 \)-bit EPROM. The use of 8-bit-wide memories saves power, board space, and cost over 16-bit-wide memory banks. The 16-bit-wide memory banks can be used with the same basic interface as the 8-bit-wide memories. Note that the 'C5x cannot directly execute code from 8-bit-wide memory. An on-chip program (such as a boot-loader program) is required to read 8-bit-wide memory to form 16-bit long instruction words and transfer them to on-chip RAM.

The program select (PS) signal is connected directly to the chip select (CS) pin to select the EPROM on any external program access. The EPROM is addressed in any 8K-word address block in program space. If you want to interface multiple blocks of memory in program space, you can use a decode circuit that gates the PS signal and the appropriate address bits to drive the memory block chip selects.

The read select (RD) signal is connected directly to the output enable (OE) pin of the EPROM. The OE signal enables the output drivers of the EPROM. The drivers are turned off in time to prevent data bus conflicts with an external write by the 'C5x device.
Figure 3–1.  ’C5x Interfacing to External EPROM
3.2 External Interface to Local Data Memory

The 'C5x devices can address up to 64K words of local data memory off-chip. The following key signals interface to off-chip memory:

- **A0–A15**: 16-bit bidirectional address bus
- **D0–D15**: 16-bit bidirectional data bus
- **DS**: Data memory select
- **STRB**: External memory access active strobe
- **RD**: Read select (external device output enable)
- **WE**: Write enable
- **READY**: Memory ready to complete cycle
- **HOLD**: Request for control of memory interface
- **HOLDA**: Acknowledge HOLD request
- **BR**: Bus request
- **IAQ**: Acknowledge bus request (when HOLDA is low)

In the example of an external RAM interface shown in Figure 3–2, the 'C5x device interfaces to four 16K-byte \( \times 4 \)-bit RAM devices. The data memory select (DS) signal is connected directly to the chip select (CS) of the devices. This allows the external RAM block to be addressed in any of the four 16K-byte banks of local data space. If there are additional banks of off-chip data memory, you can use a decode circuit that gates the DS signal with the appropriate address bits to drive the memory block chip select.

The RD signal is connected directly to the output enable (OE) pin of the RAMs. The OE signal enables the output drivers of the RAM. The drivers are turned off in time to prevent data bus conflicts with an external write by the 'C5x device. If the RAM device does not have an OE pin, then the DS signal must be gated with STRB and connected to the CS pin of the RAM to implement the same function.

The WE signal is connected directly to the WE pin of the RAMs. The 'C5x device requires at least two cycles on all external writes, including a half cycle before WE goes low and a half cycle after WE goes high; this prevents buffer conflicts on the external buses. Additional write cycles can be obtained by modifying the software wait-state generator registers.
Figure 3–2. 'C5x Interfacing to External RAM
3.3 External Interface to Global Data Memory

Global memory can be used in digital signal processing tasks, such as filters or modems, in which the algorithm being implemented is divided into sections with a distinct processor dedicated to each section. With multiple processors dedicated to distinct sections of the algorithm, throughput may be increased via pipelined execution. Figure 3–3 illustrates an example of a global memory interface. Since the processors can be synchronized by using the BR pin, the arbitration logic can be simplified and the address and data bus transfers made more efficient.

The global memory interface can also extend the data memory address map beyond the reach of the 16-bit address bus by paging in an additional 32K words. Loading the GREG with the appropriate value can overlay the local data memory with additional memory, starting at the highest memory address (FFFFh) and moving down. This additional memory is differentiated from local memory accesses by the BR pin going low. The rest of the memory interface control signals (STRB, DS, etc.) behave identically on a local or global data access.

Figure 3–3. Global Memory Interface

3.4 External Interface to I/O Space

The RD and WE signals can be used along with chip-select logic to output data to an external device. The port address can be decoded and used as a chip select for the input or output device.
3.5 Direct Memory Access (DMA) in a Personal Computer Configuration

You can implement DMA in a personal computer (PC) environment by using the PC system bus for data transfer to external 'C5x memory. Figure 3–4 illustrates a DMA access in a PC environment. In this configuration, either the master CPU or a disk controller places data onto the PC system bus, which can be downloaded into the local memory of the 'C5x. In this configuration, the 'C5x functions like a peripheral processor with multifunction capability. In a speech application, for example, the master CPU can load the 'C5x program memory with algorithms to perform such tasks as speech analysis, synthesis, or recognition, and can fill the 'C5x data memory with the required speech templates. In another application, the 'C5x can serve as a dedicated graphics engine. Programs can be downloaded via the PC system bus into program RAM. Data can come from PC disk storage or can be provided directly by the master CPU.

In Figure 3–4, decode/arbitration logic control the DMA. When the address on the PC system bus resides in the local memory of the peripheral 'C5x, the logic control asserts the HOLD signal of the 'C5x while sending the master CPU a not-ready indication to allow wait states. After the 'C5x acknowledges the direct memory access by asserting HOLDA, READY is asserted and the information is transferred.

Figure 3–4. Direct Memory Access in a PC Environment
Texas Instruments offers many products for total system solutions, including memory options, data acquisition, and analog input/output devices. This chapter describes a variety of devices that interface directly to the TMS320 DSPs in rapidly expanding applications.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Multimedia Applications</td>
<td>4-2</td>
</tr>
<tr>
<td>4.2 Telecommunications Applications</td>
<td>4-5</td>
</tr>
<tr>
<td>4.3 Dedicated Speech Synthesis Applications</td>
<td>4-10</td>
</tr>
<tr>
<td>4.4 Servo Control/Disk Drive Applications</td>
<td>4-12</td>
</tr>
<tr>
<td>4.5 Modem Applications</td>
<td>4-15</td>
</tr>
<tr>
<td>4.6 Advanced Digital Electronics Applications for Consumers</td>
<td>4-18</td>
</tr>
</tbody>
</table>
4.1 Multimedia Applications

Multimedia applications integrate different media through a centralized computer. These media can be visual or audio and can be input to or output from the central computer via a number of technologies. The technologies can be digital or analog based (such as audio or video tape recorders). The integration and interaction of media enhances the transfer of information and can accommodate both analysis of problems and synthesis of solutions.

Figure 4–1 shows both the central role of the multimedia computer and its ability to integrate the various media to optimize information flow and processing.

4.1.1 System Design Considerations

Multimedia systems can include various grades of audio and video quality. The most popular video standard currently used (VGA) covers 640 × 480 pixels with 1, 2, 4, and 8-bit memory-mapped color. Also, 24-bit true color is supported, and 1024 × 768 (beyond VGA) resolution has emerged. There are two grades of audio. The lower grade accommodates 11.25-kHz sampling for 8-bit monaural systems, while the higher grade accommodates 44.1-kHz sampling for 16-bit stereo.

Audio specifications include a musical instrument digital interface (MIDI) with compression capability, which is based on keystroke encoding, and an input/output port with a 3-disc voice synthesizer. In the media control area, video disc, CD audio, and CD ROM player interfaces are included. Figure 4–2 shows a multimedia subsystem.

Figure 4–1. System Block Diagram
The TLC32047 wide-band analog interface circuit (AIC) is well suited for multimedia applications because it features wide-band audio and up to 25-kHz sampling rates. The TLC32047 is a complete analog-to-digital and digital-to-analog interface system for the TMS320 DSPs. The nominal bandwidths of the filters accommodate 11.4 kHz, and this bandwidth is programmable. The application circuit shown in Figure 4–2 handles both speech encoding and modem communication functions, which are associated with multimedia applications.

Figure 4–3 shows how the 'C25 DSP interfaces to the TLC32047 AIC — comprising the building blocks of the 9600-bps V.32bis modem shown in Figure 4–2.
4.1.2 Multimedia-Related Devices

As listed in Table 4–1 and Table 4–2, TI provides a complete array of analog and graphics interface devices. These devices support the TMS320 DSPs for complete multimedia solutions. For application assistance or additional information, call the Semiconductor Product Information Center (PIC) as listed in If You Need Assistance on page xvii.

Table 4–1. Data Converter ICs

<table>
<thead>
<tr>
<th>Device</th>
<th>Description</th>
<th>I/O</th>
<th>Resolution (Bits)</th>
<th>Conversion CLK Rate</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLC320AC01</td>
<td>Analog interface (5 V only)</td>
<td>Serial</td>
<td>14</td>
<td>43.2 kHz</td>
<td>Portable modem and speech, multimedia</td>
</tr>
<tr>
<td>TLC32040</td>
<td>Analog interface (AIC)</td>
<td>Serial</td>
<td>14</td>
<td>19.2 kHz</td>
<td>Speech and modems</td>
</tr>
<tr>
<td>TLC32044</td>
<td>Analog interface (AIC)</td>
<td>Serial</td>
<td>14</td>
<td>19.2 kHz</td>
<td>Speech and modems</td>
</tr>
<tr>
<td>TLC32046</td>
<td>Analog interface (AIC)</td>
<td>Serial</td>
<td>14</td>
<td>25 kHz</td>
<td>Speech and modems</td>
</tr>
<tr>
<td>TLC32047</td>
<td>Analog interface (11.4-kHz BW) (AIC)</td>
<td>Serial</td>
<td>14</td>
<td>25 kHz</td>
<td>Speech, modem, and multimedia</td>
</tr>
<tr>
<td>TLC32071</td>
<td>Analog interface (AIC)</td>
<td>Parallel</td>
<td>8</td>
<td>1 MHz</td>
<td>Servo control/disk drive</td>
</tr>
<tr>
<td>TLC34058</td>
<td>Video palette</td>
<td>Parallel</td>
<td>Triple 8</td>
<td>135 MHz</td>
<td>Graphics</td>
</tr>
<tr>
<td>TLC34075/6</td>
<td>Video palette</td>
<td>Parallel</td>
<td>Triple 8</td>
<td>135 MHz</td>
<td>Graphics</td>
</tr>
<tr>
<td>TLC5501</td>
<td>Flash ADC</td>
<td>Parallel</td>
<td>6</td>
<td>20 MHz</td>
<td>Video</td>
</tr>
<tr>
<td>TLC5502/3</td>
<td>Flash ADC</td>
<td>Parallel</td>
<td>8</td>
<td>20 MHz</td>
<td>Video</td>
</tr>
<tr>
<td>TLC5601</td>
<td>Video DAC</td>
<td>Parallel</td>
<td>6</td>
<td>20 MHz</td>
<td>Video</td>
</tr>
<tr>
<td>TLC5602</td>
<td>Video DAC</td>
<td>Parallel</td>
<td>8</td>
<td>20 MHz</td>
<td>Video</td>
</tr>
<tr>
<td>TLC1550/1</td>
<td>ADC</td>
<td>Parallel</td>
<td>10</td>
<td>150 kHz</td>
<td>Servo control/speech</td>
</tr>
<tr>
<td>TMS57013/4</td>
<td>Dual audio DAC + digital filter</td>
<td>Serial</td>
<td>16/18</td>
<td>32, 37.8, 44.1, 48 kHz</td>
<td>Digital audio</td>
</tr>
</tbody>
</table>

