TEXAS INSTRUMENTS

TMS320C30 Evaluation Module

Technical Reference

1990

Microprocessor Development Systems

TMS320C30 Evaluation Module Technical Reference

2547305-9721 revision B
October 1990

Texas Instruments

IMPORTANT NOTICE

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

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

TI assumes no liability for TI applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that 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.

Texas Instruments products are not intended for use in life-support appliances, devices, or systems. Use of a TI product in such applications without the written consent of the appropriate TI officer is prohibited.

Preface

Read This First

How to Use This Manual

This document contains the following chapters:

Chapter 1 **Introduction**
Provides an overview of the TMS320C30 EVM and its features.

Chapter 2 **The TMS320C30 EVM Board**
Describes the TMS320C30 Evaluation Module (EVM), its key components and how they operate, and its various interfaces.

Chapter 3 **TMS320C30/Host Communications Protocol**
Provides a protocol overview, tells how the system is initialized, explains how the command preprocessor functions, and shows how commands are executed.

Appendix A **Host Communications Software**
Contains the PC/AT C source programs for the host communications protocol.

Appendix B **TMS320C30 Communications Software**
Contains the TMS320C30 assembly source code for the EVM communications protocol.

Appendix C **TMS320C30 EVM Reduced PAL Equations**
Contains the reduced equations that were developed for programming the PAL devices that are used on the TMS320C30 EVM.

Appendix D **TMS320C30 Schematics**
Contains the schematics for the TMS320C30 EVM.

Appendix E **Glossary**

Related Documentation

The following TMS320C3x documents are available through Texas Instruments. To obtain a copy of any of these documents, call the Texas Instruments Response Center (CRC) at (800) 232-3200. When ordering, please identify the book by its title and its literature number.

- *TMS320C3x User's Guide*, literature number SPRU031.
- *TMS320C30 Assembly Language Tools User's Guide*, literature number SPRU035.
- *TMS320C30 C Compiler Reference Guide*, literature number SPRU034.
- *TMS320C30 C Source Debugger User's Guide*, literature number SPRU053.
- *TMS320C30 Emulator User's Guide*, literature number SPRU039.
- *TMS320C30 Simulator User's Guide*, literature number SPRU017.
- *TMS320 Family Development Support Reference Guide*, literature number SPRU011.
- *Digital Signal Processing Applications with the TMS320 Family: Theory, Algorithms, and Implementations: Volume 3*, literature number SPRA017.
- *Selected Application Notes Using the TMS320C30 Evaluation Module*, literature number SPRA021.

Information About Cautions and Warnings

This book contains cautions. A **caution** describes a situation that could potentially damage your software or equipment. It looks like the following:

> **This is what a caution looks like.**
>
> CAUTION

The information in a caution is provided for your protection. Please read each caution carefully.

Trademarks

Cypress is a trademark of Cypress Semiconductor Corporation.

Contents

1 **Introduction** **1-1**

Providers an overview of the TMS320C30 evaluation module (EVM) and its features.

1.1 General Description 1-2

1.1.1 Key Features 1-2

1.1.2 Functional Overview 1-2

2 **The TMS320C30 EVM** **2-1**

Describes the TMS320C30 EVM, its key components and how they operate, and its various interfaces.

2.1 The TMS320C30 EVM Board 2-2

2.2 Host I/O Map Requirements 2-3

2.2.1 EVM Memory Map Switch Settings 2-3

2.2.2 I/O Base Address Generation 2-4

2.3 Test Bus Controller (TBC) Interface 2-5

2.3.1 Interface Format 2-5

2.3.2 The Host/TBC Interface 2-5

2.3.3 External Events—EVT0 – EVT3 2-7

2.3.4 Data Transfer Synchronization 2-8

2.4 Host/TMS320C30 Communications Port Interface 2-10

2.4.1 TMS320C30 Interrupts 2-11

2.4.2 TMS320C30 Interrupt Generation and Detection 2-13

2.4.3 Host Event Polling 2-14

2.4.4 TMS320C30 Communications Port Mapping 2-15

2.5 System Resets 2-16

2.6 TMS320C30 Interfaces 2-17

2.6.1 TMS320C30 Memory Interface 2-17

2.6.2 Analog Interface Controller (AIC) 2-17

2.6.3 External Serial Port Interface 2-20

2.6.4 AIC Initialization 2-20

3 **TMS320C30/Host Communications Protocol** **3-1**

Provides a protocol overview, tells how the system is initialized, explains how the command processor functions, and shows how commands are executed.

3.1 Protocol Overview 3-2

3.2 System Initialization 3-4

3.3 The Command Preprocessor 3-5
3.3.1 Host/TMS320C30 Command Transfer and Formatting 3-5
3.3.2 Host Writes 3-5
3.3.3 Host Reads 3-5
3.3.4 TMS320C30 EVM Command Structure 3-6
3.3.5 Transferring and Formatting Count Parameters 3-6
3.4 Command Execution 3-7
3.4.1 Types of Data Transfers 3-7
3.4.2 Command Execution Examples 3-7

A Host Communications Software A-1

Contains the PC/AT C source programs for the host communications protocol.

A.1 TMS320C30 and TBC Parameters A-2
A.2 TMS320C30 EVM Host Routines A-4
A.3 Simple EVM Test Program A-6
A.4 EVM Reset A-7
A.5 EVM_memwrite A-8
A.6 EVM_memread A-9
A.7 EVM_cmdinit A-10
A.8 EVM_status A-11

B TMS320C30 Communications Software B-1

Contains the TMS320C30 assembly source code for the EVM communications protocol.

B.1 TMS320C30 Assembly Source Code B-2

C TMS320C30 EVM Reduced PAL Equations C-1

Contains the reduced equations that were developed for programming the PAL devices that are used on the TMS320C30 EVM.

C.1 Write Control PAL, UA3 .... C-2
C.2 Read Control PAL, UA4 .... C-4
C.3 I/O Control PAL, UA5 .... C-5
C.4 Interrupt Control PAL, UB5 .... C-7

D TMS320C30 EVM Schematics D-1

Contains the schematics for the TMS320C30 EVM.

E Glossary E-1

Figures

1–1. TMS320C30 Block Diagram 1-2
2–1. TMS320C30 EVM Board 2-2
2–2. Interrupt Timing Diagram 2-12

Tables

2–1. TMS320C30 EVM Switch Settings for I/O Address Space 2-3
2–2. Host I/O Memory Mapping for the Default Switch Setting 2-3
2–3. TMS320C30 EVM Register Mapping Offsets 2-4
2–4. TBC to TMS320C30 Emulation Port Connections 2-7
2–5. EVM Event Usage 2-7
2–6. Synchronized Write Parameters 2-8
2–7. Synchronized Read Parameters 2-9
2–8. TMS320C30 Reset Parameters 2-9
2–9. TMS320C30 Interrupt Flags 2-11
2–10. Interrupt Generator State Table 2-14
2–11. TMS320C30 Communications Port Mapping 2-15
2–12. TMS320C30 EVM Register Mapping Offsets 2-16
2–13. TMS320C30/TLC32044 Interface Connections 2-17
2–14. TMS320C30 EVM Analog I/O Characteristics 2-19
2–15. J5 Pinouts 2-20
2–16. AIC Setup Values 2-21

Examples

2–1. I/O Address Generation 2-4
2–2. Write Interrupt Generator Reduced Equations 2-13
2–3. Interrupt Generator States 2-13
3–1. 16-Bit Writes — hwrite16 3-8
3–2. 16-Bit Reads — hread16 3-8
3–3. 32-Bit Writes — hwrite32 3-8
3–4. 32-Bit Reads — hread32 3-9
3–5. 16-Bit DMA Writes — dmawrite 3-9
3–6. 16-Bit DMA Reads — dmaread 3-10

Chapter 1

Introduction

The **TMS320C30 Evaluation Module (EVM)** is a development tool that lets you execute and debug your applications program by using the TMS320C30 C Source Debugger. When you connect analog input and output (such as a microphone and a speaker) to the system, the EVM becomes a simple signal analysis tool. You can also transfer your analog data to and from the host PC through the EVM's 16-bit communications port.

The sheer power of the TMS320C30 makes it a popular device for designing high-performance systems that use megabytes of memory and elaborate communications subsystems. The same features that make the TMS320C30 well suited for these high-end applications also make it appropriate for products like the EVM, which meets the goal of providing medium to high performance with minimal logic and a low product cost.

The TMS320C30 EVM also sets the stage for a new era of emulation support by using **embedded in-system emulation** (EISE).

An unprecedented amount of computing power has been packaged on a convenient PC half-card at a price that eliminates all cost barriers for evaluating the TMS320C30.

This manual decribes the TMS320C30 EVM, its features, design details, and the associated application interface software. Use this manual in conjunction with the *TMS320C30 C Source Debugger User's Guide*, literature number SPRU053, and associated addenda for instructions on installing and setting up the EVM.

1.1 General Description

1.1.1 Key Features

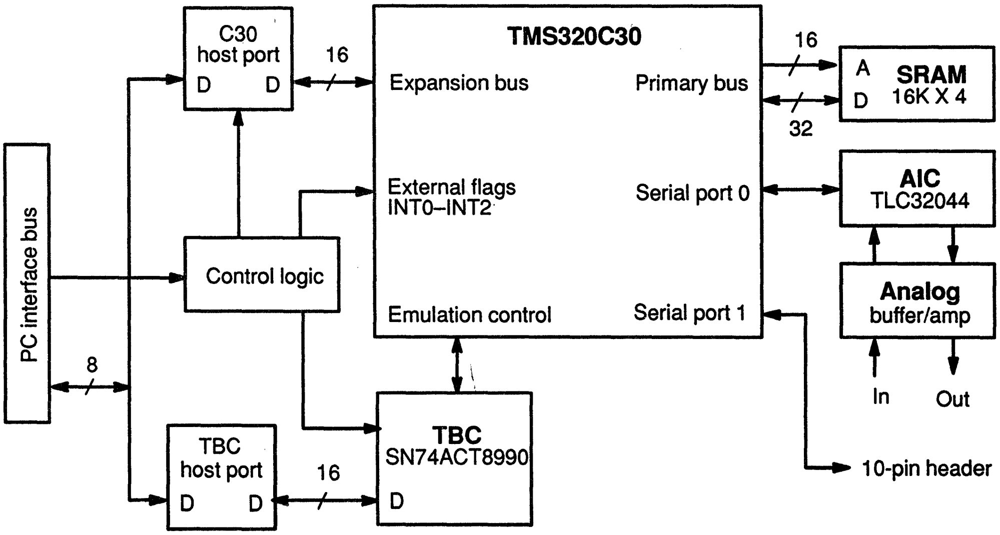
- ❑ The industry-standard 33-MFLOP TMS320C30 floating-point DSP
- ❑ 16K words of zero wait-state SRAM on the primary bus
- ❑ Voice quality analog data acquisition via the TLC32044 (Analog Interface Circuit)
- ❑ Standard RCA jacks for line-level analog input and output
- ❑ External serial port
- ❑ 16-bit bidirectional PC host communications port
- ❑ Embedded emulation support via the 74ACT8990 Test Bus Controller
- ❑ IBM PC/AT compatible 8-bit half card, mappable in one of four I/O locations
- ❑ Requires approximately 1 amp of current at 5 volts

1.1.2 Functional Overview

Figure 1–1 shows the basic block diagram and interconnects of the TMS320C30 EVM. The interconnects include the emulation interface, the host interface, an analog interface, a serial port interface, and memory.

All code for the EVM is loaded through the TMS320C30 emulation port. There is no onboard boot PROM/EPROM or direct host access into the TMS320C30 memory. Once code has been loaded, the host and TMS320C30 communicate by means of a shared bidirectional 16-bit register.

Figure 1–1. TMS320C30 Block Diagram



The register-based interface between the host and TMS320C30 is simple, keeps costs low, and provides a moderate transfer band-width of approximately 200K bytes per second (KBPS). This interface has been tailored for use with the TMS320C30 DMA channel and can be fully interrupt driven.

The EVM's analog section consists of a TLC32044 Analog Interface Circuit (AIC) coupled to an LM386 low-noise audio power amplifier for output drive, and a TL072 input amplifier. The input and output gain is fixed and supports standard line-level audio input/output voltage levels. The analog input/output is brought to the external world by two standard RCA jacks (J3 and J4).

The AIC interfaces to the TMS320C30 by means of serial port 0. Serial port 1 is brought out to a 10-pin connector for external use.

The TMS320C30 interfaces directly to 16K words of 35-ns SRAM (Cypress™ CY7C164-35VC). This supports zero wait-state memory accesses on the primary bus.

A unique feature of the TMS320C30 EVM is **embedded emulation.** Embedded emulation is defined as the ability to provide emulation support within the target system. This ability is enabled by the SN74ACT8990 test bus controller (TBC) and the TMS320C30 emulation port.

Chapter 2

The TMS320C30 EVM

This chapter describes the TMS320C30 Evaluation Module (EVM), its key components and how they operate, and its various interfaces.

These topics are discussed:

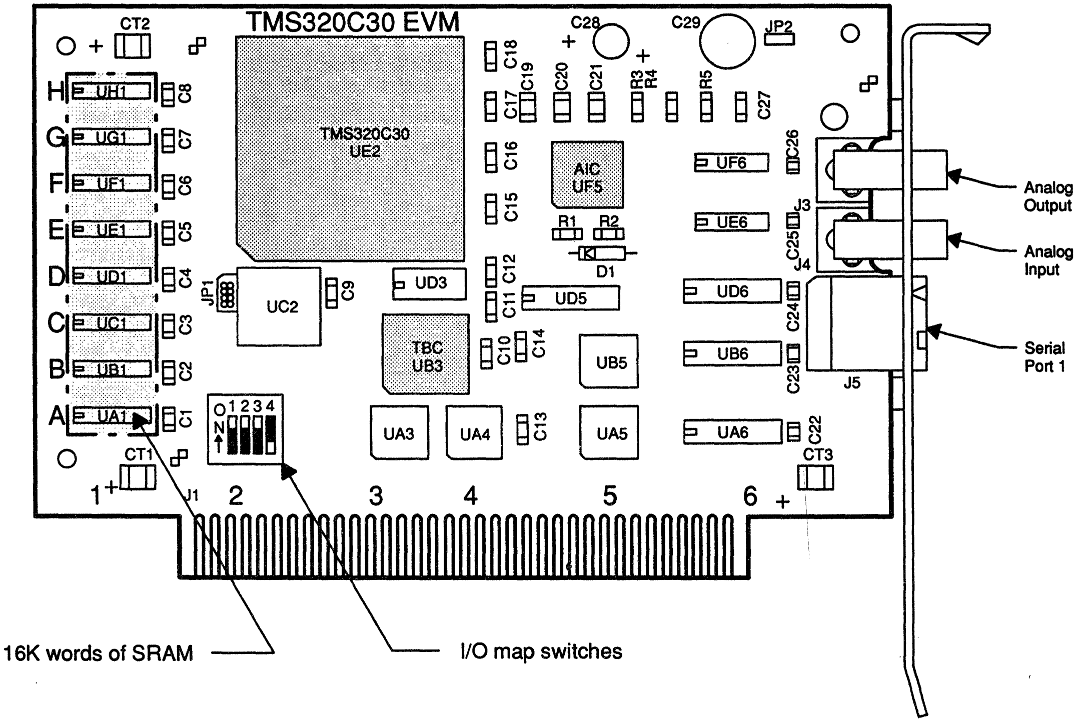
Section		Page
2.1	The TMS320C30 EVM Board	2-2
2.2	Host I/O Map Requirements	2-3
2.3	Test Bus Controller (TBC) Interface	2-5
2.4	Host/TMS320C30 Communications Port Interface	2-10
2.5	System Resets	2-16
2.6	TMS320C30 Interfaces	2-17

2.1 The TMS320C30 EVM Board

The TMS320C30 EVM board is a half-length board that installs in a vacant slot in your PC. The *TMS320C30 C Source Debugger User's Guide* and its addenda contain instructions for installing and setting up the EVM.

Figure 2–1 shows the layout of the TMS320C30 EVM board. The face of the mounting bracket is labeled OUT and IN to correspond with the analog output (J3) and input (J4) jacks.

Figure 2–1. TMS320C30 EVM Board



2.2 Host I/O Map Requirements

The TMS320C30 EVM resides in the host I/O address space. It requires three 32-byte pages for a total of 96 bytes. Each page is 1K-bytes apart. PC/AT compatible machines decode only the first 1K of I/O space; thus, the three pages appear to be mapped on top of one another.

Note:

The TMS320C30 EVM will not work in systems that do not conform to the 1K I/O decoding scheme.

2.2.1 EVM Memory Map Switch Settings

You can map the EVM into one of four I/O base address ranges by setting switches 1 and 2 as shown in Table 2–1. Although their settings are shown, switches 3 and 4 are reserved for factory testing and should not be set to any position other than the default settings. Check your PC system documentation to make sure that the I/O space selected does not conflict with other I/O devices, such as disk controllers, local area network controllers, etc.

Table 2–2 shows the host I/O map usage for the default switch settings, and Table 2–3 shows TMS320C30 EVM register address offsets.