Table 4–2. Switched-Capacitor Filter ICs

<table>
<thead>
<tr>
<th>Device</th>
<th>Function</th>
<th>Order</th>
<th>Roll-Off</th>
<th>Power Out</th>
<th>Power Down</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLC2470</td>
<td>Differential audio filter amplifier</td>
<td>4</td>
<td>5 kHz</td>
<td>500 mW</td>
<td>Yes</td>
</tr>
<tr>
<td>TLC2471</td>
<td>Differential audio filter amplifier</td>
<td>4</td>
<td>3.5 kHz</td>
<td>500 mW</td>
<td>Yes</td>
</tr>
<tr>
<td>TLC10/20</td>
<td>General-purpose dual filter</td>
<td>2</td>
<td>CLK ÷ 50</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>TLC04/14</td>
<td>Low pass, Butterworth filter</td>
<td>4</td>
<td>CLK ÷ 50</td>
<td>N/A</td>
<td>No</td>
</tr>
</tbody>
</table>
4.2 Telecommunications Applications

The TI linear product line focuses on three primary telecommunications application areas: subscriber instruments (telephones, modems, etc.), central office line card products, and personal communications. Subscriber instruments include the TCM508x dual-tone multiple frequency (DTMF) encoder family, the TCM150x tone ringer family, the TCM1520 ring detector, and the TCM3105 frequency shift keying (FSK) modem. Central office line card products include the TCM29Cxx combo (combined PCM filter plus codec) family, the TCM420x subscriber-line control circuit family, and the TCM1030/60 line card transient protector. Personal communication (PCN) and cellular products include the TCM320AC3x family of 5-V voice-band audio processors (VBAP).

TI continues to develop new telecom integrated circuits, such as a high-performance 3-volt combo family for personal communications applications, and a radio frequency (RF) power amplifier family for hand-held and mobile cellular phones.

Figure 4–4 shows a block diagram of a generic telecom application using a DSP with analog-to-digital converters (ADCs) and digital-to-analog converters (DACs). Figure 4–5 illustrates a general telecom application.
4.2.1 System Design Considerations

The size, network complexity, and compatibility requirements of telecommunications central office systems create demanding performance requirements. Combo voice-band filter performance is typically ±0.15 dB in the passband. Idle channel noise must be on the order of 15 dBrnc0. Gain tracking (S/Q) and distortion must also meet stringent requirements. The key parameters for a subscriber line interface circuit (SLIC) device are gain, longitudinal balance, and return loss.

The TCM320AC36 combo interfaces directly to the ‘C25 serial port with a minimum of external components, as shown in Figure 4–6. Half of hex inverter U3 and crystal Y1 form an oscillator that provides clock timing to the TCM320AC36. The synchronous 4-bit counters U1 and U2 generate an 8-kHz frame sync signal. DCLKR on the TCM320AC36 is connected to VDD, placing the combo in fixed data-rate mode. Two 20-kΩ resistors connected to ANLGIN and MIC_GS set the gain of the analog input amplifier to 1. The timing is shown in Figure 4–7.
Figure 4–6. Typical DSP/Combo Interface
4.2.2 Telecommunications-Related Devices

As listed in Table 4–2, Table 4–3, and Table 4–4, TI provides a complete array of devices. These devices support the TMS320 DSPs for complete telecommunications solutions. Data sheets for the telecom devices listed in Table 4–3 and Table 4–4 are available. To request your copy, for application assistance, or additional information, call the Semiconductor Product Information Center (PIC) as listed in If You Need Assistance on page xvii.
Table 4–3. Telecom Devices—Codec/Filter ICs

<table>
<thead>
<tr>
<th>Device Number</th>
<th>Coding Law</th>
<th>Clock Rates (MHz) †</th>
<th># of Bits</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCM29C13</td>
<td>A and μ</td>
<td>1.544, 1.536, 2.048</td>
<td>8</td>
<td>C.O. and PBX line cards</td>
</tr>
<tr>
<td>TCM29C14</td>
<td>A and μ</td>
<td>1.544, 1.536, 2.048</td>
<td>8</td>
<td>Includes 8th-bit signal</td>
</tr>
<tr>
<td>TCM29C16</td>
<td>μ</td>
<td>2.048</td>
<td>8</td>
<td>16-pin package</td>
</tr>
<tr>
<td>TCM29C17</td>
<td>A</td>
<td>2.048</td>
<td>8</td>
<td>16-pin package</td>
</tr>
<tr>
<td>TCM29C18</td>
<td>μ</td>
<td>2.048</td>
<td>8</td>
<td>Low-cost DSP interface</td>
</tr>
<tr>
<td>TCM29C19</td>
<td>μ</td>
<td>1.536</td>
<td>8</td>
<td>Low-cost DSP interface</td>
</tr>
<tr>
<td>TCM29C23</td>
<td>A and μ</td>
<td>Up to 4.096</td>
<td>8</td>
<td>Extended frequency range</td>
</tr>
<tr>
<td>TCM29C26</td>
<td>A and μ</td>
<td>Up to 4.096</td>
<td>8</td>
<td>Low-power TCM29C23</td>
</tr>
<tr>
<td>TCM320AC36</td>
<td>μ and linear</td>
<td>Up to 4.096</td>
<td>8 and 13</td>
<td>Single voltage (5) VBAP</td>
</tr>
<tr>
<td>TCM320AC37</td>
<td>A and linear</td>
<td>Up to 4.096</td>
<td>8 and 13</td>
<td>Single voltage (5) VBAP</td>
</tr>
<tr>
<td>TCM320AC38</td>
<td>μ and linear</td>
<td>Up to 4.096</td>
<td>8 and 13</td>
<td>Single voltage (5) GSM</td>
</tr>
<tr>
<td>TCM320AC39</td>
<td>A and linear</td>
<td>Up to 4.096</td>
<td>8 and 13</td>
<td>Single voltage (5) GSM</td>
</tr>
<tr>
<td>TP3054/64</td>
<td>μ</td>
<td>1.544, 1.536, 2.048</td>
<td>8</td>
<td>National Semiconductor second source</td>
</tr>
<tr>
<td>TP3054/67</td>
<td>A</td>
<td>1.544, 1.536, 2.048</td>
<td>8</td>
<td>National Semiconductor second source</td>
</tr>
<tr>
<td>TLC320AC01</td>
<td>Linear</td>
<td>43.2 kHz</td>
<td>14</td>
<td>5-V-only analog interface</td>
</tr>
<tr>
<td>TLC32040/1</td>
<td>Linear</td>
<td>Up to 19.2-kHz sampling</td>
<td>14</td>
<td>For high-dynamic linearity</td>
</tr>
<tr>
<td>TLC32044/5</td>
<td>Linear</td>
<td>Up to 19.2-kHz sampling</td>
<td>14</td>
<td>For high-dynamic linearity</td>
</tr>
<tr>
<td>TLC32046</td>
<td>Linear</td>
<td>Up to 25-kHz sampling</td>
<td>14</td>
<td>For high-dynamic linearity</td>
</tr>
<tr>
<td>TLC32047</td>
<td>Linear</td>
<td>Up to 25-kHz sampling</td>
<td>14</td>
<td>For high-dynamic linearity</td>
</tr>
</tbody>
</table>

† Clock rate in MHz, unless otherwise noted

Table 4–4. Telecom Devices—Transient Suppressor ICs

<table>
<thead>
<tr>
<th>Device</th>
<th>Function</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCM1030</td>
<td>Transient suppressor for SLIC-based line card</td>
<td>(30 A max)</td>
</tr>
<tr>
<td>TCM1060</td>
<td>Transient suppressor for SLIC-based line card</td>
<td>(60 A max)</td>
</tr>
</tbody>
</table>
4.3 Dedicated Speech Synthesis Applications

For dedicated speech synthesis applications, TI offers a family of dedicated speech synthesizer chips. This speech technology has been used in a wide range of products including games, toys, burglar alarms, fire alarms, automobiles, airplanes, answering machines, voice mail, industrial control machines, office machines, advertisements, novelty items, exercise machines, and learning aids.

4.3.1 System Design Considerations

Dedicated speech synthesis chips are effective in low-cost applications. The speech synthesis technology provided by the dedicated chips is either LPC (linear-predictive coding) or CVSD (continuously variable slope delta modulation). Table 4–5 shows the characteristics of the TI voice synthesizers.

Table 4–5. Voice Synthesizers

<table>
<thead>
<tr>
<th>Device</th>
<th>Microprocessor</th>
<th>Synthesis Method</th>
<th>I/O Pins</th>
<th>On-Chip Memory (Bits)</th>
<th>External Memory</th>
<th>Data Rate (Bits/Sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSP50C4x</td>
<td>8-bit</td>
<td>LPC–10</td>
<td>20/32</td>
<td>64K/128K</td>
<td>VROM</td>
<td>1200–2400</td>
</tr>
<tr>
<td>TSP50C1x</td>
<td>8-bit</td>
<td>LPC–12</td>
<td>10</td>
<td>64K/128K</td>
<td>VROM</td>
<td>1200–2400</td>
</tr>
<tr>
<td>TSP53C30</td>
<td>8-bit</td>
<td>LPC–10</td>
<td>20</td>
<td>N/A</td>
<td>From host microprocessor</td>
<td>1200–2400</td>
</tr>
<tr>
<td>TSP50C20</td>
<td>8-bit</td>
<td>LPC–10</td>
<td>32</td>
<td>N/A</td>
<td>EPROM</td>
<td>1200–2400</td>
</tr>
<tr>
<td>TMS3477</td>
<td>N/A</td>
<td>CVSD</td>
<td>2</td>
<td>None</td>
<td>DRAM</td>
<td>16K–32K</td>
</tr>
</tbody>
</table>
4.3.2 Speech Synthesis-Related Devices

TI has low-cost memories that are ideal for use with speech synthesizers chips. TI can also be of assistance in developing and processing the speech data that is used in these speech synthesis systems. Table 4–6 lists the TSP60Cxx family of speech memory devices. Additionally, audio filters are listed in Table 4–2 on page 4-4.

Table 4–6. Speech Memories

<table>
<thead>
<tr>
<th>Device</th>
<th>Size</th>
<th>No. of Pins</th>
<th>Interface</th>
<th>For use with</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSP60C18</td>
<td>256K</td>
<td>16</td>
<td>Parallel 4-bit</td>
<td>TSP50C1x</td>
</tr>
<tr>
<td>TSP60C19</td>
<td>256K</td>
<td>16</td>
<td>Serial</td>
<td>TSP50C4x</td>
</tr>
<tr>
<td>TSP60C20</td>
<td>256K</td>
<td>28</td>
<td>Parallel/serial 8-bit</td>
<td>TSP50C4x</td>
</tr>
<tr>
<td>TSP60C80</td>
<td>1M</td>
<td>28</td>
<td>Serial</td>
<td>TSP50C4x</td>
</tr>
<tr>
<td>TSP60C81</td>
<td>1M</td>
<td>28</td>
<td>Parallel 4-bit</td>
<td>TSP50C1x</td>
</tr>
</tbody>
</table>

Speech Synthesis Development Tools

Software:
- EVM: Code development tool

Speech:
- SAB: Speech audition board
- SD85000: PC-based speech analysis system

System:
- SEB: System emulator board
- SEB60Cxx: System emulator boards for speech memories

For further information on these speech synthesis products, call the Semiconductor Product Information Center (PIC) as listed in If You Need Assistance on page xvii.
4.4 Servo Control/Disk Drive Applications

Several years ago, most servo control systems used only analog circuitry. However, the growth of digital signal processing has made digital control theory a reality.

In a DSP-based control system, the control algorithm is implemented via software. No component aging or temperature drift is associated with digital control systems. Additionally, sophisticated algorithms can be implemented and easily modified to upgrade system performance.

Figure 4–8 shows a block diagram of a generic digital control system using a DSP, along with an ADC and DAC.

**Figure 4–8. Generic Servo Control Loop**

4.4.1 System Design Considerations

TMS320 DSPs have facilitated the development of high-speed digital servo control for disk drive and industrial control applications. Disk drives have increased storage capacity from 5 MB to over 1 GB in the past decade, which equates to a 23 900 percent growth in capacity. To accommodate these increasingly higher densities, the data on the servo platters, whether servo-positioning or actual storage information, must be converted to digital electronic signals at increasingly closer points in relation to the platter “pick-off” point. The ADC must have increasingly higher conversion rates and greater resolution to accommodate the increasing bandwidth requirements of higher storage densities. In addition, the ADC conversion rates must increase to accommodate the shorter data retrieval access time.

Figure 4–9 shows a block diagram of a disk drive control system. Figure 4–10 shows the interfacing of the 'C14 and the TLC32071.
Figure 4–9. Disk Drive Control System Block Diagram

Figure 4–10. TMS320C14 to TLC32071 Interface
4.4.2 Servo Control/Disk Drive-Related Devices

As listed in Table 4–7, TI provides a complete array of analog and digital interface devices. These devices support the TMS320 DSPs for servo control applications. For application assistance or additional information, call the Semiconductor Product Information Center (PIC) as listed in If You Need Assistance on page xvii.

Table 4–7. Control Related Devices

<table>
<thead>
<tr>
<th>Function</th>
<th>Device</th>
<th>Bits</th>
<th>Speed</th>
<th>Channels</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC</td>
<td>TLC1550</td>
<td>10</td>
<td>3–5 μs</td>
<td>1</td>
<td>Parallel</td>
</tr>
<tr>
<td></td>
<td>TLC1551</td>
<td>10</td>
<td>3–5 μs</td>
<td>1</td>
<td>Parallel</td>
</tr>
<tr>
<td></td>
<td>TLC5502/3</td>
<td>8</td>
<td>50 ns (Flash)</td>
<td>1</td>
<td>Parallel</td>
</tr>
<tr>
<td></td>
<td>TLC0820</td>
<td>8</td>
<td>1.5 μs</td>
<td>1</td>
<td>Parallel</td>
</tr>
<tr>
<td></td>
<td>TLC1225</td>
<td>13</td>
<td>12 μs</td>
<td>1 (diff.)</td>
<td>Parallel</td>
</tr>
<tr>
<td></td>
<td>TLC1558</td>
<td>10</td>
<td>3–5 μs</td>
<td>8</td>
<td>Parallel</td>
</tr>
<tr>
<td></td>
<td>TLC1543</td>
<td>10</td>
<td>21 μs</td>
<td>11</td>
<td>Serial</td>
</tr>
<tr>
<td></td>
<td>TLC1549</td>
<td>10</td>
<td>21 μs</td>
<td>1</td>
<td>Serial</td>
</tr>
<tr>
<td>DAC</td>
<td>TLC7524</td>
<td>8</td>
<td>9 MHz</td>
<td>1</td>
<td>Parallel</td>
</tr>
<tr>
<td></td>
<td>TLC7628</td>
<td>8</td>
<td>9 MHz (dual)</td>
<td>1</td>
<td>Parallel</td>
</tr>
<tr>
<td></td>
<td>TLC5602</td>
<td>8</td>
<td>30 MHz</td>
<td>1</td>
<td>Parallel</td>
</tr>
<tr>
<td>AIC</td>
<td>TLC32071 (ADC)</td>
<td>8</td>
<td>1 μs</td>
<td>8</td>
<td>Parallel</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>9 MHz</td>
<td>1</td>
<td>Parallel</td>
</tr>
</tbody>
</table>
4.5 Modem Applications

High-speed modems (9600 bps and above) require a great deal of analog signal processing in addition to digital signal processing. Designing both high-speed capabilities and slower fall-back modes poses significant engineering challenges. TI offers a number of analog front-end (AFE) circuits to support various high-speed modem standards.

The TLC32040, TLC32044, TLC32046, TLC32047, and TLC320AC01 analog interface circuits (AIC) are especially suited for modem applications by the integration of an input multiplexer, switched capacitor filters, high resolution 14-bit ADC and DAC, a four-mode serial port, and control and timing logic. These converters feature adjustable parameters, such as filtering characteristics, sampling rates, gain selection, \((\sin x)/x\) correction (TLC32044, TLC32046, and TLC32047 only), and phase adjustment. All these parameters are software programmable, making the AIC suitable for a variety of applications. Table 4–8 provides the description and characteristics of these devices.

<table>
<thead>
<tr>
<th>Device</th>
<th>Description</th>
<th>I/O</th>
<th>Resolution (Bits)</th>
<th>Conversion Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLC32040</td>
<td>Analog interface chip (AIC)</td>
<td>Serial</td>
<td>14</td>
<td>19.2 kHz</td>
</tr>
<tr>
<td>TLC32041</td>
<td>AIC without on-board VREF</td>
<td>Serial</td>
<td>14</td>
<td>19.2 kHz</td>
</tr>
<tr>
<td>TLC32044</td>
<td>Telephone speed/modem AIC</td>
<td>Serial</td>
<td>14</td>
<td>19.2 kHz</td>
</tr>
<tr>
<td>TLC32045</td>
<td>Low-cost version of the TLC32044</td>
<td>Serial</td>
<td>14</td>
<td>19.2 kHz</td>
</tr>
<tr>
<td>TLC32046</td>
<td>Wide-band AIC</td>
<td>Serial</td>
<td>14</td>
<td>25 kHz</td>
</tr>
<tr>
<td>TLC32047</td>
<td>AIC with 11.4-kHz BW</td>
<td>Serial</td>
<td>14</td>
<td>25 kHz</td>
</tr>
<tr>
<td>TLC320AC01</td>
<td>5-volt-only AIC</td>
<td>Serial</td>
<td>14</td>
<td>43.2 kHz</td>
</tr>
<tr>
<td>TCM29C18</td>
<td>Companding codec/filter</td>
<td>PCM</td>
<td>8</td>
<td>8 kHz</td>
</tr>
<tr>
<td>TCM29C23</td>
<td>Companding codec/filter</td>
<td>PCM</td>
<td>8</td>
<td>16 kHz</td>
</tr>
<tr>
<td>TCM29C26</td>
<td>Low-power codec/filter</td>
<td>PCM</td>
<td>8</td>
<td>16 kHz</td>
</tr>
<tr>
<td>TCM320AC36</td>
<td>Single-supply codec/filter</td>
<td>PCM and linear</td>
<td>8</td>
<td>25 kHz</td>
</tr>
</tbody>
</table>
The AIC interfaces directly with serial-input TMS320 DSPs, which execute the modem's high-speed encoding and decoding algorithms. The TLC3204x family performs level shifting, filtering, and A/D and D/A data conversion. The DSP's many software-programmable features provide the flexibility required for modem operations and make it possible to modify and upgrade systems easily. Under DSP control, the AIC's sampling rates permit designers to include fall-back modes without additional analog hardware in most cases. Phase adjustments can be made in real time so that the A/D and D/A conversions can be synchronized with the upcoming signal. In addition, the chip has a built-in loopback feature to support modem self-test requirements.