Table 2–1. TMS320C30 EVM Switch Settings for I/O Address Space

	Switches				
I/O Address Space	**SW1-1**	**SW1-2**	**SW1-3**	**SW1-4**	
0x0240 – 0x025F	ON	ON	ON	OFF	Default setting
0x0280 – 0x029F	ON	OFF	ON	OFF	
0x0320 – 0x033F	OFF	ON	ON	OFF	
0x0340 – 0x035F	OFF	OFF	ON	OFF	

Table 2–2. Host I/O Memory Mapping for the Default Switch Setting

I/O Address Space	**Function**	**I/O Page**
0x0240 – 0x025F	TBC	base
0x0640 – 0x065F	TBC	overlay 1
0x0A40 – 0x0A5F	Control/communications port	overlay 2

Table 2–3. TMS320C30 EVM Register Mapping Offsets

Register	Offset	Size	Access
RESERVED	0x0000 – 0x0008	——	——
CONTROL5	0x000A	16	R/W
RESERVED	0x000C – 0x0012	——	——
MINOR_CMD	0x0014	16	R/W
RESERVED	0x0016 – 0x001E	——	——
STATUS 0	0x0400	16	R
RESERVED	0x0402 – 0x041F	——	——
COM_CMD	0x0800	8	R/W
COM_DATA	0x0808	16	R/W
SOFT_RESET	0x0818	0	W

2.2.2 I/O Base Address Generation

The host accesses the TMS320C30 EVM registers over the PC/AT I/O expansion bus with 80x86 input and output instructions. The address of a given register is defined as an I/O base address plus an offset. Example 2–1 shows a program that generates a physical I/O address in Microsoft C code.

Example 2–1. I/O Address Generation

```
#define  outport      outpw
#define  COM_DATA     0x0808
unsigned short iobase 0x0240

{
    outport(iobase + COM_DATA, 0X1234);
}
```

2.3 Test Bus Controller (TBC) Interface

The most significant aspect of the TMS320C30 EVM is the emulation support for the TMS320C30 that is embedded into the target system. The SN74ACT8990 test bus controller (TBC) and the TMS320C30 emulation port enable this novel approach to emulation. There are several related features:

❑ Emulation is supported *without* external cabling, monitor software, or consumption of user resources.

❑ Easy access to the TMS320C30 supports high-level language (HLL) debuggers, factory testing, field diagnostics, etc.

❑ System boot ROMs are not needed — the host can download all necessary program or data through the emulation port.

2.3.1 Interface Format

The TBC provides a means for the host processor to communicate with a target device through the device emulation port. This is done with one of the following formats:

1) The Texas Instruments modular port scan device (MPSD) format, or

2) The industry standard IEEE 1149.1 JTAG format.

The TMS320C30 uses the MPSD format.

2.3.2 The Host/TBC Interface

The host interface to the TBC is quite simple and consists of two registered bidirectional SN74ALS652 data transceivers (UA6 and UB6) and two TIB-PAL16L8-15 PALS (UA3 and UA4). Appendix D contains the schematics that show logic implementation.

2.3.2.1 8-Bit to 16-Bit Conversion

UA6 and UB6 are used to convert host *8-bit byte* accesses to TBC *16-bit word* accesses. This is unique because the TMS320C30 EVM has an 8-bit host port interface and the TBC is a 16-bit device.

❑ UA6 is the low-byte transceiver.

❑ UB6 is the high-byte transceiver.

The SN74ALS652 defines the data as follows:

❑ *Stored data* is data that is held in its internal register.

❑ *Real-time data* is data that is present at its input pins.

On host writes, the first byte (LSByte) is clocked into UA6. When the second byte (MSByte) is written, a 16-bit access to the TBC is generated. The data to the TBC is the *stored data* from UA6 and *real-time data* from UB6.

On host reads, real-time data is taken from UA6, and stored data is taken from UB6 — just the opposite from a host write. For the host, the LSByte is read first from UA6 followed by the MSByte from UB6.

These actions are summarized as follows:

1) The host must always perform 16-bit accesses to the TBC.

2) Data is written to the TBC on the MSByte access.

3) On host writes, stored data comes from UA6; real-time data comes from UB6.

4) On host reads, real-time data comes from UA6; stored data comes from UB6.

5) Data is read from the TBC on the LSByte access.

2.3.2.2 Host Interface PAL logic

The host interface PAL logic consists of a TIBPAL16L8-15 address decoder PAL (UA3) and a TIBPAL16L8-15 control signal generator PAL (UA4).

- UA3 functions as the WRITE_CONTROL register.

- UA4 functions as the READ_CONTROL register.

UA3 uses the address and control signals on the host PC/AT bus and the I/O space SW1 switch settings (see Table 2–2) to generate the appropriate $\overline{\text{PERSEL}}$ and $\overline{\text{SBMSEL}}$ select signals.

These select signals combine with the host I/O read/write ($\overline{\text{HIORD}}$,$\overline{\text{HIOWR}}$) control signals to generate the TBC read/write ($\overline{\text{SBMRD}}$,$\overline{\text{SBMWR}}$) signals.

Sections C.1 and C.2 of Appendix C contain the PAL source equations required for programming the PAL devices, and Appendix D contains the TMS320C30 EVM schematics.

2.3.2.3 TBC to TMS320C30 Emulation Port Connections

The TBC provides a direct connection to the TMS320C30 emulation port. Table 2–4 shows these connections.

Table 2–4. TBC to TMS320C30 Emulation Port Connections

TBC	TMS320C30
TMS0	EMU0
TMS1	EMU1
TDO	EMU2
TDI0	EMU3
TCLKIN	H1

Note:

The TMS320C30 H1 clock serves as the emulation clock for embedded applications. This differs from the TMS320C30 XDS500 emulator, which uses H3 as the clock. This is because of TBC signal timing requirements.

2.3.3 External Events—EVT0 – EVT3

The TBC supports four external events, EVT0 – EVT3. They can be programmed as interrupt sources to the TBC or general purpose bit I/O. Table 2–5 shows how the EVM uses these signals.

Table 2–5. EVM Event Usage

Signal	Function	How Used
EVT0	Embedded emulation support.	EVT0 is used for embedded emulation support. **Do not modify.**
EVT1	Host read acknowledge.	EVT1 is a communications flag that is set when the TMS320C30 writes to the communications register. When set, the host can read valid data. The host must clear EVT1 before each communications register read.
EVT2	Host write acknowledge.	EVT2 is a communications flag that is set when the TMS320C30 reads the host communications register. When set, the host can write the next word of data to the communications register. The host must clear EVT2 before each communications register write.
EVT3	TMS320C30 reset source.	EVT3 is used to set and hold the TMS320C30 in reset. The EVM powers up with the TMS320C30 held in reset. Setting EVT3 asserts the TMS320C30 reset pin; clearing EVT3 deasserts the TMS320C30 reset pin. The inverted value of EVT3 drives the TMS320C30 reset pin.

The event pins are set up with the `EVM_reset` function that is provided in Section A.4 of Appendix A.

2.3.4 Data Transfer Synchronization

Data transfer synchronization between the TMS320C30 and the host is an integral part of the communications protocol. See Chapter 3 for details about the communications protocol between the TMS320C30 and the host.

TBC events EVT1 and EVT2 provide the synchronization mechanism. Once EVT1 and EVT2 have been set up with the `EVM_reset` function, they can be polled and cleared to provide synchronization.

Although TI recommends event polling, polling is not required. The data transfer rate with polling is approximately 200 kilobytes per second (KBPS). The transfer rate more than doubles if you do not poll. However, you should analyze your application code that is running on both the TMS320C30 and the host to ensure that data is not lost in unsynchronized transfers.

Remember that the primary purpose for the TBC is to support embedded in-system emulation (EISE). Exercise caution when accessing the TBC. Improper usage can adversely affect TMS320C30 operation. Access the TBC only with the software routines provided in this manual or with compatible routines.

2.3.4.1 Steps for Synchronized Writes

Execute the following steps to synchronize writes to the TMS320C30 EVM communications register. Table 2–6 shows the synchronized write parameters for each step.

Step 1: *Clear* the previous EVT2 write acknowledge.

Step 2: *Write* the data into the communications register (`COM_DATA`).

Step 3: *Update* the TBC STATUS0 register.

Step 4: If the EVT2 write acknowledge bit is set, go to step 1; otherwise, go to step 3.

Table 2–6. Synchronized Write Parameters

Step	Register Address Offset	Register Value	Register Name	Access
1	0x0014	0x0004	MINOR_CMD	W
2	0x0808	data value	COM_DATA	W
3	0x0014	0x6044	MINOR_CMD	W
4	0x0400	0x0004 (mask)	STATUS0	R

2.3.4.2 Steps for Synchronized Reads

Execute the following steps to synchronize reads to the TMS320C30 EVM communications register. Table 2–7 shows the synchronized read parameters for each step.

The steps for synchronized reads to the TMS320C30 EVM communications register follow:

Step 1: *Update* the TBC status register.

Step 2: If the EVT1 read acknowledge bit is set, proceed to step 3; otherwise, go to step 1.

Step 3: *Clear* the current EVT1 read acknowledge.

Step 4: *Read* the data from the communications register (`COM_DATA`).

Table 2–7. Synchronized Read Parameters

Step	Register Address Offset	Register Value	Register Name	Access
1	0x0014	0x6044	MINOR_CMD	W
2	0x0400	0x0002 (mask)	STATUS0	R
3	0x0014	0x0002	MINOR_CMD	W
4	0x0808	data value	COM_DATA	R

2.3.4.3 Using the TBC to Reset The TMS320C30

UA5 is a TIBPAL16L8-15 PAL that does IO_CONTROL functions. It also inverts the EVT3 signal and generates the TMS320C30 reset signal when EVT3 is active.

Execute the following steps to reset the TMS320C30. Table 2–8 shows the reset parameters for each step.

Step 1: *Set* EVT3 = 1.

Step 2: *Set* EVT3 = 0.

Table 2–8. TMS320C30 Reset Parameters

Step	Register Address Offset	Register Value	Register Name	Access
1	0x000A	0x0808	CONTROL5	W
2	0x000A	0x0800	CONTROL5	W

2.4 Host/TMS320C30 Communications Port Interface

In general, host communications to a system such as the TMS320C30 EVM are provided by means of FIFOs, dual-port SRAMs, or direct interface to EVM memory space. These interfaces have their strengths; however, they consume a significant amount of hardware and printed wiring board (PWB) real estate. Further, they are fairly expensive, depending on the implementation.

The TMS320C30 EVM has a simpler host port interface. This reduces product cost, while maintaining moderate throughput (up to 200 KBPS). One of the main reasons for having elaborate host interfaces in a system such as the EVM is for program loading. With the TBC handling program loading chores, the interface requirements reduce to those of passing data to and from the application software.

On the EVM, a 16-bit bidirectional register-based host interface was implemented to meet these interface requirements. This interface is similar to that of the TBC interface.

- UD5 is the low-byte half of the 16-bit communications register.
- UD6 is the high-byte half of the 16-bit communications register.

The major difference between the TBC and TMS320C30 interface schemes is that communications to the TMS320C30 can be interrupt driven. This means that host accesses to the communications register will generate interrupts to the TMS320C30 by means of the INT0 – INT2 interrupt flags (see Table 2–9). TMS320C30 accesses to the communications registers generate the TBC EVT1–EVT2 event flags, which the host polls.

Table 2–9. TMS320C30 Interrupt Flags

Flag	Function	How Used
INT0	Command interrupt.	The INT0 flag indicates that the host has deposited a command into the communications register. It is generated on byte writes to the COM_CMD register. Byte reads to the COM_CMD register do not generate an INT0 flag to the TMS320C30. This allows the host to poll the command register for commands/ status from the TMS320C30.
INT1	Data write interrupt.	The INT1 flag indicates that the host has written data to the communications register. It also requests the TMS320C30 to read data from the communications buffer. An INT1 flag is generated on 16-bit host writes to the COM_DATA register.
INT2	Data read interrupt.	The INT2 flag indicates that the host has read data from the communications register. It also requests the TMS320C30 to write the next data word to the communications register. An INT2 flag is generated on 16-bit host reads from the COM_DATA register.

2.4.1 TMS320C30 Interrupts

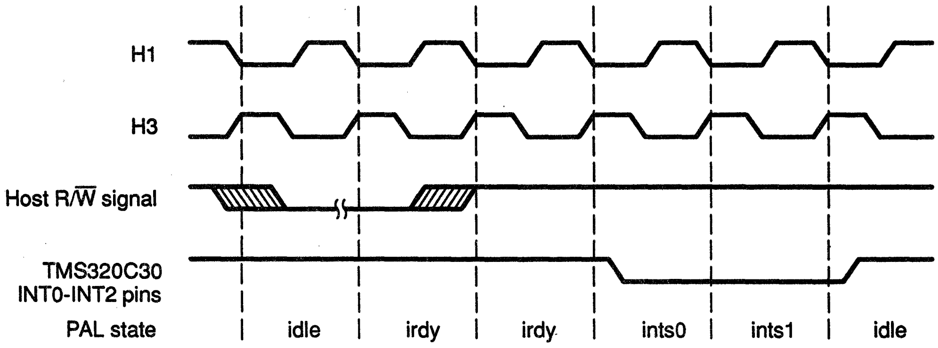
When you implement an interrupt-driven interface, you must decide whether to have software- or hardware-generated (set and clear) interrupts.

The TMS320C30 EVM uses **hardware-driven** interrupts on the TMS320C30 side of the communications register to maximize performance. This allows the TMS320C30 DMA controller to service data read/write interrupts.

Software-driven interrupts cannot be used by the DMA controller, because the TMS320C30 DMA does not execute code that clears the interrupts.

To produce external interrupt signals, a TIBPAL16R6 (UB5) monitors accesses to the communications register. When a proper access is generated, UB5 outputs an interrupt pulse to the TMS320C30. The pulse duration is two H3 clock cycles. Figure 2–2 shows the timing relationships for the interrupts.

Figure 2–2. Interrupt Timing Diagram



Before an interrupt is generated, the interrupt generator waits for a host communications register read/write signal to go active and then inactive. This ensures

- That data in the register is properly read or written before an interrupt is generated, and
- That only one interrupt is generated per host access.

The only restriction, for proper synchronization and detection, is that the host I/O read/write signal high and low time must be at least two H3 cycles in duration. The PC/AT specification for 8-bit I/O devices easily meets this criterion.

2.4.2 TMS320C30 Interrupt Generation and Detection

There are three key points regarding TMS320C30 interrupt generation and detection.

1) TMS320C30 interrupts are level sensitive, not edge sensitive.
2) Interrupts are detected on the falling edge of H1. Thus, interrupts must be set up and held to the falling edge of H1 for proper detection.
3) To get only one interrupt, an interrupt pulse must be set up and held to at least one H1 falling edge but no more than two H1 falling edges. The TMS320C30 can accept an interrupt from the same source every two H1 clock cycles.

The TMS320C30 EVM uses the H3 clock to clock the interrupt generator (UB5). This provides approximately one full clock cycle of setup time (minus a PAL delay) to the TMS320C30 H1 falling edge.

The rising edge of H3 lags the falling edge of H1 by 0 – 5 ns. This helps meet the minimum hold time requirement for interrupts (0 ns hold time from H1 low). Appendix C.4 contains the PAL source for UB5. Note that each of the three interrupt generators (cntlint, rdint, and wrint) is implemented as a four-state (two flip-flops) state machine with an asynchronous input.

Driving a state machine with an asynchronous input is generally a bad design practice because the flip-flops may detect the input at different levels. This causes the state machine to operate improperly. However, the state transitions for each interrupt generator, as implemented in this design, were chosen to prevent improper operation.

Example 2–2 and Example 2–3 show the equations that were formulated for the write interrupt generator state machine by using DATA I/O ABEL version 3.2 with a reduction level of 3.

Example 2–2. Write Interrupt Generator Reduced Equations

The reduced equations for the write interrupt generator are as follows:

```
wrint  := !( (!s1 & !wrint) # (!s1     & wclk3_) )
s1     := !( (!s1 & wrint_) # (!wclk3_ & wrint ) )
```

where: wrint is the interrupt output
s1 is a state output
wclk3_ is the interrupt source input

Example 2–3. Interrupt Generator States

The states for the interrupt generator state machine are as follows:

idle => [wrint = 1, s1 = 1]
irdy => [wrint = 1, s1 = 0]
ints0 => [wrint = 0, s1 = 0]
ints1 => [wrint = 0, s1 = 1]

Table 2–10 shows the transitions from the current state to the next state for inputs on wclk3_. The state machine always behaves properly for the following reasons:

1) Only s1 is affected by a change in wclk3_ while the interrupt generator is in the **idle** state.

2) Only wrint is affected by a change in wclk3_ while the interrupt generator is in the **irdy** state.

3) States **ints0** and **ints1** are not affected by a change in wclk3_.

Table 2–10. Interrupt Generator State Table

Input	Current State		Current State	Next State		Next State
wclk3_	**wrint**	**s1**	**Name**	**wrint**	**s1**	**Name**
0	1	1	idle	1	0	irdy
1	1	1	idle	1	1	idle
0	1	0	irdy	1	0	irdy
1	1	0	irdy	0	0	ints0
0	0	0	ints0	0	1	ints1
1	0	0	ints0	0	1	ints1
0	0	1	ints1	1	1	idle
1	0	1	ints1	1	1	idle

2.4.3 Host Event Polling

While the TMS320C30 side of the communications register is interrupt-driven, the host side is polled. When the TMS320C30 reads or writes the communications register, the TBC generates the EVT1 – EVT2 event flags. The host reads and clears these flags to provide synchronization.

There is little benefit in having the host side of the communications interface be interrupt-driven. In general, most data passed to and from the EVM is passed in blocks. Using host interrupts to service individual accesses to the communications register significantly reduces the data throughput.

2.4.4 TMS320C30 Communications Port Mapping

Table 2–3 showed the TMS320C30 EVM register address offsets mapping in the host address space. Table 2–11 shows the communications register mapping in the TMS320C30 address space. The COM_DATA register resides on the entire TMS320C30 I/O expansion bus.