Figure 4–11 shows a V.32bis modem implementation using the ’C25 and a TLC320AC01. The upper ’C25 performs echo cancellation and transmit data functions, while the lower ’C25 performs receive data and timing recovery functions. The echo canceler simulates the telephone channel and generates an estimated echo of the transmit data signal. The TLC320AC01 performs the following functions:

**Upper TLC320AC01 D/A path:** Converts the estimated echo, as computed by the upper ’C25, into an analog signal, which is subtracted from the receive signal.

**Upper TLC320AC01 A/D path:** Converts the residual echo to a digital signal for purposes of monitoring the residual echo and continuously training the echo canceler for optimum performance. The converted signal is sent to the upper ’C25.

**Lower TLC320AC01 D/A path:** Converts the upper ’C25 transmit output to an analog signal, performs a smoothing filter function, and drives the DAC.

**Lower TLC320AC01 D/A path:** Converts the echo-free receive signal to a digital signal, which is sent to the lower ’C25 to be decoded.

For application assistance or additional information, call the Semiconductor Product Information Center (PIC) as listed in If You Need Assistance on page xvii.

**Note:**
The example in Figure 4–11 is for illustration only. In reality, one single ’C5x DSP can implement high-speed modem functions.
Figure 4–11. High-Speed V.32bis and Multistandard Modem With the TLC320AC01 AIC

TLC320AC01

ADC and DAC

Fine tune echo cancel

TMS320C25/C5X

Echo canceler

Transmitter

TMS320C25/C5X

Receiver

TLC320AC01

DAA

Telephone line

RS-232 interface

Serial I/O control
4.6 Advanced Digital Electronics Applications for Consumers

With the extensive use of the TMS320 DSPs in consumer electronics, much electromechanical control and signal processing can be done in the digital domain. Digital systems generally require some form of analog interface, usually in the form of high-performance ADCs and DACs. Figure 4–12 shows the general performance requirements for a variety of applications.

Figure 4–12. Applications Performance Requirements

4.6.1 Advanced Television System Design Considerations

Advanced Digital Television (ADTV) is a technology that uses digital signal processing to enhance video and audio presentations and to reduce noise and ghosting. Because of these DSP techniques, a variety of features can be implemented, including frame store, picture-in-picture, improved sound quality, and zoom. The bandwidth requirements remain at the existing 6-MHz television allocation. From the intermediate frequency (IF) output, the video signal is converted by an 8-bit video ADC. The digital output can be processed in the digital domain to provide noise reduction, interpolation or averaging for digitally increased sharpness, and higher quality audio. The DSP digital output is converted back to analog by a video DAC, as shown in Figure 4–13.
Videocassette recorders, compact disc (CD) players, digital audio tape (DAT) players, and PCs are a few of the products that have taken a major position in the marketplace in the last ten years. The audio channels for CD and DAT require 16-bit A/D resolution to meet the distortion and noise standards. See Figure 4–14 for a block diagram of a typical digital audio system.

The motion and motor control systems usually use 8- to 10-bit ADCs for the lower frequency servo loop. Tape or disc systems use motor or motion control for proper positioning of the record or playback heads. With the storage medium compressing data into an increasingly smaller physical size, the positioning systems require more precision.

The audio processing becomes more demanding as higher fidelity is required. Better fidelity translates into lower noise and distortion in the output signal.
The TMS57013DW/57014DW 1-bit DACs include an 8-times over-sampling digital filter designed for digital audio systems, such as CD players, DAT players, CDIs, LDPs, digital amplifiers, car stereos, and BS tuners. They are also suitable for all systems that include digital sound processing like TVs, VCRs, musical instruments, NICAM systems, multimedia, and so forth.

The converters have dual channels so that the right and left stereo signals can be transformed into analog signals with only one chip. There are some functions that allow the customers to select the conditions according to their applications, such as muting, attenuation, deemphasis, and zero data detection. These functions are controlled by external 16-bit serial data from a controller like a microcomputer.

The TMS5703DW/57014DW adopt 129-tap FIR filter and third-order ΔΣ modulation to get –75-dB stop band attenuation and 96-dB signal-to-noise ratio (SNR). The output is a pulse-width modulated (PWM) waveform, which facilitates an analog signal through a low-pass filter.

### 4.6.2 Advanced Digital Electronics-Related Devices

As listed in Table 4–9, TI provides a complete array of analog interface devices. These devices support the TMS320 DSPs for digital system applications. For application assistance or additional information, call the Semiconductor Product Information Center (PIC) as listed in *If You Need Assistance* on page xvii.

Table 4–9. Audio/Video Analog/Digital Interface Devices

<table>
<thead>
<tr>
<th>Function</th>
<th>Device</th>
<th>Bits</th>
<th>Speed</th>
<th>Channels</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dual audio DAC + digital filter</td>
<td>TMS57013/4</td>
<td>16/18</td>
<td>32, 37.8, 44.1, 48 kHz</td>
<td>2</td>
<td>Serial</td>
</tr>
<tr>
<td>Analog interface</td>
<td>TLC32071</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DAC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADC</td>
<td>TLC1225</td>
<td>12</td>
<td>12 μs</td>
<td>1</td>
<td>Parallel</td>
</tr>
<tr>
<td>ADC</td>
<td>TLC1550</td>
<td>10</td>
<td>6 μs</td>
<td>1</td>
<td>Parallel</td>
</tr>
<tr>
<td>Flash ADC</td>
<td>TLC5502</td>
<td>8</td>
<td>50 ns</td>
<td>1</td>
<td>Parallel</td>
</tr>
<tr>
<td>Flash ADC</td>
<td>TLC5503</td>
<td>8</td>
<td>100 ns</td>
<td>1</td>
<td>Parallel</td>
</tr>
<tr>
<td>Triple video DAC</td>
<td>TL5632</td>
<td>8</td>
<td>16 ns</td>
<td>3</td>
<td>Parallel</td>
</tr>
<tr>
<td>Triple Flash ADC</td>
<td>TL5703</td>
<td>8</td>
<td>70 ns</td>
<td>3</td>
<td>Parallel</td>
</tr>
<tr>
<td>Video DAC</td>
<td>TL5602</td>
<td>8</td>
<td>50 ns</td>
<td>1</td>
<td>Parallel</td>
</tr>
<tr>
<td>Video DAC</td>
<td>TLC5602</td>
<td>8</td>
<td>50 ns</td>
<td>1</td>
<td>Parallel</td>
</tr>
</tbody>
</table>
The ‘C5x DSPs support emulation through a dedicated emulation port. The emulation port is a superset of the IEEE JTAG standard 1149.1 and can be accessed by the XDS510 emulator. This appendix provides information pertaining to the XDS510 cable #2563988-001 revision B.

The term JTAG, as used in this book, refers to TI scan-based emulation, which is based on the IEEE standard 1149.1. For more information concerning the IEEE standard 1149.1, contact IEEE Customer Service:

Address:  IEEE Customer Service
        445 Hoes Lane, PO Box 1331
        Piscataway, NJ 08855-1331

Phone:   (800) 678–IEEE in the US and Canada
         (908) 981–1393 outside the US and Canada

FAX:     (908) 981–9667  Telex: 833233

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.1 Cable Header and Signals</td>
<td>A-2</td>
</tr>
<tr>
<td>A.2 Bus Protocol</td>
<td>A-3</td>
</tr>
<tr>
<td>A.3 Emulator Cable Pod</td>
<td>A-4</td>
</tr>
<tr>
<td>A.4 Emulator Cable Pod Signal Timings</td>
<td>A-6</td>
</tr>
<tr>
<td>A.5 Target System Test Clock</td>
<td>A-7</td>
</tr>
<tr>
<td>A.6 Configuring Multiple Processors</td>
<td>A-8</td>
</tr>
<tr>
<td>A.7 Connections Between the Emulator and the Target System</td>
<td>A-9</td>
</tr>
<tr>
<td>A.8 Emulation Timing Calculations</td>
<td>A-11</td>
</tr>
</tbody>
</table>
A.1 Cable Header and Signals

To perform emulation with the XDS510, your target system must have a 14-pin header (two 7-pin rows) with connections as shown in Figure A–1. Table A–1 describes the emulation signals. Although you can use other headers, recommended parts include:

- Straight header, unshrouded: DuPont Electronics part number 67996–114
- Right-angle header, unshrouded: DuPont Electronics part number 68405–114

**Figure A–1. Header Signals and Header Dimensions**

```
  1   2   3   4   5   6   7   8   9   10  11  12  13  14
TMS  TRST
TDI  GND
PD (5 V) No pin (key)
TDO  GND
TCK_RET GND
TCK  GND
EMU0  EMU1

Header Dimensions:
- Pin-to-pin spacing: 0.100 inch (X,Y)
- Pin width: 0.025 inch square post
- Pin length: 0.235 inch, nominal
```

**Table A–1. XDS510 Header Signal Description**

<table>
<thead>
<tr>
<th>Pin</th>
<th>Signal</th>
<th>State</th>
<th>Target State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TMS</td>
<td>O</td>
<td>I</td>
<td>JTAG test mode select</td>
</tr>
<tr>
<td>2</td>
<td>TRST</td>
<td>O</td>
<td>I</td>
<td>JTAG test reset</td>
</tr>
<tr>
<td>3</td>
<td>TDI</td>
<td>O</td>
<td>I</td>
<td>JTAG test data input</td>
</tr>
<tr>
<td>5</td>
<td>PD</td>
<td>I</td>
<td>O</td>
<td>Presence detect. Indicates that the emulation cable is connected and that the target is powered up. PD must be tied to 5 volts in the target system.</td>
</tr>
<tr>
<td>7</td>
<td>TDO</td>
<td>I</td>
<td>O</td>
<td>JTAG test data output</td>
</tr>
<tr>
<td>9</td>
<td>TCK_RET</td>
<td>I</td>
<td>O</td>
<td>JTAG test clock return. Test clock input to the XDS510 emulator. May be a buffered or unbuffered version of TCK.</td>
</tr>
<tr>
<td>11</td>
<td>TCK</td>
<td>O</td>
<td>I</td>
<td>JTAG test clock. TCK is a 10-MHz clock source from the emulation cable pod. This signal can be used to drive the system test clock.</td>
</tr>
<tr>
<td>13</td>
<td>EMU0</td>
<td>I</td>
<td>I/O</td>
<td>Emulation pin 0</td>
</tr>
<tr>
<td>14</td>
<td>EMU1</td>
<td>I</td>
<td>I/O</td>
<td>Emulation pin 1</td>
</tr>
</tbody>
</table>
A.2 Bus Protocol

The IEEE standard 1149.1 covers the requirements for JTAG bus slave devices (C5x) and provides certain rules, summarized as follows:

- The TMS and TDI inputs are sampled on the rising edge of the TCK signal of the device.
- The TDO output is clocked from the falling edge of the TCK signal of the device.