Table 2–11. TMS320C30 Communications Port Mapping

Register Name	Address	Size	Access
COM_DATA	0x804000 – 0x805FFF	16	R/W

2.5 System Resets

System reset is essential for proper EVM operation. The EVM is reset by the host's $\overline{\text{RESET}}$ signal during system power up. However, you can also reset the EVM with software by writing to the SOFT_RESET I/O address location shown in Table 2–12.

Table 2–12. TMS320C30 EVM Register Mapping Offsets

Register	Offset	Size	Access
RESERVED	0x0000 – 0x0008	——	——
CONTROL5	0x000A	16	R/W
RESERVED	0x000C – 0x0012	——	——
MINOR_CMD	0x0014	16	R/W
RESERVED	0x0016 – 0x001E	——	——
STATUS 0	0x0400	16	R
RESERVED	0x0402 – 0x041F	——	——
COM_CMD	0x0800	8	R/W
COM_DATA	0x0808	16	R/W
SOFT_RESET	0x0818	0	W

Because a register does not exist at the SOFT_RESET address, decoding the I/O address and host I/O write signals provide a reset pulse to the TBC. The TBC places the TMS320C30 into a reset state by taking its EVT3 pin to a logic 1. EVT3 is inverted through PAL UA5 to generate a logic 0 on the TMS320C30 reset pin.

There are two reasons for implementing the TMS320C30 reset in this fashion:

1) It ensures that the TMS320C30 sees a valid reset level for at least ten H1 clocks.

2) It holds the TMS320C30 in reset during powerup.

This is important because the EVM does not have a boot ROM and could execute invalid code. Further, the TMS320C30 can be reset without resetting the TBC, because theTBC's EVT3 pin is under host software control. See subsection 2.3.4.3 on page 2-9 for more information about using the TBC to reset the TMS320C30.

2.6 TMS320C30 Interfaces

2.6.1 TMS320C30 Memory Interface

The EVM supports 16K words of zero wait-state SRAM on the TMS320C30 primary bus. The SRAMs chosen have a memory cycle time of 35 ns. To meet SRAM interface timings, the TMS320C30 device requires a 30-MHz input clock. The primary reason for using 35 ns SRAMs is cost.

The interface between the TMS320C30 and the SRAMs is a zero logic interface; there is no address decoding. This effectively maps the SRAM to 16K-word boundaries throughout the TMS320C30 primary bus address space.

2.6.2 Analog Interface Controller (AIC)

The EVM provides a single channel input/output voice quality analog interface. The Texas Instruments TLC32044 is used as the analog interface controller.

AIC features include the following:

- ❑ A single-chip D/A and A/D conversion with 14 bits of dynamic range, and
- ❑ Variable D/A and A/D sampling rate and filtering.

The AIC interfaces directly to the TMS320C30 serial port. Table 2–13 lists the interface connections between the TMS320C30 and the AIC.

Table 2–13. TMS320C30/TLC32044 Interface Connections

TMS320C30		TLC32044	
Signal	**I/O State**	**Signal**	**I/O State**
CLKX0	I	SCLK	O
DX0	O	DX	I
FSX0	I	$\overline{\text{FSX}}$	O
CLKR0	I	SCLK	O
DR0	I	DR	O
FSR0	I	$\overline{\text{FSR}}$	O
TCLK0	O	MCLK	I
XF0	O	$\overline{\text{RESET}}$	I

There are two key points regarding the TMS320C30/AIC interface:

1) The AIC master clock is derived from the TMS320C30 TCLK0 signal. TMS320C30 timer 0 is set up to operate in pulse mode with a pulse width = 1. This yields a 7.5-MHz TCLK0 signal with a 30-MHz TMS320C30 input clock frequency.

2) The TMS320C30 XF0 signal drives the AIC reset signal.

You could gain more sampling rate flexibility by externally clocking the AIC. However, this gain is not as great as the PWB real estate cost savings that result from using the TMS320C30 TCLK0 signal. In addition, this cost-effective approach supports an 8-kHz sampling rate that is commonly used in voice-quality applications.

The AIC accepts analog input and produces analog output but only within specific fixed voltage and power-level ranges. The analog circuitry, used with the AIC, is designed to allow the system to process signals that are often encountered in general-purpose signal processing.

AIC input circuitry comprises two stages of TL072 operational amplifiers:

1) Stage one (UE6A) is configured as a noninverting, high-input impedance, unity gain amplifier. Its purpose is to buffer the input signal from a microphone or some similar input source without presenting any appreciable load to the source.

2) Stage two (UE6B) is used to provide gain for the input signal. A nominal gain of two is used because input signal levels may range from several mV to several hundred mV, with a maximum of approximately 2 V. Therefore, an input signal of 500 mV RMS corresponds to just under 3 Vpp, which is the maximum input level for the AIC when the ± 1.5-V full-scale input gain is used in a single-ended confiquration.

The AIC's output is designed to drive a minimum load impedance of 300 ohms. This allows it to drive telecommunications circuitry, which commonly exhibits impedances of 600 ohms. In contrast, most speakers present impedances between four and sixteen ohms.

Accordingly, an LM386 audio power amplifier (UF6) is used to drive speaker impedances and provide power gain. Because the LM386 has an inherent gain of 20, a voltage divider is used at the AIC's output to avoid overdriving the input to this amplifier.

With a 2-to-10 voltage divider, the gain of the output amplifier circuit is four. An AIC output voltage of 1.5 Vpp corresponds to an amplifier output of approximately 6 Vpp (unclipped) when driving an 8-ohm load impedance. Refer to the LM386 data sheet for detailed information on the output characteristics.

Table 2–14 shows TMS320C30 EVM analog input/output characteristics.

Table 2–14. TMS320C30 EVM Analog I/O Characteristics

Parameter	Typ	Max	Unit
Input voltage		1.45	V
Output clipping voltage			
4-ohm load	3.5		V
8-ohm load	6.5		V
16-ohm load	10.0		V

TMS320C30 EVM analog capabilities are suited to many applications, including audio data processing. You can directly connect most speakers and microphones to the TMS320C30 EVM analog input (J4) and output(J3) jacks. An inexpensive microphone, amplifier, and a speaker are sufficent. However, higher quality audio is achieved when you route the microphone through a good quality stereo amplifier that has a volume control and good quality speakers.

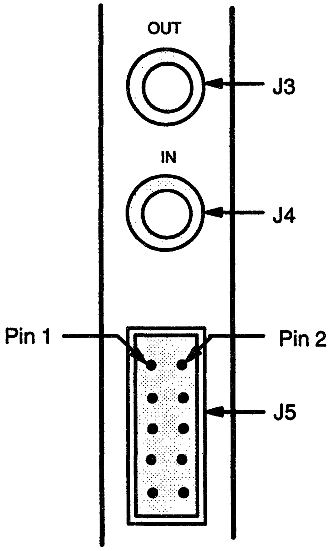
CAUTION

Consult your equipment data sheets for compatibilty before connecting analog devices. Failure to do so could damage the TMS320C30 EVM and/or your analog input/output equipment.

2.6.3 External Serial Port Interface

The TMS320C30 EVM provides one external serial port interface by means of a 10-pin header (J5). Table 2–15 shows the pinouts for the unbuffered interface.

Table 2–15. J5 Pinouts



Pin Number	Signal	Pin Number	Signal
1	GND	2	FSR1
3	CLKX1	4	DR1
5	DX1	6	TCLK1
7	FSX1	8	XF1
9	CLKR1	10	GND

2.6.4 AIC Initialization

In the following procedure, serial port 0 is set up by doing Step 1–Step 6; the AIC is then initialized in Step 7.

The numbers contained in the **Step** column of Table 2–16 correspond to each step outlined after the table.

Table 2–16. AIC Setup Values

Step	Address/Register	Write Value	'C30 Registers
1	0x808028	0x00000001	Timer period
	0x808020	0x000002C1	Timer global control
2	IOF	0x00000002	IOF
3	0x808042	0x00000111	Transmit port control
	0x808043	0x00000111	Receive port control
4	0x808040	0x0E970300	Port global control
5	0x808048	0x00000000	Data transmit
6	IOF	0x00000006	IOF
7	0x808048	0x00000003	Data transmit
	0x808048	0x00001a34	Data transmit
	0x808048	0x00000003	Data transmit
	0x808048	0x000002a7	Data transmit

To initialize the AIC, use these values and do the following:

Step 1: Set timer 0 to operate in the pulse mode with one H1 pulse. This provides a 7.5-MHz AIC master clock.

1) Set the timer period register (0x808028) to 1.
2) Set the timer control register (0x808020).
   a) Set TCLK as a timer pin.
   b) Reset/start the timer.
   c) Disable timer hold.
   d) Operate in the pulse mode.
   e) Select an internal clock source.

Step 2: Reset the AIC by writing a logic 0 to the IOF register and setting XF0 to output.

Step 3: Initialize the serial port transmit and receive control registers (0x808042 and 0x808043, respectively).

1) Set CLKX/R to a serial port pin.
2) Set DX/R to a serial port pin.
3) Set FSX/R to a serial port pin.

Step 4: Initialize the serial port global control register (0x808040).

1) Select FSX as an input.
2) Select an external transmit-and-receive clock source.

3) Set transmit/receive to the variable mode.
4) Select the standard mode for frame sync with FSX/FSR set to active low.
5) Set for 16-bit word transfers.
6) Enable the transmit/receive interrupts.
7) Reset both the transmit/receive sides of the serial port.

Step 5: Clear the transmit register (0x808048).

Step 6: Take the AIC out of reset by writing a logic 1 to the IOF, which sets XF0 output high.

Step 7: The TMS320C30 device initializes the AIC with two secondary communications sequences. The initialization parameters follow:

1) Setup secondary communications.
2) Select 8 kHz for the sampling rate.
3) Setup secondary communications.
4) Select the controls as follows:
   a) Enable A/D high pass.
   b) Disable loopback.
   c) Select the primary analog input.
   d) Select the synchronous mode for transmitting and receiving.
   e) Select the ± 1.5-V input.
   f) Insert a (sin *x*)/*x* D/A correction filter.
   g) Enable high pass.

Chapter 3

TMS320C30/Host Communications Protocol

The TMS320C30 and the host communicate by means of the TMS320C30 EVM 16-bit communications register. For reliable communications, a protocol must be established that satisfies both the host and the TMS320C30. This chapter discusses the communications protocol used by the host and the TMS320C30 EVM.

These topics are discussed:

Section **Page**

3.1 Protocol Overview 3-2
3.2 System Initialization 3-4
3.3 The Command Preprocessor 3-5
3.4 Command Execution 3-7

3.1 Protocol Overview

The TMS320C30 and the host have a register-based communications structure. While the communications interface is general purpose, its architecture is well suited for a protocol where the host is the master and the TMS320C30 is the slave. This is because the TMS320C30 side of the communications is interrupt-driven, while the host side is polled.

Note:

TMS320C30 interrupts indicate whether the host does a command (INT0) access or a data (INT1–INT2) access.

The host cannot distinguish between command and data accesses, because of the limited number of event pins on the TBC. Consequently, a more sophisticated communications protocol is required where the TMS320C30 is the master and the host PC is the slave.

Communications source code is provided as follows:

- In Appendix A, *Host Communications Software.*
- In Appendix B, *TMS320C30 Communications Software.*
- On disk as part of the *TMS320C30 C Source Debugger.*
  - *evm30.exe* – TMS320C30 EVM C source debugger.
- On disk as part of the TMS320C30 EVM Applications Software Kit.
  - *evmload.exe* – Standalone loader. Loads a TMS320C30 *.out* file and begins execution.
  - *evminit.asm* – TMS320C30 assembly file. Contains initialization and communications code executed by the TMS320C30.
  - *evmhost.c* – Microsoft™ C source file. Contains initialization and communications code executed by the host PC.

The communications protocol presented in this manual capitalizes on the TMS320C30 EVM's best features. You may wish to modify the provided code or develop your own protocol to meet your specific needs.

Note:

The communications protocol provided in this manual assumes that the host is the master and that the TMS320C30 is the slave.

The basic elements of the protocol are as follows:

- System initialization
  - Host
  - TMS320C30

- Command preprocessor
  - Command transfer
  - Command formatting
- Command execution
  - 16-bit transfers
  - 32-bit transfers
  - 16-bit DMA transfers

The examples given in this manual support data transfers, but they can be easily extended to support any type of command.

3.2 System Initialization

The host initializes the TMS320C30 EVM to support communications by doing the following:

- It resets the TMS320C30 EVM, then holds the TMS320C30 in a reset condition.
- It Initializes the TBC for default operation.
- It downloads the TMS320C30 monitor/communications code into the EVM by means of the TBC.
- It removes the TMS320C30 reset condition.

The TMS320C30 debugger (EVM30) or the EVM standalone loader (EVMLOAD) is used to accomplish these operations. The software kits that are shipped with the TMS320C30 EVM contain more information about using the EVM30 and EVMLOAD capabilities.

Once host initialization is completed, the TMS320C30 initializes the EVM. The TMS320C30 is initialized by the source code implemented in the *evminit.asm* routine (see Appendix B). This code does the following:

- It clears and disables interrupts.
- It initializes the data page pointer and the stack pointer.
- It enables cache memory.
- It initializes the memory interface.
- It resets and initializes the AIC.
- It clears out the communications command structure.
- It enables TMS320C30 interrupt 0 and the global interrupt bit.
- It goes to the idle state and awaits command interrupts.

3.3 The Command Preprocessor

The **command preprocessor** consists of two separate operations:

1) *Command transfers* transfer commands to the EVM command structure, and

2) *Command formatting* formats command parameters and starts command execution.

3.3.1 Host/TMS320C30 Command Transfer and Formatting

The following explains host/TMS320C30 command transfer and formatting sequences:

Step 1: The host initiates a command transfer by writing a command send request to the EVM. This generates an INT0 to the TMS320C30, which prepares the TMS320C30 to accept command parameters.

Step 2: The host sends command parameters to the EVM as standard data transfers. Each data transfer generates an INT1 to the TMS320C30. The TMS320C30 stores the command parameters in the command structure.

Step 3: The host terminates the command parameter transfer by sending a command end to the EVM, which generates an INT0 to the TMS320C30.

Step 4: The TMS320C30 formats the command parameters. Command parameter formatting is necessary because the host sends the parameters as 16-bit values; the TMS320C30 requires 32-bit values.

Step 5: Once the command parameters are formatted, the TMS320C30 looks at the command code in the parameter structure and prepares for command execution.

Step 6: When the command is set up and ready for execution, the TMS320C30 sends a command ready acknowledge back to the host. The host begins data transfer.

3.3.2 Host Writes

The host writes to the communications register. This generates an INT1 to the TMS320C30. The TMS320C30 responds by reading the data from the communications register, which generates an EVT2 to the TBC.

3.3.3 Host Reads

The host reads data from the communications register and generates an INT2 to the TMS320C30. The TMS320C30 responds by writing the next word of data to the communications register, which generates an EVT1 to the TBC. The host must perform one dummy read to the communications register, at the beginning of each block transfer, to initialize the register.

3.3.4 TMS320C30 EVM Command Structure

The TMS320C30 EVM command structure is defined as follows:

```
com_stat    .word 00000000h ;command status
com_cmd     .word 00000000h ;command code
com_countl .word 00000000h ;transfer count, low
com_counth .word 00000000h ;transfer count, high
com_saddrl .word 00000000h ;source address, low
com_saddrl .word 00000000h ;source address, high
com_daddrl .word 00000000h ;destination address, low
com_daddrh .word 00000000h ;destination address, high
```

3.3.5 Transferring and Formatting Count Parameters

When count parameters are transferred, the low 16 bits are stored in the `com_countl` word; the high 16 bits are stored in the `com_counth` word.

When the count is formatted, `com_counth` is shifted left 16 bits, added to `com_countl`, and stored in `com_countl`.

This procedure applies to source and destination addresses as well. Formatting the parameters as a secondary operation eliminates the need for the transfer function to recognize the type of command being transferred. This generalizes the transfer operation.

All data transfer commands are interrupt driven. The *hwrite16*, *hread16*, *hwrite32*, and *hread32* commands require an interrupt service routine (ISR) in order to transfer data; DMA commands do not.

To transfer data, a non-DMA command requires that the entry point of the ISR reside at the interrupt vector location for the interrupt source. For example, to execute an *hwrite16* command, its entry point is placed at the interrupt vector location for INT1, address 0x000001.

The correct interrupt entry is stored in the vector location after the host terminates the command parameter transfer. These steps are carried out by the *hcontrol* (INT0) routine in *evminit.asm.*

3.4 Command Execution

While this manual concentrates on the handling of data transfers, virtually any command is supported by the communications protocol. Command extension requires adding a new command code, a command formatter, and the command execution code itself.

Adding a new command code and a command formatter is quite simple. However, writing the execution code itself is as simple or as difficult as the application requires.

3.4.1 Types of Data Transfers

Three types of data transfers are supported by the communications protocol: 16-bit, 32-bit, and 16-bit DMA. In each case, data is transferred to and from sequential locations in TMS320C30 memory.

3.4.1.1 Non-DMA Commands

Non-DMA commands do not use the count value. Once a non-DMA command has been started, the host continues to transfer data until the next command request is generated. Consequently, non-DMA commands do not terminate on their own. Rather, the host is expected to keep track of the number of words transferred.

For 16-bit commands, the upper 16 bits of the data word are set to 0 before being stored in memory.

3.4.1.2 DMA Commands

DMA transfers are well suited for EVM applications because audio data is generally 16 bits or less. Also, DMA cycles do not consume valuable CPU execution time to transfer data.

DMA commands do use the count value to generate a CPU interrupt when the count is zero — thus, disabling the DMA. For example, the DMA is disabled by writing a 0 to the DMA global control register (0x808000).

DMA cycles do not cause code execution; the upper 16 bits of a DMA data word are garbage. DMA data may require special processing before a mathematical computation is performed.

3.4.2 Command Execution Examples

Example 3–1 through Example 3–6 show the command execution sequence for each of the commands on the TMS320C30 side of the communications interface. Appendix A, *Host Communications Software*, and Appendix B, *TMS320C30 Communications Software*, contain the source codes for these commands.

Example 3–1. 16-Bit Writes — hwrite16

This example shows the sequence for transferring data from the communications register to the TMS320C30 memory. The INT1 register drives the *hwrite16* interrupt routine, which executes the following sequence:

Step 1: Load the communications register address `@hostport`.

Step 2: Fetch the data word from the communications register.

Step 3: Zero the upper half of the data value.

Step 4: Load the destination address from the command structure `@com_daddrl`.

Step 5: Store the data word at the location pointed to by the destination address; increment the destination address.

Step 6: Store the destination address `@com_daddrl`.

Example 3–2. 16-Bit Reads — hread16

This example shows the sequence for transferring data from TMS320C30 memory and writing it to the communications register. The INT2 register drives the *hread16* interrupt routine, which executes the following sequence:

Step 1: Load the source address from the command structure `@com_saddrl`.

Step 2: Fetch the data word from the location pointed to by the source address; increment the source address.

Step 3: Store the source address `@com_saddrl`.

Step 4: Load the communications register address `@hostport`.

Step 5: Store data to the communications register.

Example 3–3. 32-Bit Writes — hwrite32

This example shows the sequence for transferring data from the communications register to the TMS320C30 memory. The INT1 register drives the *hwrite32* interrupt routine, which executes the following sequence:

Step 1: Load the address of the communications register `@hostport`.

Step 2: Fetch the data word from the communications register.

Step 3: Load the destination address from the command structure `@com_daddrl`.

Step 4: Load the value `@wordflag`.

Step 5: Test the LSB of the wordflag, add 1 to the wordflag value, and store the value `@wordflag`.

- ❑ If the LSB was 0, the low word is loaded; zero out the upper half of the word.
- ❑ If the LSB was 1, the high word is loaded; shift the data left 16 bits and add it to the previous low word.

Step 6: Store the data word at the location pointed to by the destination address; increment the destination address.

Step 7: Store the destination address `@com_daddrl`.

Example 3–4. 32-Bit Reads — hread32

This example shows the sequence for transferring data from TMS320C30 memory to the communications register. The INT2 register drives the *hread32* interrupt routine, which executes the following sequence:

Step 1: Load the source address from the command structure `@com_saddrl`.

Step 2: Load the value `@wordflag`.

Step 3: Test the LSB of the wordflag, add 1 to the wordflag value, and store the value `@wordflag`.

Step 4: Fetch the data word from the location pointed to by the source address.

Step 5: If the LSB was 1, the high word is loaded; shift the data right 16 bits and add 1 to the source address.

Step 6: Store the source address `@com_saddrl`.

Step 7: Load the address of the communications register `@hostport`.

Step 8: Store data value to the communications register.

Example 3–5. 16-Bit DMA Writes — dmawrite

This examples shows the sequence for transferring data from the communications register to the TMS320C30 memory. The *dmawrite* routine sets up DMA writes, which are driven by the INT1 register. The sequence follows:

Step 1: Set the DMA source address register to `@hostport`.

Step 2: Set the DMA destination register address to `@com_daddrl`.

Step 3: Set the DMA count to `@com_countl`.

Step 4: Set the DMA control register to increment the destination address, sync off of source interrupts, and generate a CPU interrupt on counter equal to zero.

Step 5: Enable INT1 as the DMA interrupt source.

Example 3–6. 16-Bit DMA Reads — dmaread

This example shows the sequence for transferring data from TMS320C30 memory to the communications register. The *dmaread* routine sets up DMA reads, which are driven by the INT2 register. The sequence follows:

Step 1: Set the DMA source address register to `@com_daddr1`.

Step 2: Set the DMA destination address register to `@hostport`.

Step 3: Set the DMA count to `@com_count1`.

Step 4: Set the DMA control register to increment the source address, sync off of destination interrupts, and generate a CPU interrupt on counter equal to zero.

Step 5: Enable INT2 as the DMA interrupt source.

Appendix A

Host Communications Software

This appendix contains the PC/AT C source code programs that are used for EVM communications protocol. They can also be found on the *TMS320C30 EVM Applications Software* diskette, part number 2547304-1601. Specifically, the beginning of Section A.2, *TMS320 EVM Host Routines*, details how executable source code is built.

Section **Page**

A.1 TMS320C30 and TBC Parameters A-2
A.2 TMS320C30 EVM Host Routines A-4
A.3 Simple EVM Test Program A-6
A.4 EVM_reset A-7
A.5 EVM_memwrite A-8
A.6 EVM_memread A-9
A.7 EVM_cmdinit A-10
A.8 EVM_status A-11

A.1 TMS320C30 and TBC Parameters

```
/* TMS320C30 and Test Bus Controller Parameters */

typedef           int      INT;
typedef           char     INT8;
typedef           short    INT16;
typedef           long     INT32;
typedef unsigned  char     UINT8;
typedef unsigned  short    UINT16;
typedef unsigned  long     UINT32;
typedef unsigned  long     TRG_ADDR;
typedef unsigned  long     TRG_MBLOCK;

#define SC_REG_CNTL0                0x0000
#define SC_REG_CNTL1                0x0002
#define SC_REG_CNTL2                0x0004
#define SC_REG_CNTL3                0x0006
#define SC_REG_CNTL4                0x0008
#define SC_REG_CNTL5                0x000A
#define SC_REG_CNTL6                0x000C
#define SC_REG_CNTL7                0x000E

#define SC_REG_CNTL8                0x0010
#define SC_REG_CNTL9                0x0012
#define SC_REG_MINOR_CMD            0x0014
#define SC_REG_MAJOR_CMD            0x0016
#define SC_REG_CNT1_LOW             0x0018
#define SC_REG_CNT1_HIGH            0x001A
#define SC_REG_CNT2_LOW             0x001C
#define SC_REG_CNT2_HIGH            0x001E

#define SC_REG_STATUS0              0x0400
#define SC_REG_STATUS1              0x0402
#define SC_REG_STATUS2              0x0404
#define SC_REG_STATUS3              0x0406
#define SC_REG_CAPT_LOW             0x0408
#define SC_REG_CAPT_HIGH            0x040A
#define SC_REG_SERIAL_RD            0x040C
#define SC_REG_SERIAL_WR            0x040E

#define SC_REG_XL0                  0x0410
#define SC_REG_XL1                  0x0412
#define SC_REG_XL2                  0x0414
#define SC_REG_XL3                  0x0416
#define SC_REG_XL4                  0x0418
#define SC_REG_XL5                  0x041A
#define SC_REG_XL6                  0x041C
#define SC_REG_XL7                  0x041E

#define MCNTL                       0x0800
#define MPORT                       0x0808
#define SC_C30EVM_RESET             0x0818

#define SC_CTL2_MODE0               0x0001
#define SC_CTL2_MODE1               0x0002
#define SC_CTL3_CLKOUT_OFF          0x0010
#define SC_CTL3_ECD                 0x0300
#define SC_CTL3_FORMAT_MPSD         0x0400

#define SC_CTL4_DSOURCE_TDI0         0x0080
#define SC_CTL4_TDI_FALL_EDGE        0x1000
#define SC_CTL4_EVENT_FALL_EDGE      0x2000
```

```
#define SC_CTL5_SET_EVT3         0x0008
#define SC_CTL5_EVT3_OUTPUT      0x0800

#define SC_MINOR_CMD_CLEAR0      0x0000
#define SC_MINOR_CMD_OPERATE2    0x6000

#define SC_OPERATE2_STATUS       0x0040
#define SC_OPERATE2_CAPT_CNT1    0x0004

#define SC_DEFAULT_CABLE_DELAY   0x0002

#define    UPDATE_STATUS  outport(ctladdr+SC_REG_MINOR_CMD,SC_MINOR_CMD_OPERATE2| \
                                                           SC_OPERATE2_STATUS   |  \
                                                           SC_OPERATE2_CAPT_CNT1)

#define clear_int(x)   outport(ctladdr+SC_REG_MINOR_CMD,    \
                                       SC_MINOR_CMD_CLEAR0 | x)
```

A.2 TMS320C30 EVM Host Routines

```
/****************************************************************/
/*              TMS320C30 EVM HOST ROUTINES                     */
/*                                                              */
/*     These routines were compiled and linked with the         */
/*     Microsoft 5.1 compiler package using the large model.    */
/*     Command used:  cl /AL evmhost.c [return]                 */
/*                                                              */
/*     The TMS320C30 monitor/communications code was loaded     */
/*     and executed via the EVM loader.                         */
/*     Command used: evmload evminit.out [return]               */
/*                                                              */
/*     After loading the monitor you can execute the test code. */
/*     Command used: evmhost [return]                           */
/*                                                              */
/*     The result of the test should be as follows:             */
/*                                                              */
/*         Read data 0001 000b 0015                             */
/*         Read data 0002 000c 0016                             */
/*         Read data 0003 000d 0017                             */
/*         Read data 0004 000e 0018                             */
/*         Read data 0005 000f 0019                             */
/*         Read data 0006 0010 001a                             */
/*         Read data 0007 0011 001b                             */
/*         Read data 0008 0012 001c                             */
/*         Read data 0009 0013 001d                             */
/*         Read data 000a 0014 001e                             */
/****************************************************************/

/****************************************************************/
/*     Functions                                                */
/*                                                              */
/*     EVM_reset()      Reset the TMS320C30 EVM                 */
/*     EVM_memwrite()   Write block of memory to C30            */
/*     EVM_memread()    Read block of memory from C30           */
/*     EVM_cmdinit()    Send command to EVM                     */
/*     EVM_status()     Check TBC status                        */
/****************************************************************/

#include  "tbc.h"

#define    BEGIN_CMD_SEND      1    /*Begin sending cmd parameters   */
#define    END_CMD_SEND        2    /*End sending cmd parameters */

#define    CMD_OK              0    /*Received cmd parameters ok     */
#define    CMD_ERROR          -1    /*Error on receiving command parameters */

#define    CMD_FINISH          0    /*Status, command is finished   */
#define    CMD_LOAD            1    /*Status, command is being loaded   */
#define    CMD_ACTIVE          2    /*Status, command is currently active   */

#define    CMD_NOP             0    /*Nop command                 */
#define    CMD_HOST_MR16      11    /*C30 memory read, 16 bit mode   */
#define    CMD_HOST_MW16      12    /*C30 memory write, 16 bit mode   */
#define    CMD_HOST_MR32      13    /*C30 memory read, 32 bit mode   */
#define    CMD_HOST_MW32      14    /*C30 memory write, 32 bit mode */
#define    CMD_HOST_DMAR      15    /*C30 memory read via dma, 16 bit mode   */
#define    CMD_HOST_DMAW      16    /*C30 memory write via dma,16 bit mode   */

#define    outport         outpw    /*output a word to I/O port  */
#define    inport          inpw     /*input a word from I/O port  */
#define    outcntl         outp     /*output control byte  */
#define    incntl          inp      /*input a control byte from data reg   */
```

```
#define    MWRITE_ACK    0x0004    /*TBC interrupt 2, c30 read   */
#define    MREAD_ACK     0x0002    /*TBC interrupt 1, c30 write   */

#define    MAX_TIMEOUT     10
#define    TRUE             1
#define    FALSE            0

unsigned short ctladdr = 0x0240;
```

A.3 Simple EVM Test Program

```
/***********************************************************************/
/*  Simple test program that passes data to the EVM and reads it back. */
/*                                                                     */
/*  It assumes that the TMS320C30 monitor has been loaded and is       */
/*  executing.  You can load and execute the monitor program via the   */
/*  debugger (EVM30) or with the loader (EVMLOAD).                     */
/*                                                                     */
/*  Operations: - Read in EVM I/O base address if provided             */
/*              - Reset the EVM                                        */
/*              - Write data to the EVM with 16-bit CPU command        */
/*              - Write data to the EVM with 16-bit DMA command        */
/*              - Write data to the EVM with 32-bit CPU command        */
/*              - Read data from the EVM with 16-bit CPU command       */
/*              - Read data from the EVM with 16-bit DMA command       */
/*              - Read data from the EVM with 32-bit CPU command       */
/*              - Print out the data read back from the EVM            */
/***********************************************************************/
main(argc, argv)
       int argc;
       char *argv[];
{
       INT      type = CMD_HOST_MW16;
       UINT32   count = 10;
       UINT16   wdata[30],rdata[30];
       UINT32   c30_addr = 0x00001000;
       int  i;
       char * argp;
       for (i = 1; i < argc; ++i)
       {
          char *argp = argv[i];
          if (*argp == ’-’)
          {
             while (*++argp) switch(*argp)
             {
               case ’P’: case ’p’:  sscanf(argv[++i], ”%x”,&ctladdr);  break;
               default :           continue;
             }
          }
       }
       if(EVM_reset())
           { printf(”EVM reset failed\n”); exit(-1); }
       for(i=0;i<30;i++) {wdata[i] = i+1; rdata[i] = 0;}
       if(EVM_memwrite(CMD_HOST_MW16,count,&wdata[0],c30_addr))
           { printf(”16 bit write command failed\n”); exit(-1); }
       if(EVM_memwrite(CMD_HOST_DMAW,count,&wdata[10],c30_addr+10))
           { printf(”dma write command failed\n”); exit(-1); }
       if(EVM_memwrite(CMD_HOST_MW32,count/2,&wdata[20],c30_addr+20))
           { printf(”32 bit write command failed\n”); exit(-1); }
       if(EVM_memread(CMD_HOST_MR16,count,c30_addr,&rdata[0]))
           { printf(”16 bit read command failed\n”); exit(-1); }
       if(EVM_memread(CMD_HOST_DMAR,count,c30_addr+10,&rdata[10]))
           { printf(”dma read command failed\n”); exit(-1); }
       if(EVM_memread(CMD_HOST_MR32,count/2,c30_addr+20,&rdata[20]))
           { printf(”32 bit read command failed\n”); exit(-1); }
       for(i=0;i<10;i++)
         printf(”Read data %04x %04x %04x\n”,rdata[i],rdata[i+10],rdata[i+20]);
}
```

A.4 EVM Reset

```
/************************************************************************/
/*  EVM_reset()                                                         */
/*                                                                      */
/*  Reset the TMS320C30 EVM and intialize the TBC for default operation.*/
/*                                                                      */
/*  Operations: - Reset the EVM                                         */
/*              - Intialize the TBC                                     */
/*                - Set TBC mode                                        */
/*                - Set TBC event handling                              */
/*                - Set TBC to use MPSD format                          */
/*              - Set EVT3 to output, toggle the TMS320C30 reset pin   */
/*              - Check to see if the EVM is reset                      */
/************************************************************************/

int    EVM_reset()
  {
       int i;
       outport(ctladdr+SC_C30EVM_RESET,0);

       outport(ctladdr+SC_REG_CNTL2,SC_CTL2_MODE0 | SC_CTL2_MODE1);

       outport(ctladdr+SC_REG_CNTL4,SC_DEFAULT_CABLE_DELAY |
                                    SC_CTL4_DSOURCE_TDIO |
                                    SC_CTL4_TDI_FALL_EDGE |
                                    SC_CTL4_EVENT_FALL_EDGE);

       outport(ctladdr+SC_REG_CNTL3,SC_CTL3_CLKOUT_OFF |
                                    SC_CTL3_ECD ¦
                                    SC_CTL3_FORMAT_MPSD);

       outport(ctladdr+SC_REG_CNTL5,SC_CTL5_EVT3_OUTPUT |  SC_CTL5_SET_EVT3);
       outport(ctladdr+SC_REG_CNTL5,SC_CTL5_EVT3_OUTPUT & ~SC_CTL5_SET_EVT3);

       outport(ctladdr+SC_REG_CNTL1, 5);
       if(inport(ctladdr+SC_REG_CNTL1) != 5)
           return(-1);
       else
           outport(ctladdr+SC_REG_CNTL1,0);

       for(i=0;i<100;i++);

       return(0);
  }
```

A.5 EVM_memwrite

```
/**************************************************************************/
/*  EVM_memwrite()                                                        */
/*                                                                        */
/*  Transfers N words of data from the host memory to TMS320C30 EVM       */
/*  memory.  Transfers can be 16/32-bit or 16-bit DMA.                    */
/*                                                                        */
/*  Operations: - Send command initialization                             */
/*              - Clear out any outstanding write events in the TBC       */
/*              - If 32 bit transfer, adjust the transfer count           */
/*              - Transfer data until count is zero or error occurs       */
/**************************************************************************/

int    EVM_memwrite(type,count,src_addr,dst_addr)
       INT        type;
       UINT32     count;
       void      *src_addr;
       TRG_ADDR   dst_addr;
   {
       UINT16  mport    = ctladdr+MPORT;
       UINT16  *memaddr = (UINT16*)src_addr;

       if(EVM_cmdinit(type,count,(UINT32)src_addr,(UINT32)dst_addr))
           return(-1);

       clear_int(MWRITE_ACK);

       if (type == CMD_HOST_MW32)
           count = count * 2;

       while(count --)
       {
           outport(mport,*memaddr++);
           if(!EVM_status(MWRITE_ACK)) return(-1);
       }
       return(0);
   }
```