When JTAG devices are daisy-chained together, the TDO of one device has approximately a half TCK cycle setup time before the next device’s TDI signal. This timing scheme minimizes race conditions that would occur if both TDO and TDI were timed from the same TCK edge. The penalty for this timing scheme is a reduced TCK frequency.

The IEEE standard 1149.1 does not provide rules for JTAG bus master (XDS510) devices. Instead, it states that it expects a bus master to provide bus-slave compatible timings. The XDS510 provides timings that meet the bus slave rules and also provides an optional timing mode that allows you to run the emulation at a much higher frequency for improved performance.
A.3 Emulator Cable Pod

Figure A–2 shows a portion of the XDS510 emulator cable pod. The functional features of the emulator pod are:

- TDO and TCK_RET can be parallel-terminated inside the pod if required by the application. By default, these signals are not terminated.

- TCK is driven with a 74AS1034 device. Because of the high-current drive (48 mA $I_{OL}/I_{OH}$), this signal can be parallel-terminated. If TCK is tied to TCK_RET, you can use the parallel terminator in the pod.

- TMS and TDI can be generated from the falling edge of TCK_RET, according to the IEEE (JTAG) standard 1149.1 bus-slave device timing rules. They can also be driven from the rising edge of TCK_RET, which allows a higher TCK_RET frequency. The default is to match the IEEE standard 1149.1 slave device timing rules. This is an emulator software option that can be selected when the emulator is invoked. In general, single-processor applications can benefit from the higher clock frequency. However, in multiprocessing applications, you may wish to use the IEEE standard 1149.1 bus slave timing mode to minimize emulation system timing constraints.

- TMS and TDI are series-terminated to reduce signal reflections.

- A 10-MHz test clock source is provided. You can also provide your own test clock for greater flexibility.
Figure A–2. Emulator Cable Pod Interface

Note: All devices are 74AS, unless otherwise specified.
A.4 Emulator Cable Pod Signal Timings

Figure A–3 shows the signal timings for the emulator cable pod. Table A–2 defines the timing parameters illustrated in the figure. These timing parameters are calculated from values specified in the standard data sheets for the cable pod and are for reference only. Texas Instruments does not test or guarantee these timings.

The emulator pod uses TCK_RET as its clock source for internal synchronization. TCK is provided as an optional target system test clock source.

Figure A–3. Emulator Cable Pod Timings

![Timings Diagram]

Table A–2. Emulator Cable Pod Timing Parameters

<table>
<thead>
<tr>
<th>No.</th>
<th>Parameter</th>
<th>Description</th>
<th>Min</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>tTCKmin</td>
<td>TCK_RET period</td>
<td>35</td>
<td>200</td>
<td>ns</td>
</tr>
<tr>
<td></td>
<td>tTCKmax</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>tTCKhighmin</td>
<td>TCK_RET high pulse duration</td>
<td>15</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>3</td>
<td>tTCKlowmin</td>
<td>TCK_RET low pulse duration</td>
<td>15</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>4</td>
<td>t_d(XTMXmin)</td>
<td>TMS/TDI valid from TCK_RET low (default timing)</td>
<td>6</td>
<td>20</td>
<td>ns</td>
</tr>
<tr>
<td></td>
<td>t_d(XTMXmax)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>t_d(XTMSmin)</td>
<td>TMS/TDI valid from TCK_RET high (optional timing)</td>
<td>7</td>
<td>24</td>
<td>ns</td>
</tr>
<tr>
<td></td>
<td>t_d(XTMSmax)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>t_su(XTDOmin)</td>
<td>TDO setup time to TCK_RET high</td>
<td>3</td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>7</td>
<td>t_hd(XTDOmin)</td>
<td>TDO hold time from TCK_RET high</td>
<td>12</td>
<td></td>
<td>ns</td>
</tr>
</tbody>
</table>
A.5 Target System Test Clock

Figure A–4 shows an application with the system test clock generated in the target system. In this application the TCK signal is left unconnected. There are two benefits to having the target system generate the test clock:

1) You can set the test clock frequency to match your system requirements. The emulator provides only a single 10-MHz test clock.

2) You may have other devices in your system that require a test clock when the emulator is not connected.

Figure A–4. Target-System Generated Test Clock
A.6 Configuring Multiple Processors

Figure A–5 shows a typical daisy-chained multiprocessor configuration that meets the minimum requirements of the IEEE (JTAG) standard 1149.1. The emulation signals are buffered to isolate the processors from the emulator and provide adequate signal drive for the target system. One of the benefits of this test interface is that you can slow down the test clock to eliminate timing problems. Several key points to multiprocessor support are as follows:

- The processor TMS, TDI, TDO, and TCK signals should be buffered through the same physical device package for better control of timing skew.

- The input buffers for TMS, TDI, and TCK should have pullup resistors connected to 5 V to hold these signals at a known value when the emulator is not connected. A pullup resistor value of 4.7 kΩ or greater is suggested.

- Buffering EMU0 and EMU1 is optional but highly recommended to provide isolation. These are not critical signals and do not have to be buffered through the same physical package as TMS, TCK, TDI, and TDO.

Figure A–5. Multiprocessor Connections
A.7 Connections Between the Emulator and the Target System

It is extremely important to provide high-quality signals between the emulator and the target system. You must supply the correct signal buffering, test clock inputs, and multiple processor interconnections to ensure proper emulator and target system operation.

EMU0 and EMU1 are I/O pins on the 'C5x; however, they are only inputs to the XDS510. In general, these pins are used in multiprocessor systems to provide global run/stop operations.

A.7.1 Emulation Signals Not Buffered

If the distance between the emulation header and the target device is less than 6 inches, no buffering is necessary. Figure A–6 shows the no-buffering configuration.

The EMU0 and EMU1 signals must have pullup resistors connected to 5 V to provide a signal rise time of less than 10 μs. A 4.7-kΩ resistor is suggested for most applications.

Figure A–6. Emulator Connections Without Signal Buffering
A.7.2 Emulation Signals Buffered

If the distance between the emulation header and the JTAG target device is greater than 6 inches, the emulation signals must be buffered. Figure A–7 shows the buffering configuration. Emulation signals TMS, TDI, TDO, and TCK_RET are buffered through the same device package.

The EMU0 and EMU1 signals must have pullup resistors connected to 5 V to provide a signal rise time of less than 10 μs. A 4.7-kΩ resistor is suggested for most applications.

To have high-quality signals (especially the processor TCK and the emulator TCK_RET signals), you may have to employ special care when routing the printed wiring board trace. You also may have to use termination resistors to match the trace impedance. The emulator pod provides optional internal parallel terminators on the TCK_RET and TDO. TMS and TDI provide fixed series termination.

*Figure A–7. Buffered Signals*
A.8 Emulation Timing Calculations

The following are a few examples of how to calculate the emulation timings in your system. For actual target timing parameters, see the appropriate device data sheets.

Assumptions:

- $t_{su}(TTMS)$ Target TMS/TDI setup to TCK high $10$ ns
- $t_{h}(TTMS)$ Target TMS/TDI hold from TCK high $5$ ns
- $t_{d}(TTDO)$ Target TDO delay from TCK low $15$ ns
- $t_{d}(buf\max)$ Target buffer delay maximum $10$ ns
- $t_{d}(buf\min)$ Target buffer delay minimum $1$ ns
- $t_{bufskew}$ Target buffer skew between two devices in the same package: $(t_{d}(buf\max) - t_{d}(buf\min)) \times 0.15$ $1.35$ ns
- $t_{cckfactor}$ A 40/60 duty cycle clock $0.4$

Given in Table A–2 (page A-6):

- $t_{d}(XTMS\max)$ XDS510 TMS/TDI delay from TCK_RET low, maximum $20$ ns
- $t_{d}(XTMX)$ XDS510 TMS/TDI delay from TCK_RET low, minimum $6$ ns
- $t_{d}(XTMS\max)$ XDS510 TMS/TDI delay from TCK_RET high, maximum $24$ ns
- $t_{d}(XTMX\min)$ XDS510 TMS/TDI delay from TCK_RET high, minimum $7$ ns
- $t_{su}(XTDOS\min)$ TDO setup time to XDS510 TCK_RET high $3$ ns

There are two key timing paths to consider in the emulation design:

1) The TCK_RET/TMS/TDI ($t_{prdtck\_TMS}$) path
2) The TCK_RET/TDO ($t_{prdtck\_TDO}$) path

In each case, the worst-case path delay is calculated to determine the maximum system test clock frequency.
**Emulation Timing Calculations**

**Case 1:** Single processor, direct connection, TMS/TDI timed from TCK_RET low (default timing).

\[
\begin{align*}
\tau_{\text{prdtck,TMS}} &= \left[ t_d(XTMS_{\text{max}}) + t_{\text{su(TTMS)}} \right] / t_{\text{tckfactor}} \\
&= (20 \text{ ns} + 10 \text{ ns}) / 0.4 \\
&= 75 \text{ ns (13.3 MHz)}
\end{align*}
\]

\[
\begin{align*}
\tau_{\text{prdtck,TDO}} &= \left[ t_d(TTDO) + t_{\text{su(XTDO_{min})}} \right] / t_{\text{tckfactor}} \\
&= (15 \text{ ns} + 3 \text{ ns}) / 0.4 \\
&= 45 \text{ ns (22.2 MHz)}
\end{align*}
\]

In Case 1, the TCK/TMS path is the limiting factor.