A.6 EVM_memread

```
/************************************************************************/
/*. EVM_memread()                                                       */
/*                                                                      */
/*  Transfers N words of data from the TMS320C30 EVM memory to host     */
/*  memory.  Transfers can be 16/32-bit or 16-bit DMA.                  */
/*                                                                      */
/*  Operations: - Send command initialization                           */
/*              - Clear out any outstanding read  events in the TBC     */
/*              - Perform dummy read to fill the comm. buffer           */
/*              - If 32-bit transfer adjust the transfer count          */
/*              - Transfer data until count is zero or error occurs     */
/************************************************************************/
int    EVM_memread(type,count,src_addr,dst_addr)
       INT        type;
       UINT32     count;
       TRG_ADDR   src_addr;
       void      *dst_addr;
   {
       UINT16  mport    = ctladdr+MPORT;
       UINT16  *memaddr = (UINT16*)dst_addr;

       if(EVM_cmdinit(type,count,(UINT32)src_addr,(UINT32)dst_addr))
           return(-1);

       clear_int(MREAD_ACK);
       inport(mport);

       if (type == CMD_HOST_MR32)
           count = count * 2;

       while(count --)
       {
           if(!EVM_status(MREAD_ACK)) return(-1);
           *memaddr++ = inport(mport);
       }
       return(0);
   }
```

A.7 EVM_cmdinit

```
/*******************************************************************/
/*  EVM_cmdinit()                                                  */
/*                                                                 */
/*  Send command to the TMS320C30 EVM                              */
/*                                                                 */
/*  Operations: - Send command begin request                       */
/*              - Wait for write acknowledge (i.e. C30 read buffer)*/
/*              - Send count, source address and dest. address     */
/*              - Clear read acknowledge (i.e. C30 write buffer)   */
/*              - Send command end request                         */
/*              - Wait for C30 to send command acknowledge         */
/*******************************************************************/

int    EVM_cmdinit(type,count,src_addr,dst_addr)
       INT      type;
       UINT32   count;
       UINT32   src_addr;
       UINT32   dst_addr;
   {

       UINT16 mport = ctladdr+MPORT;
       char   temp;

       clear_int(MWRITE_ACK);
       outcntl(ctladdr+MCNTL,BEGIN_CMD_SEND);

       if(EVM_status(MWRITE_ACK)) outport(mport,type);
       else return(-1);

       if(EVM_status(MWRITE_ACK)) outport(mport,(UINT16)count);
       else return(-1);

       if(EVM_status(MWRITE_ACK)) outport(mport,(UINT16)(count>>16));
       else return(-1);

       if(EVM_status(MWRITE_ACK)) outport(mport,(UINT16)src_addr);
       else return(-1);

       if(EVM_status(MWRITE_ACK)) outport(mport,(UINT16)(src_addr>>16));
       else return(-1);

       if(EVM_status(MWRITE_ACK)) outport(mport,(UINT16)dst_addr);
       else return(-1);

       if(EVM_status(MWRITE_ACK)) outport(mport,(UINT16)(dst_addr>>16));
       else return(-1);

       clear_int(MREAD_ACK);

       if(EVM_status(MWRITE_ACK)) outcntl(ctladdr+MCNTL,END_CMD_SEND);
       else return(-1);

       if( EVM_status(MWRITE_ACK) && EVM_status(MREAD_ACK) &&
           ((temp = incntl(mport)) == CMD_OK))       return(0);
       else return(-1);
   }
```

A.8 EVM_status

```
/*************************************************************************/
/*  EVM_status()                                                         */
/*                                                                       */
/*  Poll the TBC looking for an event                                    */
/*                                                                       */
/*  Operations: - Update the TBC status register                         */
/*              - Read/mask status                                       */
/*              - Return if status true                                  */
/*              - Else continue until status becomes true or timeout     */
/*                                                                       */
/*                Note: The software timeout is simple, however it's     */
/*                      operation is unpredictable depending on host     */
/*                      CPU speed.  You may wish to modify to meet       */
/*                      your own needs.                                  */
/*************************************************************************/

int     EVM_status(mask)
        UINT16  mask;
   {
        int timeout = MAX_TIMEOUT;

        while(timeout--)
        {
            UPDATE_STATUS;
            if(inport(ctladdr+SC_REG_STATUS0) & mask)
            {   clear_int(mask); return(TRUE); }
        }
        return(FALSE);

   }
```

Appendix B

TMS320C30 Communications Software

This appendix contains the TMS320C30 assembly source code for EVM communications protocol. These programs can be found on the *TMS320C30 EVM Applications Software* diskette, part number 2547304-1601. Specifically, the opening comments at the beginning of Section B.1, *TMS320C30 Assembly Source Code*, details how executable source code is built.

B.1 TMS320C30 Assembly Source Code

```
           .length        60
           .width        132

           .global sysinit,aicreset,com_parm,hcontrol
           .global hread16,hwrite16,hread32,hwrite32
           .global wait_transmit_0, dmadone, wordflag

*************************************************************************
*      TMS320C30 EVM Monitor/Communications Code                        *
*                                                                       *
*      Texas Instruments Incorporated                                   *
*                                                                       *
*      Tools Used:        TMS320C330 Assy/Lnkr Version 2.10             *
*      Assembly File:     evminit.asm                                   *
*      Link Command File: evminit.cmd                                   *
*      Commands Used:     asm30 -s evminit.asm                          *
*                         lnk30 evminit evminit.cmd                     *
*                                                                       *
*************************************************************************

STACK_SIZE .set  400h          ;size of system stack

stack      .usect  ".stack",STACK_SIZE

           .sect   "vectors"
PARMS:
reset      .word   sysinit
int0       .word   hcontrol
int1       .word   hwrite16
int2       .word   hread16
int3       .word   null_int
xint0      .word   transmit0
rint0      .word   receive0
xint1      .word   transmit1
rint1      .word   recieve1
tint0      .word   null_int
tint1      .word   timer1
dint0      .word   dmadone
*****************************************************************
*hosport is placed in the vectors section so that the loader    *
*can modify this value at load time or the user can modify it   *
*at run time.  The value is used to set the address of the host *
*communications port.                                           *
*****************************************************************

hostport   .word   000804000h
reserved   .space  033h

           .sect   "comdata"

*****************************************************************
* Variables used to hold addresses of stack, interrupt routines  *
* that are mapped in/out, and parameters                        *
*****************************************************************

stack_addr .word   stack              ;address of stack
int1_hwr16 .word   hwrite16           ;Address of 16 bit host write function
int1_hwr32 .word   hwrite32           ;Address of 32 bit host write function
int1_cwr   .word   cmd_write          ;Address of command write function
int2_hrd16 .word   hread16            ;Address of 16 bit host read function
int2_hrd32 .word   hread32            ;Address of 32 bit host read function
cmd_temp   .word   com_cmd            ;Temporary address of command parameters
com_parm   .word   com_stat           ;Address of command parameters
wordflag   .word   0
```

```
*****************************************************************
* Addresses of various peripherals and memory control registers *
*****************************************************************
dma_ctl    .word   000808000h       ;dma global control register
mcntlr0    .word   000808064h       ;i/o interface control reg. addr.
mcntlr1    .word   000808060h       ;parallel interf. cntl. reg. addr.
t0_ctladdr .word   000808020h       ;Timer 0
t1_ctladdr .word   000808030h       ;Timer 1
p0_addr    .word   000808040h       ;Serial port 0

*****************************************************************
* Control parameters to large to fit in immediate value         *
*****************************************************************

enbl_eint1 .word   000020400h       ;int1 interrupt dma (host writes)
enbl_eint2 .word   000040400h       ;int2 interrupt dma (host writes)
enbl_sp0_r .word   000000020h       ;serial port 0 receive interrupt
intoff     .word   0fff0fbf1h       ;turn off int1,int2,int3,eint0,eint1
                                    ;eint2, eint3, dma
intclr     .word   0fffffff9h       ;clr out int0-2
t0_ctlinit .word   0C00002C1h       ;set timer as clk out, H1/2 period
                                    ;timer will run when cpu stops in
                                    ;emulation mode

p0_global  .word   00e970300h       ;serial port 0 global control register
dma_wctl   .word   0C0000943h       ;dma write control
                                    ;com. reg. -> C30 mem.
                                    ;interrupt driven from host writes
dma_rctl   .word   0C0000A13h       ;dma read control
                                    ;c30 mem -> com reg
                                    ;interrupt driven from host reads

*****************************************************************
* Host communications command structure                         *
*****************************************************************

com_stat   .word   0000000000h        ;command status
com_cmd    .word   0000000000h        ;command
com_countl .word   0000000000h        ;transfer count low
com_counth .word   0000000000h        ;transfer count high
com_saddrl .word   0000000000h        ;source addr low
com_saddrh .word   0000000000h        ;source addr high
com_daddrl .word   0000000000h        ;destination address low
com_daddrh .word   0000000000h        ;destination address high
```

```
*****************************************************************
* Various constants                                             *
*****************************************************************
WAIT0             .set    0000h       ;memory control reg val, parallel bus
WAIT1             .set    0000h       ;memory control reg val, i/o bus
CACHE             .set    1800h       ;clear and enable cache
ENBL_GIE          .set    2000h       ;global interrupt enable
ENBL_INT0         .set    0001h       ;interrupt 0 enable
ENBL_INT1         .set    0002h       ;interrupt 1 enable
ENBL_INT2         .set    0004h       ;interrupt 2 enable
ENBL_INT3         .set    0008h       ;interrupt 3 enable
ENBL_XINT0        .set    0010h       ;serial port 0 transmit int. enable
ENBL_RINT0        .set    0020h       ;serial port 0 receive  int. enable
ENBL_XINT1        .set    0040h       ;serial port 1 transmit int. enable
ENBL_RINT1        .set    0080h       ;serial port 1 receive int. enable
ENBL_TINT0        .set    0100h       ;timer 0 interrupt enable
ENBL_TINT1        .set    0200h       ;timer 1 interrupt enable
ENBL_DINT         .set    0400h       ;dma interrupt enable (cpu)

BEGIN_CMD_SEND    .set    1           ;Begin sending cmd parameters
END_CMD_SEND      .set    2           ;End sending cmd parameters

CMD_OK            .set    0           ;Received cmd parameters ok
CMD_ERROR         .set   -1           ;Error on receiving command parameters

CMD_FINISH        .set    0           ;Status, command is finished
CMD_LOAD          .set    1           ;Status, command is being loaded
CMD_ACTIVE        .set    2           ;Status, command is currently active

CMD_NOP           .set   10           ;Nop command
CMD_HOST_MR16     .set   11           ;C30 memory read, 16 bit mode
CMD_HOST_MW16     .set   12           ;C30 memory write, 16 bit mode
CMD_HOST_MR32     .set   13           ;C30 memory read, 32 bit mode
CMD_HOST_MW32     .set   14           ;C30 memory write, 32 bit mode
CMD_HOST_DMAR     .set   15           ;C30 memory read via dma, 16 bit mode
CMD_HOST_DMAW     .set   16           ;C30 memory write via dma,16 bit mode
```

```
************************************************************************
*       sysinit                                                        *
*                                                                      *
*       Initialize the TMS320C30 EVM                                   *
*                                                                      *
*       Operations: - Disable/clear interrupts                         *
*                   - Set the data page and stack pointers             *
*                   - Enable the cache                                 *
*                   - Intialize the memory ports                       *
*                   - Intialize the AIC                                *
*                   - Enable INT0 and GIE to handle command interrupts *
*                   - Wait in IDLE loop for interrupts                 *
*                                                                      *
*       Note.  When debugging you may want to change the IDLE ins.     *
*              to a NOP.  The debugger will not terminate the IDLE     *
*              instruction.  The user must either do a reset from the  *
*              debugger or enable/set an interrupt flag to break the   *
*              IDLE instruction                                        *
*                                                                      *
************************************************************************

           .text
sysinit:   xor    ie,ie                ;disable all interrupts
           xor    if,if                ;clear all interrupts
           ldp    PARMS                ;load data page pointer to parmaters
           ldi    @stack_addr,sp       ;load the address into stack pointer
           ldi    CACHE,st             ;load the status register
           ldi    WAIT0,r0             ;get i/o ready setup
           ldi    @mcntlr0,ar0         ;get memcntl reg address
           sti    r0,*ar0              ;set parallel ready
           ldi    WAIT1,r0             ;get i/o ready
           ldi    @mcntlr1,ar0         ;get memcntl reg address
           sti    r0,*ar0              ;set io ready
           call   aicreset             ;routine to reset aic
           ldi    @com_parm,ar0        ;address of host communications parms
           xor    r0,r0                ;clear out host communications
           rpts   7                    ;paramaters
           sti    r0,*ar0++
           or     ENBL_INT0,ie         ;enable interrupt 0
           or     ENBL_GIE,st          ;enable global interrupt
wait_int:  idle                        ;wait for host to generate interrupts
           br     wait_int
```

```
************************************************************************
*      aicreset                                                        *
*                                                                      *
*      Reset and intialize the AIC                                     *
*                                                                      *
*      Operations:- Set up timer 0 to supply AIC master clock          *
*                 - Reset the AIC                                      *
*                 - Initialize the serial ports                        *
*                 - Take AIC out of reset                              *
*                 - Intialize the AIC                                  *
*                 - Enable receive interrupts                          *
*                                                                      *
************************************************************************

aicreset:
           ldi   2,iof               ;xf0 to output, set xf0 to 0
           ldi   @t0_ctladdr,ar0     ;get address of timer control register
           ldi   1,r1                ;tclk0 will equal h1/2
           sti   r1,*+ar0(8)         ;set the period register to 1
           ldi   @t0_ctlinit,r1      ;get timer 0 setup value
           sti   r1,*ar0             ;set timer 0 to run in pulse mode

           ldi   @p0_addr,ar0        ;get address of serial port 0
           ldi   111h,r1
           sti   r1,*+ar0(2)         ;initialize transmit port control
           sti   r1,*+ar0(3)         ;initialize receive port control
           ldi   @p0_global,r1       ;initialize port 0 global control
           sti   r1,*ar0
           xor   r1,r1
           sti   r1,*+ar0(8)         ;set transmit data to 0

           rpts  99
           nop                       ;wait for 50 timer out clocks
           ldi   6,iof               ;set xf0 to 1, !reset AIC

           call  wait_transmit_0     ;setup aic control register
           ldi   3,r1
           sti   r1,*+ar0(8)
           call  wait_transmit_0
           ldi   1a34h,r1
           sti   r1,*+ar0(8)
           ldi   *+ar0(12),r1

           call  wait_transmit_0     ;setup aic transmit and receive
           ldi   3,r1                ;sampling rates
           sti   r1,*+ar0(8)
           call  wait_transmit_0
           ldi   2a7h,r1
           sti   r1,*+ar0(8)
           ldi   *+ar0(12),r1
           xor   if,if               ;clear out all interrupt flags
           or    @enbl_sp0_r,ie      ;enable serial port 0
           rets

wait_transmit_0:
           xor   if,if               ;wait for the transmit interrupt
wloop:     tstb  10h,if              ;flag to be set.
           bz    wloop
           rets

null_int:  reti
transmit0: reti
transmit1: reti
recieve1:  reti
timer1:    reti
```

```
*****************************************************************
*  hcontrol()                                                   *
*                                                               *
*  Interrupt routine that controls host command processing.     *
*  Uses INTO.                                                   *
*                                                               *
*****************************************************************

hcontrol:
           push    st                  ;save registers
           push    r0
           push    ar0
           push    ar6
           push    dp

           ldp   PARMS
           and   @intoff,ie          ;turn off int1,int2, dma
           ldi   @intclr,r0          ;clear old outstanding
           and   r0,if               ;interrupts
           or    ENBL_GIE,st         ;turn global interrupt back on
           ldi   @hostport,ar6       ;load host port address
           ldi   *ar6,r0             ;load control request
           tstb  BEGIN_CMD_SEND,r0   ;test for control start/end
           bz    format              ;format command
           ldi   2,ar6               ;address of interrupt vector 1
           ldi   @int1_cwr,r0        ;change interrupt 1 vector
           sti   r0,*ar6

           ldi   @com_parm,ar0       ;load command parameter address
           ldi   CMD_LOAD,r0         ;set status to command loading
           brd   cntl_done           ;point to status + 1
           sti   r0,*ar0++
           sti   ar0,@cmd_temp       ;save address to temp location
           or    ENBL_INT1,ie        ;enable interrupt 1

           ;Compress command parameters(build 32 bit parms from 16 bit
           ;values)
format:    ldi   @com_parm,ar0       ;load address of parameter block
           addi  2,ar0               ;index into first parameter
           ldi   2,rc                ;compress  the count, source addr
           rptb  compress            ;and dest. addr
           ldi   *+ar0,r0            ;get high half
           lsh   16,r0               ;left justify
           addi  *ar0,r0             ;add with low half
compress:  sti   r0,*ar0++(2)        ;store back to word, point to next

           ;Check for the command type and branch to special handling
cmd_end:   ldi   @com_parm,ar0
           ldi   0,r0
           sti   r0,@wordflag        ;set word flag to init. value
           ldi   *+ar0,r1            ;switch on command code

           cmpi  CMD_HOST_MR16,r1    ;host memory read, 16 bit
           bz    mr16

           cmpi  CMD_HOST_MW16,r1    ;host memory write, 16 bit
           bz    mw16

           cmpi  CMD_HOST_MR32,r1    ;host memory read, 32 bit
           bz    mr32

           cmpi  CMD_HOST_MW32,r1    ;host memory write, 32 bit
           bz    mw32
```

```
            cmpi  CMD_HOST_DMAR,r1    ;host read via dma, 16 bit
            bz    dmaread

            cmpi  CMD_HOST_DMAW,r1    ;host write via dma, 16 bit
            bz    dmawrite

            ldi   CMD_ERROR,r0        ;invalid command, send host error
            sti   r0,*ar6             ;code
            br    cntl_done

cmd_ack:    ldi   @com_parm,ar0
            ldi   CMD_ACTIVE,r0       ;set command status to active
            sti   r0,*ar0
            ldi   CMD_OK,r0           ;tell host that command is initialized
            sti   r0,*ar6             ;and ok to start data xfer

cntl_done:  pop   dp                  ;restore registers
            pop   ar6
            pop   ar0
            pop   r0
            pop   st
            reti

mr16:       brd   cmd_ack
            ldi   @int2_hrd16,r0      ;set int2 vector to service routine
            sti   r0,@int2            ;enable int2
            or    ENBL_INT2,ie

mw16:       brd   cmd_ack
            ldi   @int1_hwr16,r0      ;set int1 vector to service routine
            sti   r0,@int1            ;enable int1
            or    ENBL_INT1,ie

mr32:       brd   cmd_ack
            ldi   @int2_hrd32,r0      ;set int2 vector to service routine
            sti   r0,@int2            ;enable int2
            or    ENBL_INT2,ie

mw32:       brd   cmd_ack
            ldi   @int1_hwr32,r0      ;set int1 vector to service routine
            sti   r0,@int1            ;enable int 1
            or    ENBL_INT1,ie
```

```
*****************************************************************
*  dmaread                                                      *
*                                                               *
*  DMA setup routine to read C30 memory and write to host       *
*  communications register.  The transfers are interrupt driven *
*  from host reads.                                             *
*                                                               *
*  As the communications register is only 16 bits wide the      *
*  upper half of the data bus is garbage. The host must do      *
*  one dummy read on the front end to fill the buffer.          *
*****************************************************************

dmaread:
           ldi   @dma_ctl,ar0        ;get dma control address
           ldi   @com_saddrl,r0      ;fetch source address
           sti   r0,*+ar0(4)         ;set dma source address
           ldi   @hostport,r0        ;load host port address
           sti   r0,*+ar0(6)         ;store dma destination address
           ldi   @com_countl,r0      ;read count value
           sti   r0,*+ar0(8)         ;store dma count value
           brd   cmd_ack             ;branch back to control
           ldi   @dma_rctl,r0        ;fetch dma control word
           sti   r0,*ar0             ;store dma control register
           or    @enbl_eint2,ie      ;enable dma read interrupt

*****************************************************************
*  dmawrite                                                     *
*                                                               *
*  DMA setup routine to read from the communications reg.       *
*  and write to C30 memory.  The transfers are interrupt driven *
*  from host writes.                                            *
*                                                               *
*  As the communications register is only 16 bits wide the      *
*  upper half of the data bus is garbage.                       *
*                                                               *
*****************************************************************

dmawrite:
           ldi   @dma_ctl,ar0        ;get dma control address
           ldi   @hostport,r0        ;load host port address
           sti   r0,*+ar0(4)         ;set dma source address
           ldi   @com_daddrl,r0      ;fetch destination address
           sti   r0,*+ar0(6)         ;store dma destination address
           ldi   @com_countl,r0      ;read count value
           sti   r0,*+ar0(8)         ;store dma count value
           brd   cmd_ack             ;branch back to control
           ldi   @dma_wctl,r0        ;fetch dma control word
           sti   r0,*ar0             ;store dma control register
           or    @enbl_eint1,ie      ;enable dma read interrupt
```

```
*****************************************************************
*  cmd_write()                                                  *
*                                                               *
*  Transfer command parameters from the comm. port to command   *
*  structure.                                                   *
*****************************************************************

cmd_write:
           push  st                   ;save registers
           push  r0
           push  ar6
           push  dp

           ldp   PARMS
           ldi   @hostport,ar6        ;load host port address
           ldi   *ar6,r0              ;load control request
           ldi   @cmd_temp,ar6        ;load current command pointer
           and   -1,r0                ;mask off upper bits
           sti   r0,*ar6++            ;store data to command parameters
           sti   ar6,@cmd_temp        ;store command pointer back to memory

           pop   dp                   ;restore registers
           pop   ar6
           pop   r0
           pop   st
           reti

*****************************************************************
*  hwrite16                                                     *
*                                                               *
*  Read data from communications register and store in C30 mem. *
*  Data is stored as 16 bit value with 16 msb's set to zero.    *
*****************************************************************

hwrite16:
           push  st                   ;save registers
           push  r0
           push  ar6
           push  dp

           ldp   PARMS
           ldi   @hostport,ar6        ;load host port address
           ldi   *ar6,r0              ;fetch first word of data
           and   -1,r0                ;mask off upper data
           ldi   @com_daddrl,ar6      ;load destination address
           sti   r0,*ar6++            ;store data
           sti   ar6,@com_daddrl      ;store data pointer back to memory

           pop   dp                   ;restore registers
           pop   ar6
           pop   r0
           pop   st
           reti
```

```
******************************************************************
*  hread16()                                                     *
*                                                                *
*  Read data from c30 memory and write to communications reg.   *
*                                                                *
******************************************************************

hread16:
           push  st                   ;save registers
           push  r0
           push  ar6
           push  dp

           ldp   PARMS
           ldi   @com_saddrl,ar6      ;load source address
           ldi   *ar6++,r0            ;fetch first word of data
           sti   ar6,@com_saddrl      ;store data pointer back to memory
           ldi   @hostport,ar6        ;load host port address
           sti   r0,*ar6              ;store data

           pop   dp                   ;restore registers
           pop   ar6
           pop   r0
           pop   st
           reti

 *****************************************************************
*  hwrite32                                                      *
*                                                                *
*  Read data from communications register and store in C30 mem. *
*  Data is read from host as two 16 bit values and added to      *
*  form a 32 bit value.                                          *
******************************************************************

hwrite32:
           push  st                   ;save registers
           push  r0
           push  r1
           push  ar6
           push  dp

           ldp   PARMS
           ldi   @hostport,ar6        ;load host port address
           ldi   *ar6,r0              ;load from host port
           ldi   @com_daddrl,ar6      ;load destination address

           ldi   @wordflag,r1         ;load current word size
           tstb  1,r1                 ;if '1' then msw
           bzd   savedata             ;skip over add
           addi  1,r1                 ;increment flag
           sti   r1,@wordflag         ;store flag
           and   -1,r0                ;mask off msw

           lsh   16,r0                ;if this is msw, then left justify
           addi3 r0,*--ar6,r0         ;and add msw to lsw

savedata:  sti   r0,*ar6++            ;store data
           sti   ar6,@com_daddrl      ;store data pointer back to memory

           pop   dp                   ;restore registers
           pop   ar6
           pop   r1
           pop   r0
           pop   st
           reti
```

```
*****************************************************************
*  hread32                                                      *
*                                                               *
*  Read data from C30 memory and store to communications reg.   *
*  Data is read from c30 memory as 32 bit and adjusted to write *
*  16 bit values as required.                                   *
*****************************************************************

hread32:
           push  st                   ;save registers
           push  r0
           push  r1
           push  ar6
           push  dp

           ldp   PARMS
           ldi   @com_saddrl,ar6      ;load source address

           ldi   @wordflag,r1         ;load current word size
           tstb  1,r1                 ;if '1' then send msw
           bzd   senddata             ;skip over shift and addr++ if lsw
           addi  1,r1                 ;increment flag
           sti   r1,@wordflag         ;store flag
           ldi   *ar6,r0              ;fetch data word

           lsh  -16,r0                ;right justify data
           addi  1,ar6                ;increment data pointer

senddata:  sti   ar6,@com_saddrl      ;store data pointer back to memory
           ldi   @hostport,ar6        ;load host port address
           sti   r0,*ar6              ;store data

           pop   dp                   ;restore registers
           pop   ar6
           pop   r1
           pop   r0
           pop   st
           reti

*****************************************************************
*  dmadone                                                      *
*                                                               *
*  Turn off DMA on completion.                                  *
*****************************************************************

dmadone:
           push  st                   ;save registers
           push  r0
           push  ar0
           push  dp

           ldp   PARMS
           ldi   @dma_ctl,ar0         ;get dma control address
           xor   r0,r0
           sti   r0,*ar0              ;turn off dma

           pop   dp                   ;restore registers
           pop   ar0
           pop   r0
           pop   st
           reti
```

```
****************************************************************
*   receive0                                                   *
*                                                              *
*   Receive data from serial port 0, send same data out serial *
*   serial port 0.                                             *
****************************************************************

receive0:  push  st                    ;save registers
           push  r0
           push  ar0
           push  dp

           ldp   PARMS
           ldi   @p0_addr,ar0          ;get port address
           ldi   *+ar0(12),r0          ;read input
           sti   r0,*+ar0(8)           ;send output

           pop   dp                    ;restore registers
           pop   ar0
           pop   r0
           pop   st
           reti
```