**Case 2:** Single processor, direct connection, TMS/TDI timed from TCK_RET high (optional timing).

\[
\begin{align*}
\tau_{\text{prdtck,TMS}} &= t_d(XTMS_{\text{max}}) + t_{\text{su(TTMS)}} \\
&= (24 \text{ ns} + 10 \text{ ns}) \\
&= 34 \text{ ns (29.4 MHz)}
\end{align*}
\]

\[
\begin{align*}
\tau_{\text{prdtck,TDO}} &= \left[ t_d(TTDO) + t_{\text{su(XTDO_{min})}} \right] / t_{\text{tckfactor}} \\
&= (15 + 3) / 0.4 \\
&= 45 \text{ ns (22.2 MHz)}
\end{align*}
\]

In Case 2, the TCK/TDO path is the limiting factor. One other thing to consider in this case is the TMS/TDI hold time. The minimum hold time for the XDS510 cable pod is 7 ns, which meets the 5-ns hold time of the target device.

**Case 3:** Single/multiple processor, TMS/TDI buffered input; TCK_RET/TDO buffered output, TMS/TDI timed from TCK_RET high (optional timing).

\[
\begin{align*}
\tau_{\text{prdtck,TMS}} &= t_d(XTMS_{\text{max}}) + t_{\text{su(TTMS)}} + 2t_d(buf_{\text{max}}) \\
&= 24 \text{ ns} + 10 \text{ ns} + 2(10) \\
&= 54 \text{ ns (18.5 MHz)}
\end{align*}
\]

\[
\begin{align*}
\tau_{\text{prdtck,TDO}} &= \left[ t_d(TTDO) + t_{\text{su(XTDO_{min})}} + t_{\text{bufskew}} \right] / t_{\text{tckfactor}} \\
&= (15 \text{ ns} + 3 \text{ ns} + 1.35 \text{ ns}) / 0.4 \\
&= 58.4 \text{ ns (20.7 MHz)}
\end{align*}
\]

In Case 3, the TCK/TMS path is the limiting factor. The hold time on TMS/TDI is also reduced by the buffer skew (1.35 ns) but still meets the minimum device hold time.
**Case 4:** Single/multiprocessor, TMS/TDI/TCK buffered input; TDO buffered output, TMS/TDI timed from TCK RET low (default timing).

\[
t_{\text{prdtck_TMS}} = \left( t_d(X_{\text{TM}S_{\text{max}}}) + t_{su}(T_{\text{TM}S}) + t_{\text{bufskew}} \right) / t_{\text{ckfactor}}
\]
\[
= (24 \text{ ns} + 10 \text{ ns} + 1.35 \text{ ns}) / 0.4
\]
\[
= 88.4 \text{ ns (11.3 MHz)}
\]

\[
t_{\text{prdtck_TDO}} = \left( t_d(T_{\text{DO}}) + t_{su}(X_{\text{TD}O_{\text{min}}}) + t_{d(bufmax)} \right) / t_{\text{ckfactor}}
\]
\[
= (15 \text{ ns} + 3 \text{ ns} + 10 \text{ ns}) / 0.4
\]
\[
= 70 \text{ ns (14.3 MHz)}
\]

In Case 4, the TCK/TMS path is the limiting factor.

In a multiprocessor application, it is necessary to ensure that the EMU0 and EMU1 lines can go from a logic low level to a logic high level in less than 10 \( \mu \text{s} \). This can be calculated as follows (remember that \( t = 5 RC \)):

\[
t_{\text{rise}} = 5(R_{\text{pullup}} \times N_{\text{devices}} \times C_{\text{load_per_device}})
\]
\[
= 5(4.7 \text{ k}\Omega \times 16 \times 15 \text{ pF})
\]
\[
= 5.64 \mu\text{s}
\]
This appendix provides development support information, device part numbers, and support tool ordering information for the ‘C5x.

Each ‘C5x support product is described in the TMS320 DSP Development Support Reference Guide. In addition, more than 100 third-party developers offer products that support the TI TMS320 family. For more information, refer to the TMS320 Third-Party Support Reference Guide.

For information on pricing and availability, contact the nearest TI Field Sales Office or authorized distributor. See the list at the back of this book.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.1 Development Support</td>
<td>B-2</td>
</tr>
<tr>
<td>B.2 Part Order Information</td>
<td>B-4</td>
</tr>
<tr>
<td>B.3 Hewlett-Packard E2442A Preprocessor ‘C5x Interface</td>
<td>B-8</td>
</tr>
</tbody>
</table>
B.1 Development Support

This section describes the development support provided by Texas Instruments.

B.1.1 Software and Hardware Development Tools

TI offers an extensive line of development tools for the 'C5x generation of DSPs, including tools to evaluate the performance of the processors, generate code, develop algorithm implementations, and fully integrate and debug software and hardware modules. The following products support development of 'C5x-based applications:

- Software development tools:
  - Assembler/linker
  - Simulator
  - Optimizing ANSI C compiler
  - Application algorithms
  - C/Assembly debugger and code profiler

- Hardware development tools:
  - Emulator XDS510
  - 'C5x Evaluation Module (EVM)
  - 'C5x DSP Starter Kit (DSK)

B.1.2 Third-Party Support

The TMS320 family is supported by products and services from more than 100 independent third-party vendors and consultants. These support products take various forms (both as software and hardware), from cross-assemblers, simulators, and DSP utility packages to logic analyzers and emulators. The expertise of those involved in support services ranges from speech encoding and vector quantization to software/hardware design and system analysis.

To ask about third-party services, products, applications, and algorithm development packages, contact the third party directly. Refer to the TMS320 Third-Party Support Reference Guide for addresses and phone numbers.
B.1.3 TMS320C5x DSP Design Workshop

This workshop is tailored for hardware and software design engineers and decision-makers who design and utilize the ’C5x generation of DSP devices. Hands-on exercises throughout the course give participants a rapid start in developing ’C5x design skills. Microprocessor/assembly language experience is required. Experience with digital design techniques and C language programming experience is desirable.

These topics are covered in the ’C5x workshop:

- DSP fundamentals
- ’C5x architecture/instruction set
- Use of the PC-based software simulator
- Use of the ’C5x assembler/linker
- C programming environment
- System architecture considerations
- Memory and I/O interfacing
- Serial ports and multiple processor features

For registration information, pricing, or to enroll, call (972) 644–5580.

B.1.4 Assistance

For assistance to TMS320 questions on device problems, development tools, documentation, software upgrades, and new products, you can contact TI. See If You Need Assistance on page xvii for information.
Part Order Information

B.2 Part Order Information

This section describes the part numbers of ‘C5x devices, development support hardware, and software tools.

B.2.1 Device and Development Support Tool Nomenclature

To designate the stages in the product development cycle, TI assigns prefixes to the part numbers of all TMS320 devices and support tools. Each TMS320 member has one of three prefix designators: TMX, TMP, or TMS. Each support tool has one of two possible prefix designators: TMDX or TMDS. These prefixes represent evolutionary stages of product development, from engineering prototypes (TMX/TMDX) through fully qualified production devices and tools (TMS/TMDS). This development flow is defined below.

Device Development Flow:

TMX  The part is an experimental device that is not necessarily representative of the final device’s electrical specifications.

TMP  The part is a device from a final silicon die that conforms to the device’s electrical specifications but has not completed quality and reliability verification.

TMS  The part is a fully qualified production device.

Support Tool Development Flow:

TMDX  The development-support product that has not yet completed Texas Instruments internal qualification testing.

TMDS  The development-support product is a 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.”

TMS devices and TMDS development-support tools have been fully characterized, and the quality and reliability of the device has been fully demonstrated. Texas Instruments standard warranty applies to these products.

Note:

It is expected that prototype devices (TMX or TMP) have a greater failure rate than standard production devices. Texas Instruments recommends that these devices not be used in any production system, because their expected end-use failure rate is still undefined. Only qualified production devices should be used.
## B.2.2 Device Nomenclature

TI device nomenclature includes the device family name and a suffix. Figure B–1 provides a legend for reading the complete device name for any TMS320C5x family member.

**Figure B–1. TMS320C5x Device Nomenclature**

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Temperature range</th>
<th>Package type</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMX = Experimental device</td>
<td>H = 0 to 50°C</td>
<td>FD = Ceramic leadless CC</td>
<td>'C5x DSP:</td>
</tr>
<tr>
<td>TMP = Prototype device</td>
<td>L = 0 to 70°C</td>
<td>FN = Plastic leaded CC</td>
<td>50</td>
</tr>
<tr>
<td>TMS = Qualified device</td>
<td>A = -40 to 85°C</td>
<td>FZ = Ceramic CER-QUAD</td>
<td>51</td>
</tr>
<tr>
<td>SM = High reliability (non 883C)</td>
<td>S = -55 to 100°C</td>
<td>GB = Ceramic PGA</td>
<td>52</td>
</tr>
<tr>
<td>SMJ = MIL-STD-883C</td>
<td>M = -55 to 125°C</td>
<td>J = Ceramic CER-DIP</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JD = Ceramic DIP side-brazed</td>
<td>53S</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N = Plastic DIP</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PJ = 100-pin plastic EIAJ QFP</td>
<td>57</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PQ = 100/132-pin plastic BQFP</td>
<td>57S</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PZ = 100-pin plastic TQFP</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PBK = 120/128-pin plastic TQFP</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PGE = 144-pin plastic TQFP</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Device family</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>320 = DSP Family</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low voltage (3.3 V) option</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C = CMOS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E = CMOS EPROM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boot loader option</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Technology</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C = CMOS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E = CMOS EPROM</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### B.2.3 Development Support Tools

Figure B–2 provides a legend for reading the part number for any TMS320 hardware or software development tool. Table B–1 lists the development support tools available for the 'C5x, the platform on which they run, and their part numbers.