Appendix C

TMS320C30 EVM PAL Equations

This appendix contains the programmable logic source for the four PALs used on the TMS320C30 EVM. These PAL equations were reduced with DATA I/O ABEL version 3.2 at a reduction level of 3.

These equations are described:

Section **Page**

C.1 Write Control PAL, UA3 .... C-2

C.2 Read Control PAL, UA4 .... C-4

C.3 I/O Control PAL, UA5 .... C-5

C.4 Interrupt Control PAL, UB5 .... C-7

C.1 Write Control PAL, UA3

```
module WRITE_CNTL
title'
DWG NAME    TMS320C30 EVM
ASSY #      2563910-0001
COMPANY     TEXAS INSTRUMENTS INCORPORATED
ENGR        TONY COOMES
DATE        03/19/90'

       xua3    device          'P16L8';

       ha0     Pin 1;
       ha5     Pin 2;
       ha6     Pin 3;
       ha7     Pin 4;
       ha8     Pin 5;
       ha9     Pin 6;
       ha10    Pin 7;
       ha11    Pin 8;
       aen     Pin 9;
       mapsel0 Pin 13;
       mapsel1 Pin 14;
       hiowr   Pin 18;
       vss     Pin 10;

       persel  Pin 19;
       sbmsel  Pin 17;
       sbmwr   Pin 16;
       wden    Pin 15;
       wclk0   Pin 12;
       vcc     Pin 20;

"      Module I/O mapping
"      sw1 == mapsel0; sw2 == mapsel1
"
"      sw1     sw2      mapsel0  mapsel1    sel    host I/O base
"      ---     ---      -------  -------    ---    -------------
"       on      on           0        0      0      0x240-0x25f
"       on     off           0        1      1      0x280-0x29F
"      off      on           1        0      2      0x320-0x33F
"      off     off           1        1      3      0x340-0x35F
"

    x = .X.;

    addr = [ha11,ha10,ha9,ha8,ha7,ha6,ha5,x,x,x,x,x];
    sel  = [mapsel0,mapsel1];

 equations
       !persel = (!aen &
                  (  ((addr >= ^hA40) & (addr < ^hA60) & (sel == 0))
                   # ((addr >= ^hA80) & (addr < ^HAA0) & (sel == 1))
                   # ((addr >= ^hB20) & (addr < ^hB40) & (sel == 2))
                   # ((addr >= ^hB40) & (addr < ^hB60) & (sel == 3))));

       !sbmsel =  (!aen  &
                  (  ((addr >= ^h240) & (addr < ^h260) & (sel == 0))
                   # ((addr >= ^h640) & (addr < ^h660) & (sel == 0))
                   # ((addr >= ^h280) & (addr < ^h2A0) & (sel == 1))
                   # ((addr >= ^h680) & (addr < ^h6A0) & (sel == 1))
                   # ((addr >= ^h320) & (addr < ^h340) & (sel == 2))
                   # ((addr >= ^h720) & (addr < ^h740) & (sel == 2))
                   # ((addr >= ^h340) & (addr < ^h360) & (sel == 3))
                   # ((addr >= ^h740) & (addr < ^h760) & (sel == 3))));
```

```
        !sbmwr = (!sbmsel & !hiowr & ha0);

        !wden  = sbmwr;

        !wclk0 = (!sbmsel & !hiowr & !ha0);

test_vectors
([ha0, aen,  addr, hiowr, sel] -> [persel , sbmsel, sbmwr, wden, wclk0])
 [ 0 , 0 , ^hA40 ,   0,    0 ] -> [  0    ,   1   ,  1   ,   0 ,   1  ];  "1
 [ 0 , 0 , ^hA80 ,   0,    1 ] -> [  0    ,   1   ,  1   ,   0 ,   1  ];  "2
 [ 0 , 0 , ^hB20 ,   0,    2 ] -> [  0    ,   1   ,  1   ,   0 ,   1  ];  "3
 [ 0 , 0 , ^hB40 ,   0,    3 ] -> [  0    ,   1   ,  1   ,   0 ,   1  ];  "4

 [ 0 , 0 , ^h240 ,   0,    0 ] -> [  1    ,   0   ,  1   ,   0 ,   0  ];  "5
 [ 1 , 0 , ^h640 ,   0,    0 ] -> [  1    ,   0   ,  0   ,   1 ,   1  ];  "6
 [ 1 , 0 , ^h280 ,   0,    1 ] -> [  1    ,   0   ,  0   ,   1 ,   1  ];  "7
 [ 0 , 0 , ^h680 ,   0,    1 ] -> [  1    ,   0   ,  1   ,   0 ,   0  ];  "8

"[ha0,  aen,  addr, hiowr, sel] -> [persel , sbmsel, sbmwr, wden, wclk0]
 [ 0 , 0 , ^h320 ,   0,    2 ] -> [  1    ,   0   ,  1   ,   0 ,   0  ];  "9
 [ 1 , 0 , ^h720 ,   0,    2 ] -> [  1    ,   0   ,  0   ,   1 ,   1  ];  "10
 [ 1 , 0 , ^h340 ,   0,    3 ] -> [  1    ,   0   ,  0   ,   1 ,   1  ];  "11
 [ 0 , 0 , ^h740 ,   0,    3 ] -> [  1    ,   0   ,  1   ,   0 ,   0  ];  "12

 [ 0 , 0 , ^h260 ,   0,    0 ] -> [  1    ,   1   ,  1   ,   0 ,   1  ];  "13
 [ 0 , 0 , ^h660 ,   0,    0 ] -> [  1    ,   1   ,  1   ,   0 ,   1  ];  "14
 [ 0 , 0 , ^h2A0 ,   0,    1 ] -> [  1    ,   1   ,  1   ,   0 ,   1  ];  "15
 [ 0 , 0 , ^h6A0 ,   0,    1 ] -> [  1    ,   1   ,  1   ,   0 ,   1  ];  "16

 [ 0 , 0 , ^h360 ,   0,    3 ] -> [  1    ,   1   ,  1   ,   0 ,   1  ];  "17
 [ 0 , 0 , ^h760 ,   0,    3 ] -> [  1    ,   1   ,  1   ,   0 ,   1  ];  "18
 [ 0 , 0 , ^h400 ,   0,    3 ] -> [  1    ,   1   ,  1   ,   0 ,   1  ];  "19
 [ 0 , 0 , ^h800 ,   0,    3 ] -> [  1    ,   1   ,  1   ,   0 ,   1  ];  "20

"[ha0,  aen,  addr, hiowr, sel] -> [persel , sbmsel, sbmwr, wden, wclk0]
 [ 0 , 1 , ^h240 ,   0,    0 ] -> [  1    ,   1   ,  1   ,   0 ,   1  ];  "21
 [ 0 , 0 , ^h320 ,   1,    2 ] -> [  1    ,   0   ,  1   ,   0 ,   1  ];  "22
 [ 1 , 1 , ^h640 ,   0,    0 ] -> [  1    ,   1   ,  1   ,   0 ,   1  ];  "23
 [ 1 , 0 , ^h720 ,   1,    2 ] -> [  1    ,   0   ,  1   ,   0 ,   1  ];  "24
end WRITE_CNTL
```

C.2 Read Control PAL, UA4

```
module READ_CNTL
title'
DWG NAME    TMS320C30 EVM
ASSY #      2563910-0001
COMPANY     TEXAS INSTRUMENTS INCORPORATED
ENGR        TONY COOMES
DATE        03/19/90'

       xua4    device          'P16L8';

       hiord   Pin 4;
       ha0     Pin 5;
       ha3     Pin 6;
       ha4     Pin 7;
       sbmsel  Pin 8;
       persel  Pin 9;
       vss     Pin 10;

       sbmrd   Pin 19;
       rden0   Pin 18;
       rden1   Pin 17;
       rclk1   Pin 16;
       rden2   Pin 13;
       rden3   Pin 12;
       vcc     Pin 20;

equations

       sbmrd   = rclk1;
       !rden0  = (!sbmsel & !ha0 & !hiord);
       !rden1  = (!sbmsel &  ha0 & !hiord);
       !rclk1  = (!sbmsel & !ha0 & !hiord);
       !rden2  = (  (!persel & !ha4 &  ha3 & !ha0 & !hiord)
                  # (!persel & !ha4 & !ha3 & !ha0 & !hiord));

       !rden3  = (!persel & !ha4 &  ha3 &  ha0 & !hiord);

test_vectors
([hiord, sbmsel, ha0] -> [sbmrd, rden0,   rden1, rclk1])
 [ 1    ,   1   ,  1 ] -> [ 1    ,   1,       1  ,   1  ];
 [ 0    ,   0   ,  0 ] -> [ 0    ,   0,       1  ,   0  ];
 [ 0    ,   0   ,  1 ] -> [ 1    ,   1,       0  ,   1  ];
 [ 1    ,   0   ,  0 ] -> [ 1    ,   1,       1  ,   1  ];
 [ 0    ,   1   ,  0 ] -> [ 1    ,   1,       1  ,   1  ];
 [ 1    ,   0   ,  1 ] -> [ 1    ,   1,       1  ,   1  ];
 [ 0    ,   1   ,  1 ] -> [ 1    ,   1,       1  ,   1  ];

test_vectors
([hiord, persel , ha4,ha3,ha0] -> [rden2    , rden3])
 [  1  ,   1    ,  1 , 1 , 1 ] -> [  1       ,   1  ];
 [  0  ,   0    ,  0 , 1 , 0 ] -> [  0       ,   1  ];
 [  0  ,   0    ,  0 , 1 , 1 ] -> [  1       ,   0  ];
 [  1  ,   1    ,  1 , 0 , 1 ] -> [  1       ,   1  ];
 [  1  ,   1    ,  0 , 0 , 0 ] -> [  1       ,   1  ];
 [  0  ,   0    ,  0 , 0 , 0 ] -> [  0       ,   1  ];
end READ_CNTL
```