**Figure B–2. TMS320 Development Tool Nomenclature**

<table>
<thead>
<tr>
<th>TMDS</th>
<th>32</th>
<th>4</th>
<th>28</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qualification status</td>
<td>TMDX = Prototype</td>
<td>TMDS = Qualified</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Device family</td>
<td>32 = TMS320 family</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Product type</td>
<td>4 = Software</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6 = Hardware</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8 = Upgrade</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Model</td>
<td>11 = XDS/11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>22 = XDS/22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>88 = Upgrade kits</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operating system</td>
<td>02 = 'C1x VAX/VMS™</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>08 = 'C1x IBM MS/PC-DOS™</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>22 = 'C2x VAX/VMS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>25 = 'C2x/'C2xx/'C5x SPARC™</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>28 = 'C2x or 'C1x/'C2x/'C2xx/'C5x IBM MS/PC-DOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>32 = 'C3x VAX/VMS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>38 = 'C3x IBM MS/PC-DOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>42 = 'C4x VAX/VMS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>48 = 'C4x IBM MS/PC-DOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>52 = 'C5x VAX/VMS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>55 = 'C5x or 'C2xx/'C5x SPARC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>58 = 'C5x or 'C2xx/'C5x IBM MS/PC-DOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Medium†</td>
<td>2 = 5.25-inch floppy disk</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8 = 1600 BPI magnetic tape</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S/W format†</td>
<td>0 = Object code</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 = Source code</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequence number‡</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generation‡</td>
<td>1 = 'C1x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 = 'C2x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3 = 'C3x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 = 'C4x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5 = 'C5x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Format‡</td>
<td>1 = TI-tagged</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5 = COFF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

† Software only
‡ Hardware only
Table B–1. TMS320C5x Development Support Tools Part Numbers

<table>
<thead>
<tr>
<th>Development Tool</th>
<th>Platform</th>
<th>Part Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembler/linker</td>
<td>PC (DOS, OS/2)</td>
<td>TMDS3242850-02</td>
</tr>
<tr>
<td>C Compiler/assembler/linker</td>
<td>PC (DOS, OS/2)</td>
<td>TMDS3242855-02</td>
</tr>
<tr>
<td>C Compiler/assembler/linker</td>
<td>HP (HP-UX) / SPARC (Sun OS)</td>
<td>TMDS3242555-08</td>
</tr>
<tr>
<td>Digital Filter Design Package</td>
<td>PC (DOS)</td>
<td>DFDP</td>
</tr>
<tr>
<td>DSP Starter Kit (DSK)</td>
<td>PC (DOS)</td>
<td>TMDS3200051</td>
</tr>
<tr>
<td>Evaluation Module (EVM)</td>
<td>PC (DOS, Windows 3.xx)</td>
<td>TMDS3260050</td>
</tr>
<tr>
<td>Simulator (C language)</td>
<td>PC (DOS, Windows 3.xx)</td>
<td>TMDS3245851-02</td>
</tr>
<tr>
<td>Simulator (C language)</td>
<td>SPARC (Sun OS)</td>
<td>TMDS3245551-09</td>
</tr>
<tr>
<td>XDSS10 debugger/emulation software</td>
<td>PC (DOS, Windows 3.xx, OS/2)</td>
<td>TMDS3240150</td>
</tr>
<tr>
<td>XDSS10xl emulator†</td>
<td>PC (DOS, OS/2)</td>
<td>TMDS00510</td>
</tr>
<tr>
<td>XDSS10WS debugger/emulation software</td>
<td>SPARC (Sun OS)</td>
<td>TMDS3240650</td>
</tr>
<tr>
<td>XDSS10WS emulator‡</td>
<td>SPARC (Sun OS)</td>
<td>TMDS00510WS</td>
</tr>
<tr>
<td>3 V/5 V PC/SPARC JTAG emulation cable</td>
<td>XDS510 / XDS510WS</td>
<td>TMDS3080002</td>
</tr>
</tbody>
</table>

† Includes XDSS10 board and JTAG cable
‡ Includes XDSS10WS box and JTAG cable
B.3 Hewlett-Packard E2442A Preprocessor 'C5x Interface

The Hewlett-Packard E2442A preprocessor 'C5x interface provides a mechanical and electrical connection between your target system and an HP logic analyzer. Preprocessor hardware captures processor signals and passes them to the logic analyzer at the appropriate time, depending on the type of measurement you are making. With the preprocessor plugged in, both state and timing analysis is available. Two connectors are loaded onto the preprocessor to facilitate communications with other debugging tools. A BNC connector, when used with the sequencer of the logic analyzer halts the processor on a condition. Then you can use the 'C5x HLL debugger to examine the state of the system (for example, microprocessor registers). Likewise, a 14-pin connector is available to receive signals from the XDS510 development system. These signals can be used when defining a trigger condition for the analyzer.

The preprocessor includes software that automatically labels address, data, and status lines. Additionally, a disassembler is included. The disassembler processes state traces and displays the information on TMS320 mnemonics.

B.3.1 Capabilities

The preprocessor supports three modes of operation: in the first mode, state per transfer, the preprocessor clocks the logic analyzer only when a bus transfer is complete. In this mode, wait and halt states are filtered out. In the second mode, CLKOUT1 clocks the logic analyzer every time the microprocessor is clocked. This mode captures all bus states. An example application would be to locate memory locations that do not respond to requests for data. In the third mode, you can use the preprocessor to make timing measurements.

The JTAG TAP (test access port) controller can be monitored in realtime. TAP state can be viewed under the predefined label TAP.

B.3.2 Logic Analyzers Supported

The preprocessor 'C5x interface supports the following logic analyzers:

- HP 1650A/B
- HP 16510B
- HP 16511B
- HP 16540/41(A/D)
- HP 16550A
- HP 1660A/61A/62A
B.3.3 Pods Required

There are eight pod connectors on the preprocessor. Three are terminated and best used for state analysis, as all signals needed for disassembly are available. The other five connectors are not terminated and contain all processor signals, including a second set of the signals needed for disassembly. This allows you to double probe these signals, making simultaneous state and timing measurements.

B.3.4 Termination Adapters (TAs)

Of the eight pods, three are terminated. You may need to order up to five termination adapters, depending on how many pods are connected at the same time.

B.3.5 Availability

For more information and availability of the Hewlett-Packard E2442A, contact:

Hewlett-Packard Company
2000 South Park Place
Atlanta, GA 30339
(404) 980–7351
Application Reports and Designer’s Notebook Pages

This appendix lists the TMS320C5x application reports in Table C–1 and the TMS320C5x designer’s notebook pages (DNP) in Table C–2 available to you. To obtain a copy of any application report, call the Texas Instruments Literature Response Center at (800) 477–8924. When ordering, please identify the book by its title and literature number. To view a copy of any designer’s notebook pages, refer to the world-wide web site at:


Table C–1. TMS320C5x Application Reports

<table>
<thead>
<tr>
<th>Application Report Literature Number</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPRA030</td>
<td>Calculation of TMS320C5x Power Dissipation</td>
</tr>
<tr>
<td>SPRA033</td>
<td>Telecommunications Applications With the TMS320C5x DSPs</td>
</tr>
<tr>
<td>SPRA052</td>
<td>PCMCIA TMS320 DSP Media Card</td>
</tr>
<tr>
<td>SPRA054</td>
<td>Use of the TMS320C5x Internal Oscillator With External Crystals or Ceramic Resonators</td>
</tr>
<tr>
<td>SPRA057</td>
<td>Enhanced Control of an Alternating Current Motor Using Fuzzy Logic and a TMS320 Digital Signal Processor</td>
</tr>
<tr>
<td>SPRA085</td>
<td>Improving 32-Channel DTMF Decoders Using the TMS320C5x</td>
</tr>
</tbody>
</table>
### Table C-2. TMS320C5x Designer’s Notebook Pages

<table>
<thead>
<tr>
<th>DNP Number</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Optimizing Control Algorithms on 'C5x</td>
</tr>
<tr>
<td>6</td>
<td>'C5x EVM Provides for Audio Processing</td>
</tr>
<tr>
<td>10</td>
<td>Initializing the Fixed-Point EVM’s AIC</td>
</tr>
<tr>
<td>15</td>
<td>Efficient Coding on the TMS320C5x</td>
</tr>
<tr>
<td>19</td>
<td>Dual-Access Into Single-Access RAM on a 'C5x Device</td>
</tr>
<tr>
<td>21</td>
<td>TMS320C5x Interrupts</td>
</tr>
<tr>
<td>24</td>
<td>TMS320C5x Interrupt Response Time</td>
</tr>
<tr>
<td>25</td>
<td>TMS320C2x/C5x EVM AIC Initialization and Configuration</td>
</tr>
<tr>
<td>35</td>
<td>TMS320C5x Interrupts and the Pipeline</td>
</tr>
<tr>
<td>39</td>
<td>Bootload of C Code for the TMS320C5x</td>
</tr>
<tr>
<td>41</td>
<td>Supporting External DMA Activity to Internal RAM for TMS320C5x</td>
</tr>
<tr>
<td>42</td>
<td>Binary Search Algorithm on the TMS320C5x</td>
</tr>
<tr>
<td>43</td>
<td>Random Number Generation on a TMS320C5x</td>
</tr>
<tr>
<td>45</td>
<td>Fast TMS320C5x External Memory Interface</td>
</tr>
<tr>
<td>46</td>
<td>TMS320C5x Memory Paging (Expanding its Address Reach)</td>
</tr>
<tr>
<td>47</td>
<td>TMS320C5x Clock Modes</td>
</tr>
<tr>
<td>48</td>
<td>TMS320C5x Wait States</td>
</tr>
<tr>
<td>49</td>
<td>Clocking Options on the TMS320C5x</td>
</tr>
<tr>
<td>50</td>
<td>TMS320C5x DSK Analog I/O</td>
</tr>
<tr>
<td>54</td>
<td>Accessing TMS320C5x Memory-Mapped Register in C–C5xREGS.H</td>
</tr>
<tr>
<td>55</td>
<td>C Routines for Setting Up the AIC on the TMS320C5x EVM</td>
</tr>
<tr>
<td>57</td>
<td>Initializing the TMS320C5x DSK Board</td>
</tr>
<tr>
<td>59</td>
<td>Designing Macros for the TMS320C5x</td>
</tr>
<tr>
<td>63</td>
<td>Shared Memory Interface with a TMS320C5x DSP</td>
</tr>
<tr>
<td>65</td>
<td>Interfacing External Memory to the TMS320C5x DSK</td>
</tr>
<tr>
<td>66</td>
<td>Interfacing a TMS320C2x, 'C2xx, 'C5x DSP to a TLC548 8-bit A/D Convertor</td>
</tr>
</tbody>
</table>
### Table C–2. TMS320C5x Designer's Notebook Pages (Continued)