C.3 I/O Control PAL, UA5

```
module IO_CNTL
title'
DWG NAME   TMS320C30 EVM
DWG #      2563910-0001
COMPANY    TEXAS INSTRUMENTS INCORPORATED
ENGR       TONY COOMES
DATE       03/19/90'

       xua5      device          'P16L8';

       ha0       Pin 1;
       ha3       Pin 2;
       ha4       Pin 3;
       hiowr     Pin 5;
       persel    Pin 6;
       c3iorw    Pin 7;
       c3iostrb  Pin 8;
       rst_out   Pin 9;
       hreset    Pin 11;
       vss       Pin 10;

       wclk2     Pin 19;
       wclk3     Pin 18;
       c3wclk    Pin 17;
       c3rden    Pin 16;
       cntlwrd   Pin 15;
       sbm_reset Pin 14;
       c30rst    Pin 13;
       vcc       Pin 20;

       x = .X.;
       hr = hreset;
       ro = rst_out;

equations
       !wclk2     =  (!persel & !ha4 &  ha3 & !ha0 & !hiowr)       "data
                    #(!persel & !ha4 & !ha3 & !ha0 & !hiowr);     "cntl word

       !wclk3     = (!persel & !ha4 & ha3 &  ha0 & !hiowr);

       !c3wclk    = (!c3iorw & !c3iostrb);

       c3rden     = (c3iorw  & !c3iostrb);

       !cntlwrd   = (!persel & !ha4 & !ha3 &  !ha0 & !hiowr);

       !c30rst    = (rst_out # hreset);

       !sbm_reset = (hreset # (!persel & ha4 & ha3 & !hiowr));

test_vectors
([ha4,ha3,ha0,hiowr,persel ,hr,ro] -> [wclk2,wclk3,cntlwrd,c30rst,sbm_reset])
 [ 0 , 0 , 0 , 0  ,  0     ,0 ,0 ] -> [ 0   ,  1  , 0      , 1   ,   1     ];
 [ 0 , 1 , 0 , 0  ,  0     ,0 ,0 ] -> [ 0   ,  1  , 1      , 1   ,   1     ];
 [ 0 , 1 , 1 , 0  ,  0     ,0 ,0 ] -> [ 1   ,  0  , 1      , 1   ,   1     ];
 [ 1 , 1 , 0 , 0  ,  0     ,0 ,0 ] -> [ 1   ,  1  , 1      , 1   ,   0     ];
 [ 1 , 1 , 0 , 1  ,  0     ,0 ,0 ] -> [ 1   ,  1  , 1      , 1   ,   1     ];
 [ 1 , 1 , 0 , 0  ,  1     ,0 ,0 ] -> [ 1   ,  1  , 1      , 1   ,   1     ];
 [ 1 , 0 , 0 , 0  ,  0     ,0 ,0 ] -> [ 1   ,  1  , 1      , 1   ,   1     ];
 [ 1 , 0 , 0 , 1  ,  1     ,1 ,0 ] -> [ 1   ,  1  , 1      , 0   ,   0     ];
 [ 1 , 0 , 0 , 1  ,  1     ,0 ,1 ] -> [ 1   ,  1  , 1      , 0   ,   1     ];
```

```
test_vectors
([c3iorw, c3iostrb] ->[ c3wclk, c3rden ])
 [ 1    ,     1    ] -> [   1  ,    0    ];
 [ 1    ,     0    ] -> [   1  ,    1    ];
 [ 0    ,     0    ] -> [   0  ,    0    ];
 [ 0    ,     1    ] -> [   1  ,    0    ];
end IO_CNTL
```

C.4 Interrupt Control PAL, UB5

```
module INT_CNTL
title'
DWG NAME   TMS320C30 EVM
DWG #      2563910-0001
COMPANY    TEXAS INSTRUMENTS INCORPORATED
ENGR       TONY COOMES
DATE       06/01/90'

       xub5      device          'P16R6';


       h3        Pin 1;          "c30 h3 clock
       cntlwrd_  Pin 2;          "control word write
       wclk3_    Pin 3;          "com. buffer write
       rden3_    Pin 4;          "com. buffer read
       oe        Pin 11;
       vss       Pin 10;


       cntlint   Pin 18;         "control word interrupt (c30 int0)
       s0        Pin 17;
       wrint     Pin 16;         "com. buffer write int. (c30 int1)
       s1        Pin 15;
       rdint     Pin 14;         "com. buffer read int. (c30 int2)
       s2        Pin 13;
       vcc       Pin 20;

       c,x = .C.,.X.;

       "states

        idle  = ^b11;            "waiting for interrupt source active
        irdy  = ^b10;            "waiting for interrupt source !active
        ints0 = ^b00;            "interrupt out low 1 clock
        ints1 = ^b01;            "interrupt out low 2 clock

        "convert inputs to positive logic
        cntlwrd = !cntlwrd_;
        wclk3   = !wclk3_;
        rden3   = !rden3_;

        "short names for vectors
        cw = cntlwrd_;
        wc = wclk3_;
        rd = rden3_;

"  Interrupt generation
"
"             ____      ____      ____      ____      ____      ____
" h3        _/    \____/    \____/    \____/    \____/    \____/    \____
"
"
"           _________           _____________________________
" int input  \\\\\\\\_____~ ~__//////
"
"
"           _________________________________                    _________
" int output                                 \__________________/
"
"            | idle    | irdy    |  irdy   |  inst0  |  ints1  |  idle
```

```
"generate control word interrupt, c30 interrupt 0
state_diagram  [cntlint,s0]

       state   idle:
                 IF (cntlwrd) THEN irdy
                 ELSE idle;

       state   irdy:
                 IF (cntlwrd) THEN irdy
                 ELSE ints0;

       state   ints0:
                 GOTO ints1;

       state   ints1:
                 GOTO idle;

"generate write buffer interrupt, c30 interrupt 1
state_diagram  [wrint,s1]

       state   idle:
                 IF (wclk3) THEN irdy
                 ELSE idle;

       state   irdy:
                 IF (wclk3) THEN irdy
                 ELSE ints0;

       state   ints0:
                 GOTO ints1;

       state   ints1:
                 GOTO idle;

"generate read buffer interrupt, c30 interrupt 2
state_diagram  [rdint,s2]

       state   idle:
                 IF (rden3) THEN irdy
                 ELSE idle;

       state   irdy:
                 IF (rden3) THEN irdy
                 ELSE ints0;

       state   ints0:
                 GOTO ints1;

       state   ints1:
                 GOTO idle; @page
```

```
test_vectors
([ h3, cw   , wc,  rd,  oe ] -> [cntlint, s0, wrint, s1, rdint,   s2 ])
"test cntlint
 [ c , 1  , 1 ,   1,  0 ] -> [  x    ,  x,  x  ,  x,  x  ,      x ];
 [ c , 1  , 1 ,   1,  0 ] -> [  x    ,  x,  x  ,  x,  x  ,      x ];
 [ c , 1  , 1 ,   1,  0 ] -> [  x    ,  x,  x  ,  x,  x  ,      x ];
 [ c , 1  , 1 ,   1,  0 ] -> [  x    ,  x,  x  ,  x,  x  ,      x ];
 [ c , 1  , 1 ,   1,  0 ] -> [  x    ,  x,  x  ,  x,  x  ,      x ];
 [ c , 1  , 1 ,   1,  0 ] -> [  1    ,  1,  1  ,  1,  1  ,      1 ];"idle
 [ c , 0  , 1 ,   1,  0 ] -> [  1    ,  0,  1  ,  1,  1  ,      1 ];"irdy
 [ c , 0  , 1 ,   1,  0 ] -> [  1    ,  0,  1  ,  1,  1  ,      1 ];"irdy
 [ c , 1  , 1 ,   1,  0 ] -> [  0    ,  0,  1  ,  1,  1  ,      1 ];"ints0
 [ c , 1  , 1 ,   1,  0 ] -> [  0    ,  1,  1  ,  1,  1  ,      1 ];"ints1
 [ c , 1  , 1 ,   1,  0 ] -> [  1    ,  1,  1  ,  1,  1  ,      1 ];"idle
 [ c , 1  , 1 ,   1,  0 ] -> [  1    ,  1,  1  ,  1,  1  ,      1 ];"idle
 [ c , 1  , 0 ,   1,  0 ] -> [  1    ,  1,  1  ,  0,  1  ,      1 ];"irdy
 [ c , 1  , 0 ,   1,  0 ] -> [  1    ,  1,  1  ,  0,  1  ,      1 ];"irdy
 [ c , 1  , 1 ,   1,  0 ] -> [  1    ,  1,  0  ,  0,  1  ,      1 ];"ints0
 [ c , 1  , 1 ,   1,  0 ] -> [  1    ,  1,  0  ,  1,  1  ,      1 ];"ints1
 [ c , 1  , 1 ,   1,  0 ] -> [  1    ,  1,  1  ,  1,  1  ,      1 ];"idle
 [ c , 1  , 1 ,   1,  0 ] -> [  1    ,  1,  1  ,  1,  1  ,      1 ];"idle
 [ c , 1  , 1 ,   0,  0 ] -> [  1    ,  1,  1  ,  1,  1  ,      0 ];"irdy
 [ c , 1  , 1 ,   0,  0 ] -> [  1    ,  1,  1  ,  1,  1  ,      0 ];"irdy
 [ c , 1  , 1 ,   1,  0 ] -> [  1    ,  1,  1  ,  1,  0  ,      0 ];"ints0
 [ c , 1  , 1 ,   1,  0 ] -> [  1    ,  1,  1  ,  1,  0  ,      1 ];"ints1
 [ c , 1  , 1 ,   1,  0 ] -> [  1    ,  1,  1  ,  1,  1  ,      1 ];"idle

end INT_CNTL
```

Appendix D

TMS320C30 EVM Schematics

This appendix contains the schematics for the TMS320C30 EVM.

NOTES, UNLESS OTHERWISE SPECIFIED:

1. VCC IS APPLIED TO PIN 8 OF ALL 8-PIN IC'S, PIN 14 OF ALL 14-PIN IC'S, PIN 16 OF ALL 16-PIN IC'S, PIN 20 OF ALL 20-PIN IC'S, ETC.
2. GROUND IS APPLIED TO PIN 4 OF ALL 8-PIN IC'S, PIN 7 OF ALL 14-PIN IC'S, PIN 8 OF ALL 16-PIN IC'S, PIN 10 OF ALL 20-PIN IC'S, ETC.
3. RESISTANCE VALUES ARE IN OHMS.
4. CAPACITANCE VALUES ARE IN MICROFARADS.
5. SWITCH S1 SETTINGS

   A. I/O MAP SELECTION

SW1	SW2	I/O BASE ADDRESS SELECTED
ON	ON	0X0240-0X025F
ON	OFF	0X0280-0X029F
OFF	ON	0X0320-0X033F
OFF	OFF	0X0340-0X035F

   B. MICROCOMPUTER/MICROPROCESSOR MODE

SW3	MODE
ON	MICROPROCESSOR
OFF	MICROCOMPUTER

   C. SBM TRI-STATE

SW4	MODE
ON	SBM TRI-STATE
OFF	SBM ACTIVE

6. MEMORY SIZE SELECTION

RAM SIZE	JP1 JUMPERS
16K	1-2, 5-6
64K	3-4, 7-8

7. AC/DIRECT ANALOG OUTPUT

AC COUPLED	JP2 OPEN
DIRECT COUPLED	JP2 JUMPER 1-2

8. HIGHEST VALUES USED

CERAMIC CAPS	C29
TANTALUM CAPS	CT3
RESISTORS	R5

REVISION STATUS OF SHEETS

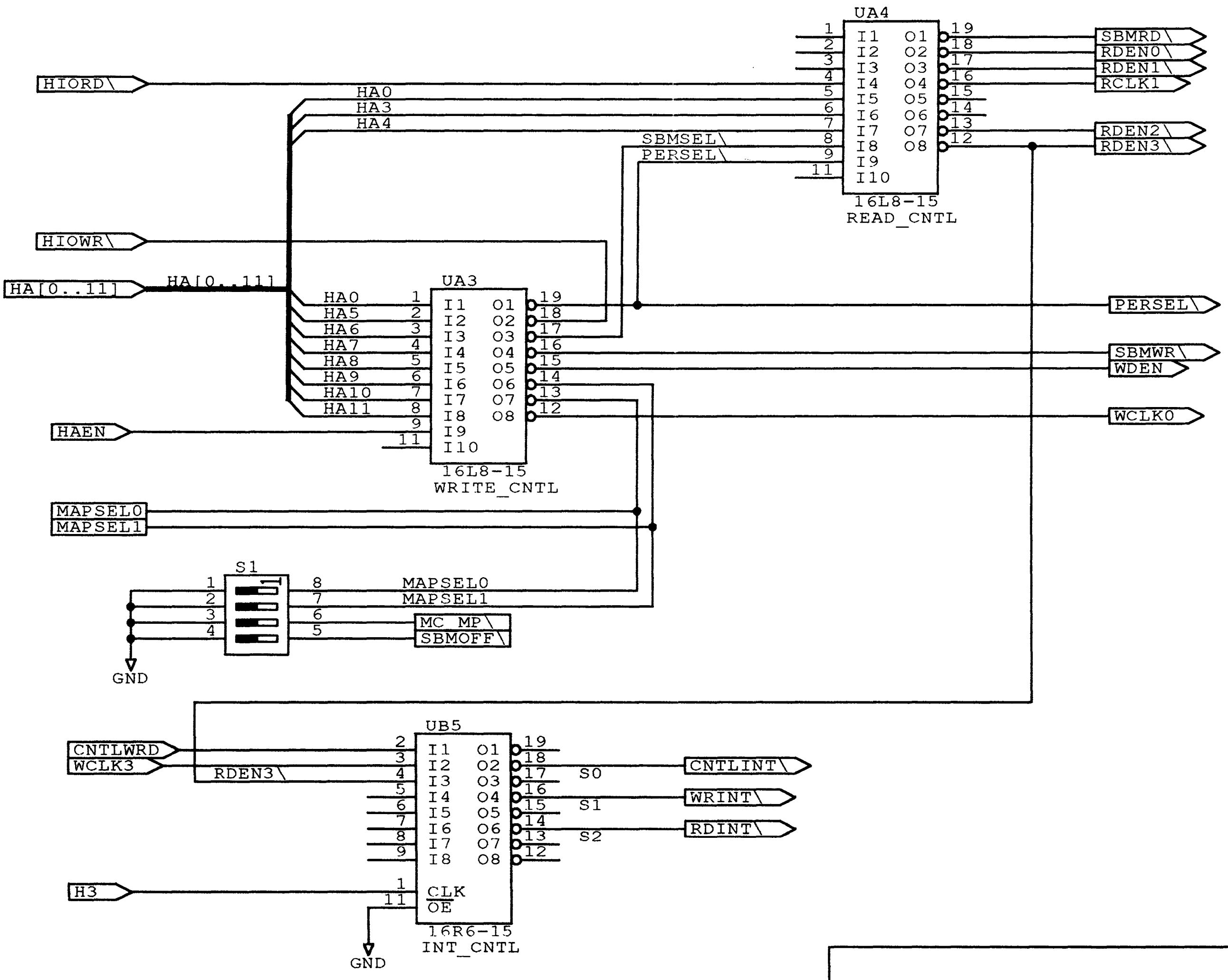
REV							
SH							
REV	*	*	*	*			
SH	8	9	10	11			
REV	*	*	*	*	*	*	*
SH	1	2	3	4	5	6	7

2563910	7119
NEXT ASSY	USED ON
APPLICATION	

DWN	TONY COOMES	DATE 4/20/90
CHK		DATE
ENGR	TONY COOMES	DATE
APVD ENGR		DATE
QA	J.J NANEZ	DATE
APRD MFG	M. JACKSON	DATE
RLSE	J. CLARK	DATE

TEXAS INSTRUMENTS
SEMICONDUCTOR GROUP
HOUSTON TEXAS

Title	TMS320C30 EVM		
Size	Document Number		REV
A	2563912		*
Date:	April 21, 1990	Sheet	1 of 11