<table>
<thead>
<tr>
<th>DNP Number</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>67</td>
<td>Interfacing a TMS320C2x, 'C2xx, 'C5x DSP to an 8-bit Boot EEPROM</td>
</tr>
<tr>
<td>68</td>
<td>Using the Circular Buffer on the TMS320C5x</td>
</tr>
<tr>
<td>72</td>
<td>Interfacing Two Analog Interface Circuits to One TMS320C5x Serial Port</td>
</tr>
<tr>
<td>74</td>
<td>Reading a 16-bit Bus With the TMS320C5x Serial Port</td>
</tr>
<tr>
<td>76</td>
<td>Interfacing 20-MSPS TLC5510 Flash A/D Converter to TMS320C2xx and TMS320C5x Fixed-Point DSPs</td>
</tr>
<tr>
<td>77</td>
<td>IDLE2 Instruction on a TMS320C51 When Using a Divide-by-One Clock Option</td>
</tr>
<tr>
<td>78</td>
<td>Initializing the TLC32046 AIC on the TMS320C5x EVM Board</td>
</tr>
<tr>
<td>79</td>
<td>Initializing the TLC320040 AIC on the TMS320C5x DSK</td>
</tr>
<tr>
<td>81</td>
<td>Setting up and Simulating Interrupts on the TMS320C5x</td>
</tr>
</tbody>
</table>
A

adaptive filtering  2-49 to 2-51
ADC devices  4-20
ADTV  4-18
analog interface
  converters  4-4
  peripherals  4-1
application reports  C-1
application-oriented operations  2-46 to 2-56
  adaptive filtering  2-49
  dynamic programming  2-55
  infinite impulse response (IIR) filtering  2-52
  modern application  2-46
applications  1-4
architecture  1-5
arithmetic
  extended-precision  2-26
  floating-point  2-39
assistance  B-3

B

backtracking algorithm  2-56
BCND example  2-13
bit-reversal addressing  2-61
BLDD example  2-21
BLDP example  2-21
block moves  2-20 to 2-22
block repeat  2-20
BLPD example  2-21
boot loader  3-2
BSAR example  2-44
buffered signals, JTAG  A-10
buffers, circular  2-17
bus protocol  A-3

C

C bit  2-26, 2-29
'C14 to TLC32071 interface  4-13
'C25 to TLC32047 interfacing  4-3
'C5x
  applications  1-4
  characteristics  1-6
  key features  1-7
  overview  1-5
cable, target system to emulator  A-1 to A-14
circular buffer registers  2-17 to 2-19
delayed branches  2-24
designer's notebook pages  C-2
development tool nomenclature  B-6
development tools  B-2
device nomenclature  B-5
digital audio  4-19
digital electronics applications  4-18
  advanced television system design  4-18
digital electronics-related devices  4-20
direct memory access  3-7
DS pin  3-4

Index
Index

dynamic bit manipulation register (DBMR) 2-9
dynamic programming 2-55 to 2-70

E
emulator A-1
  buffered signals A-10
  bus protocol A-3
  cable header A-2
  cable pod A-4
  designing the JTAG cable A-2
  header signals A-2
  signal buffering A-9 to A-10
  signal timings A-6
  timing A-11
  timings A-6
  unbuffered signals A-9
emulator cable pod, interface A-5
extended-precision arithmetic 2-26 to 2-38
  addition 2-26 to 2-28
  division 2-35 to 2-46
  multiplication 2-30 to 2-34
  subtraction 2-29 to 2-30
external interfacing
  global data memory 3-6
  I/O space 3-6
  local data memory 3-4
  program memory 3-2
external memory interface 3-2, 3-4, 3-5

F
fast Fourier transforms (FFT) 2-57 to 2-70
filters
  adaptive 2-49
  FIR 2-50
  IIR 2-52 to 2-54
  switched capacitor 4-4
floating-point, arithmetic 2-39 to 2-45
fractional division 2-35 to 2-37

G
global data memory, external interfacing 3-6
global memory allocation register (GREG) 3-6

H
hardware development tools B-2, B-7
hardware stack 2-5
Harvard architecture 1-5
HDTV 4-18
Hewlett-Packard interface B-8

I
I/O, external interfacing 3-6
I/O space, external interfacing 3-6
IACK pin 3-2
IEEE 1149.1 A-3
infinite impulse response (IIR) filtering 2-52 to 2-54
initialization, processor 2-2
initialization routine 2-69
instruction conditions
  branch 2-12
  call 2-12
  return 2-12
integer division 2-35 to 2-37
interfacing memories
  EPROM 3-3
  global data memory 3-6
  RAM 3-4, 3-5
interrupt service routine (ISR) 2-5
interrupt trap 2-5
interrupts 2-5 to 2-7
INTR example 2-6
introduction 1-1
  TMS320 family overview 1-2
  TMS320C5x key features 1-7
  TMS320C5x overview 1-5

J
JTAG A-1
  signals A-3
JTAG emulator
  buffered signals A-10
  connection to target system A-1 to A-14
  no signal buffering A-9

K
key features 1-7 to 1-9
Index

L
LMMR example 2-23
local data memory, external interfacing 3-4, 3-5
logical and arithmetic operations 2-9 to 2-16
   matrix multiplication 2-14
   multiconditional branch 2-12 to 2-13
   parallel logic unit (PLU) 2-9 to 2-11
   search algorithm 2-14
LTD example 2-54

M
MACD example 2-53
matrix multiplication 2-14
memory interface 3-2, 3-4
modem AFE data converters 4-15
modem application 2-46 to 2-48
modem applications 4-15
MPYA example 2-54
multiconditional branch 2-12
multimedia applications 4-2
   multimedia-related devices 4-4
   system design 4-2
multiplication
   16-bit algorithm 2-31
   32-bit algorithm 2-32
multiprocessor configuration A-8

N
nested loops 2-14, 2-16
nomenclature B-4, B-5

P
parallel logic unit (PLU) 2-9
part numbers, tools B-7
part-order information B-4
PC environment (DMA) 3-7
preprocessor interface B-8
processor initialization 2-2 to 2-4
program memory, external interfacing 3-2
PS pin 3-2

R
RD pin 3-2, 3-4
repeat counter register (RPTC) 2-20
repeat loops 2-20 to 2-23
RMS routine 2-24
RPT example 2-21, 2-22, 2-23
RS pin 2-2

S
SATH example 2-40
SATL example 2-40
search algorithm 2-14
seminars B-3
servo control-related devices 4-14
servo control/disk drive applications 4-12
   servo control/disk drive-related devices 4-14
   system design 4-12
shadow registers 2-5
signals
   buffered A-10
   buffering for emulator connections A-9 to A-10
SMMR example 2-22
software applications
   application-oriented operations 2-46
   circular buffers 2-17
   extended-precision arithmetic 2-26
   fast Fourier transforms (FFT) 2-57
   floating-point arithmetic 2-39
   interrupts 2-5
   logical and arithmetic operations 2-9
   processor initialization 2-2
   single-instruction repeat loops 2-20
   software stack 2-8
   subroutines 2-24
software development tools B-2, B-7
software stack 2-8
speech encoding 4-3
speech memories 4-11
speech synthesis applications 4-10
   speech synthesis-related devices 4-11
   system design 4-10
stack, hardware 2-5
status register 1 (ST1), C bit 2-26, 2-29
STRB pin 3-2, 3-4
SUBC example 2-37, 2-38
subroutines 2-24 to 2-25
support tools
  development B-6
  device B-6
  nomenclature B-4

target system, connection to emulator A-1 to A-14
target system clock A-7
TBLR example 2-22
TBLW example 2-22
technical training B-3
telecommunications applications 4-5
  DSP/combo interface 4-7
  system design 4-6
  telecommunications-related devices 4-8
third-party support B-2
timing
  combo interface 4-8
  emulator A-11
TMS320
  advantages 1-2
  development 1-2
  evolution 1-3
  family overview 1-2
  history 1-2
  roadmap 1-3
  typical applications 1-4
TMS320C5x
  applications 1-4
  characteristics 1-6
  key features 1-7
    compatibility 1-7
    CPU 1-8
    instruction set 1-8
    memory 1-7
    on-chip peripherals 1-9
    packages 1-9
    power 1-7
    program control 1-8
    speed 1-7
    test/emulation 1-9
    overview 1-5
tools, part numbers B-7

V
  V.32 encoder example 2-47
  video signal processing 4-19
  voice synthesizers 4-10

W
  WE pin 3-2, 3-4
  workshops B-3

X
  XC example 2-24
  XDS510 emulator A-1 to A-14
IMPORTANT NOTICE

Texas Instruments (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 the specifications applicable at the time of sale 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.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage (“Critical Applications”).

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer’s applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

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.

Copyright © 1996, Texas Instruments Incorporated