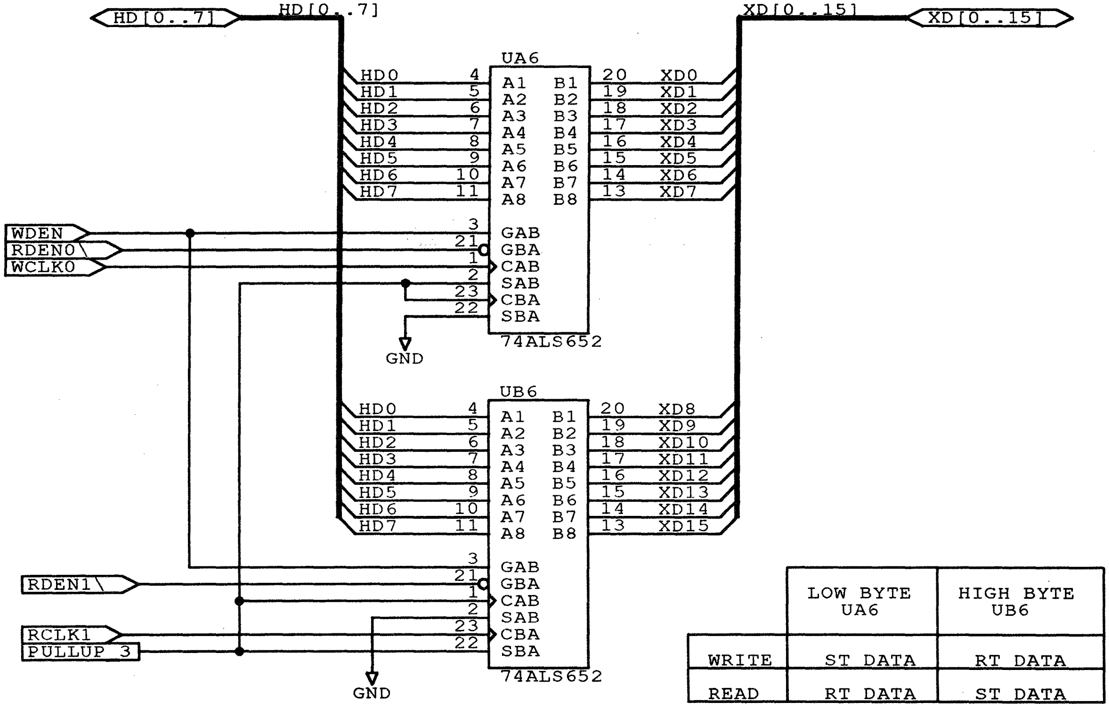


Title

TMS320C30 EVM

Size	Document Number	REV
A	2563912	*

Date: January 13, 1990 Sheet 2 of 11

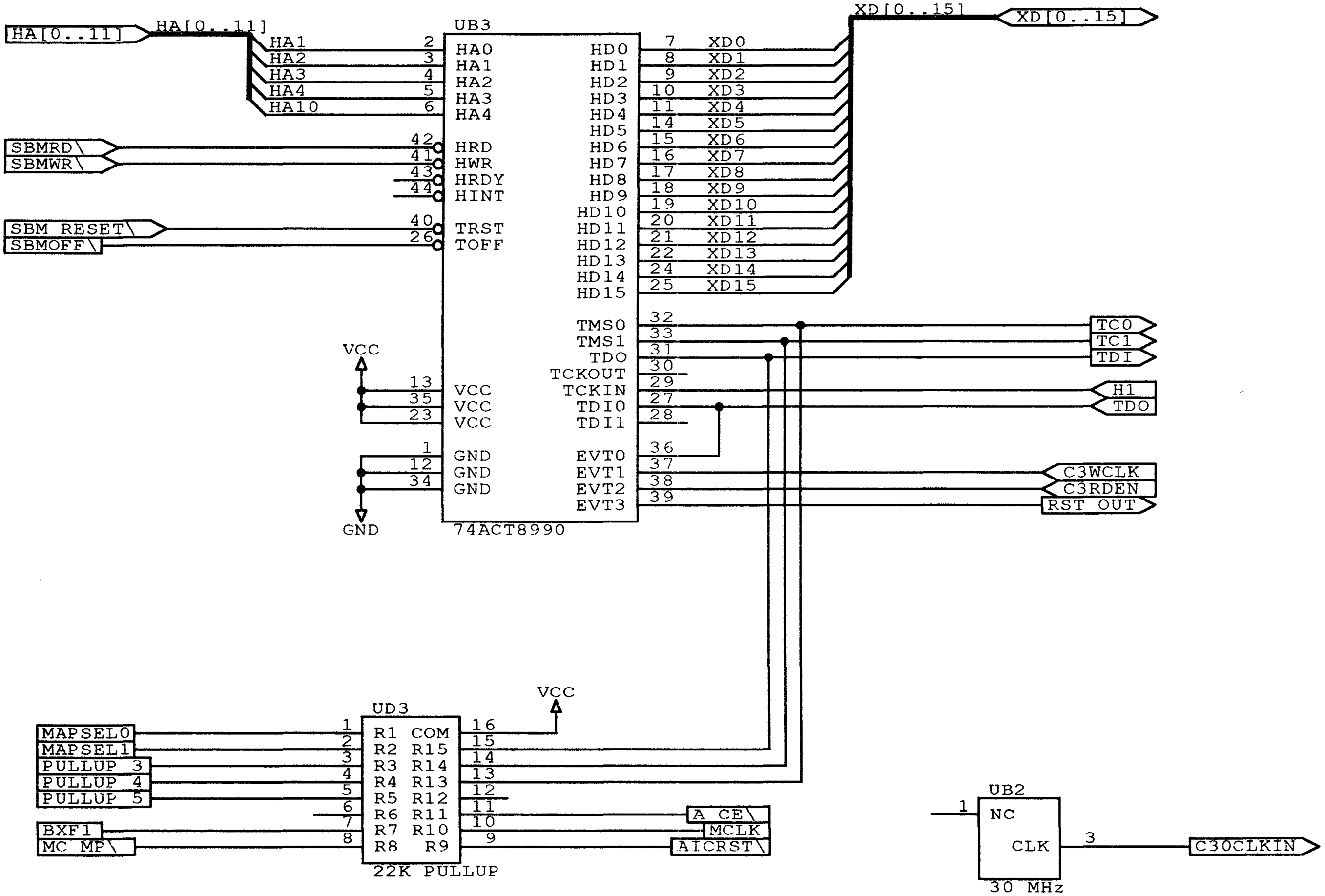


	LOW BYTE UA6	HIGH BYTE UB6
WRITE	ST DATA	RT DATA
READ	RT DATA	ST DATA

Title
TMS320C30 EVM

Size	Document Number	REV
A	2563912	*

Date: January 13, 1990 Sheet 3 of 11

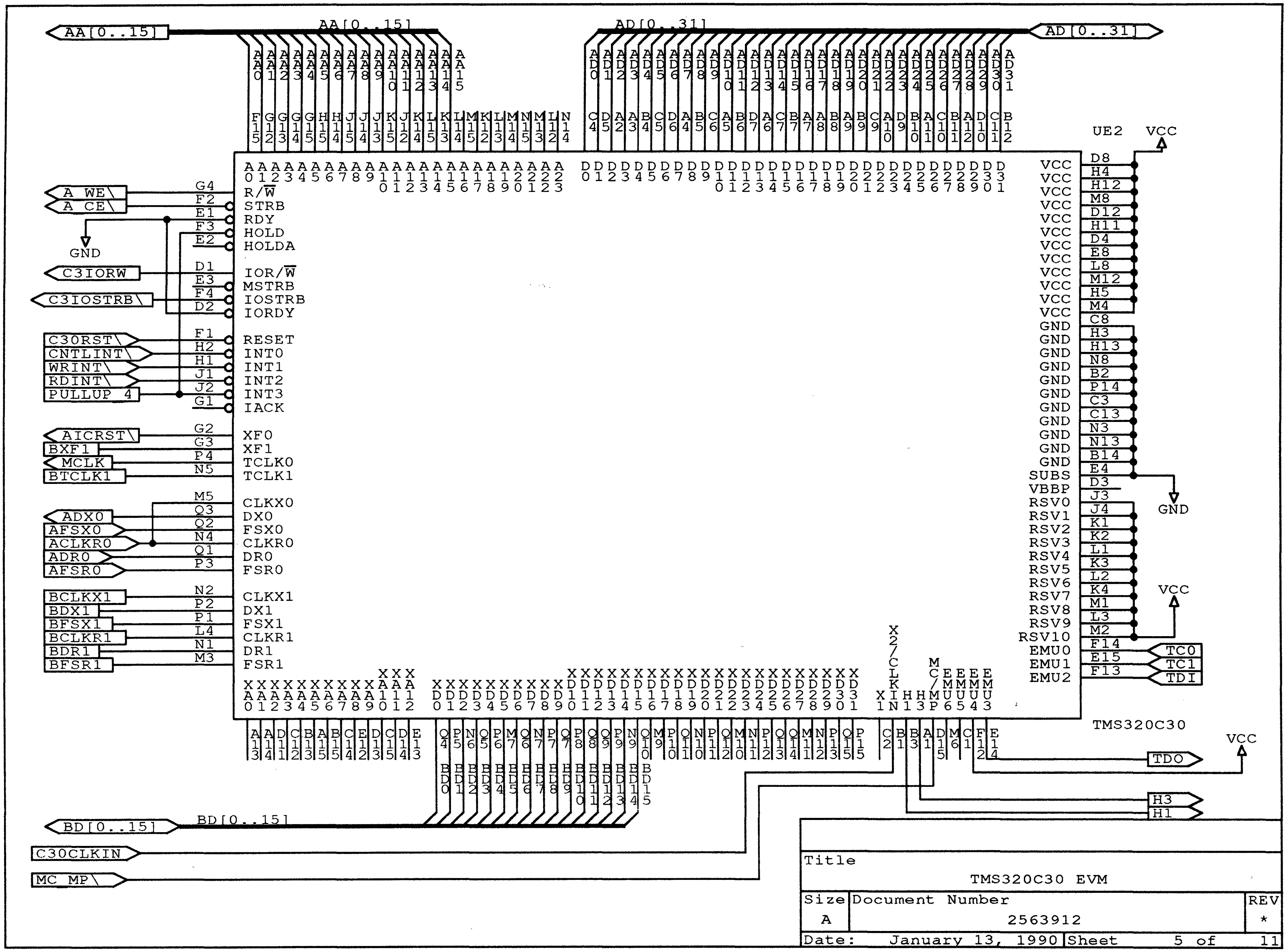


Title

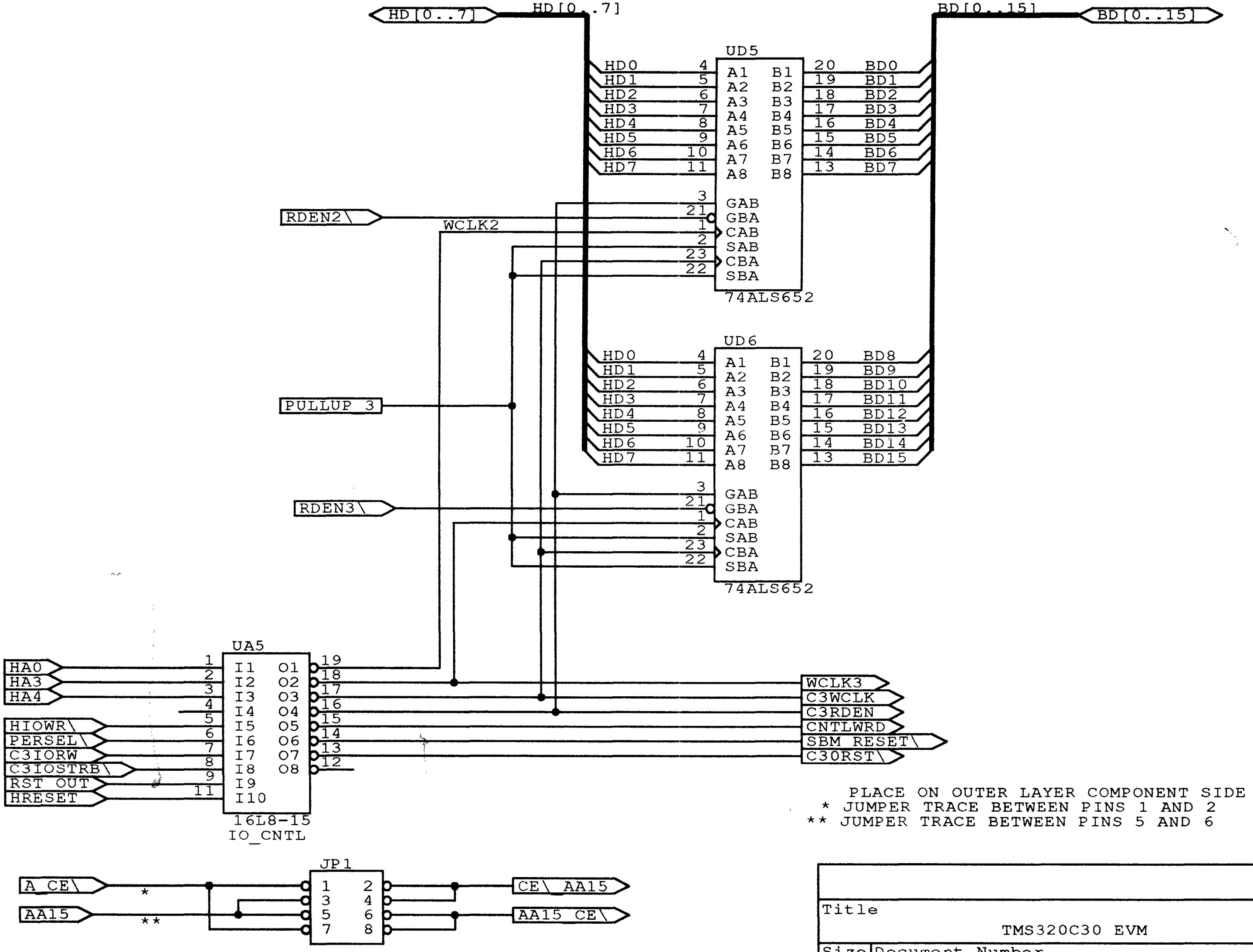
TMS320C30 EVM

Size	Document Number	REV
A	2563912	*

Date: April 3, 1990 | Sheet 4 of 11

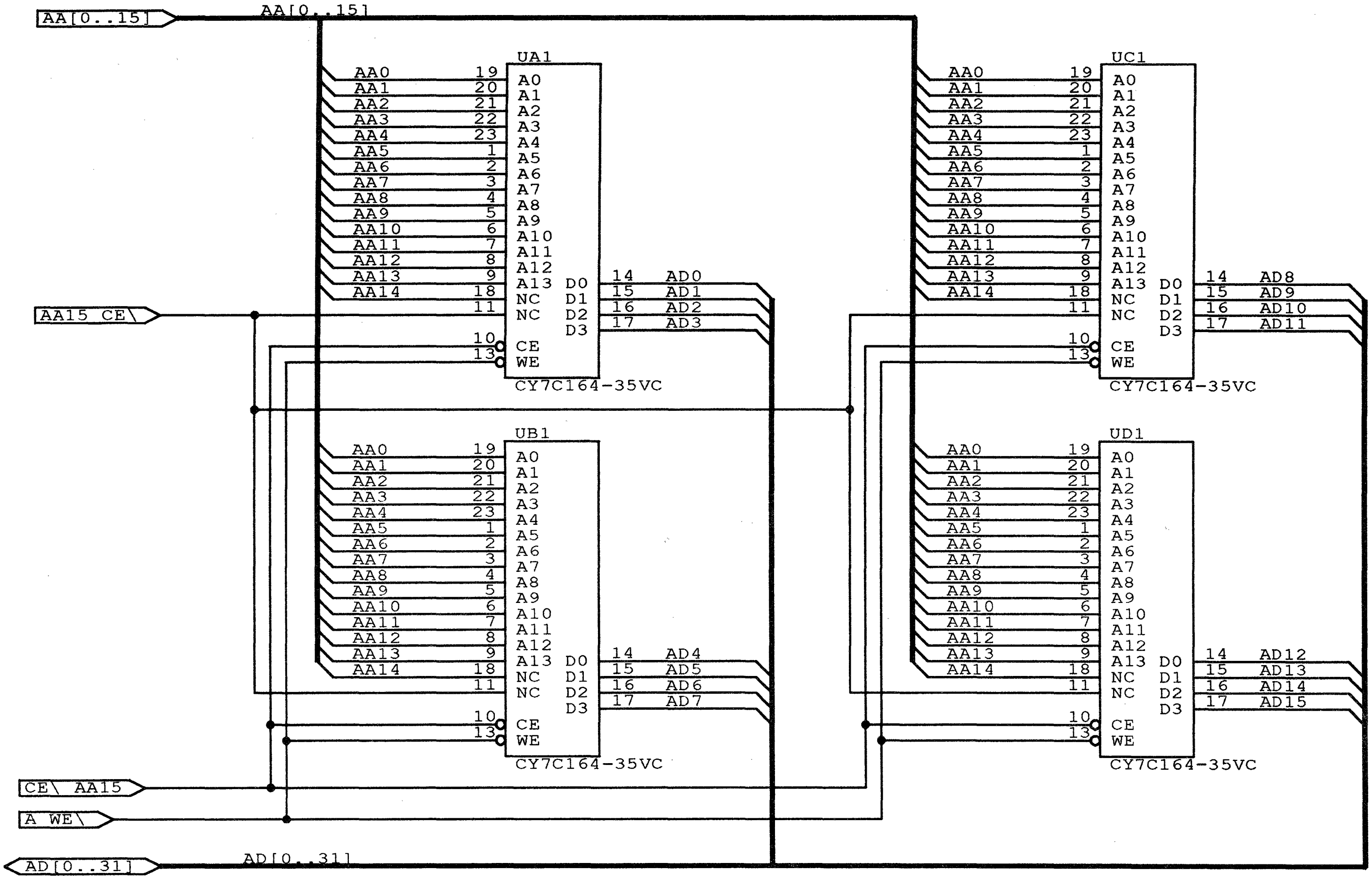


Title		
TMS320C30 EVM		
Size	Document Number	REV
A	2563912	*
Date: January 13, 1990	Sheet 5 of 11	

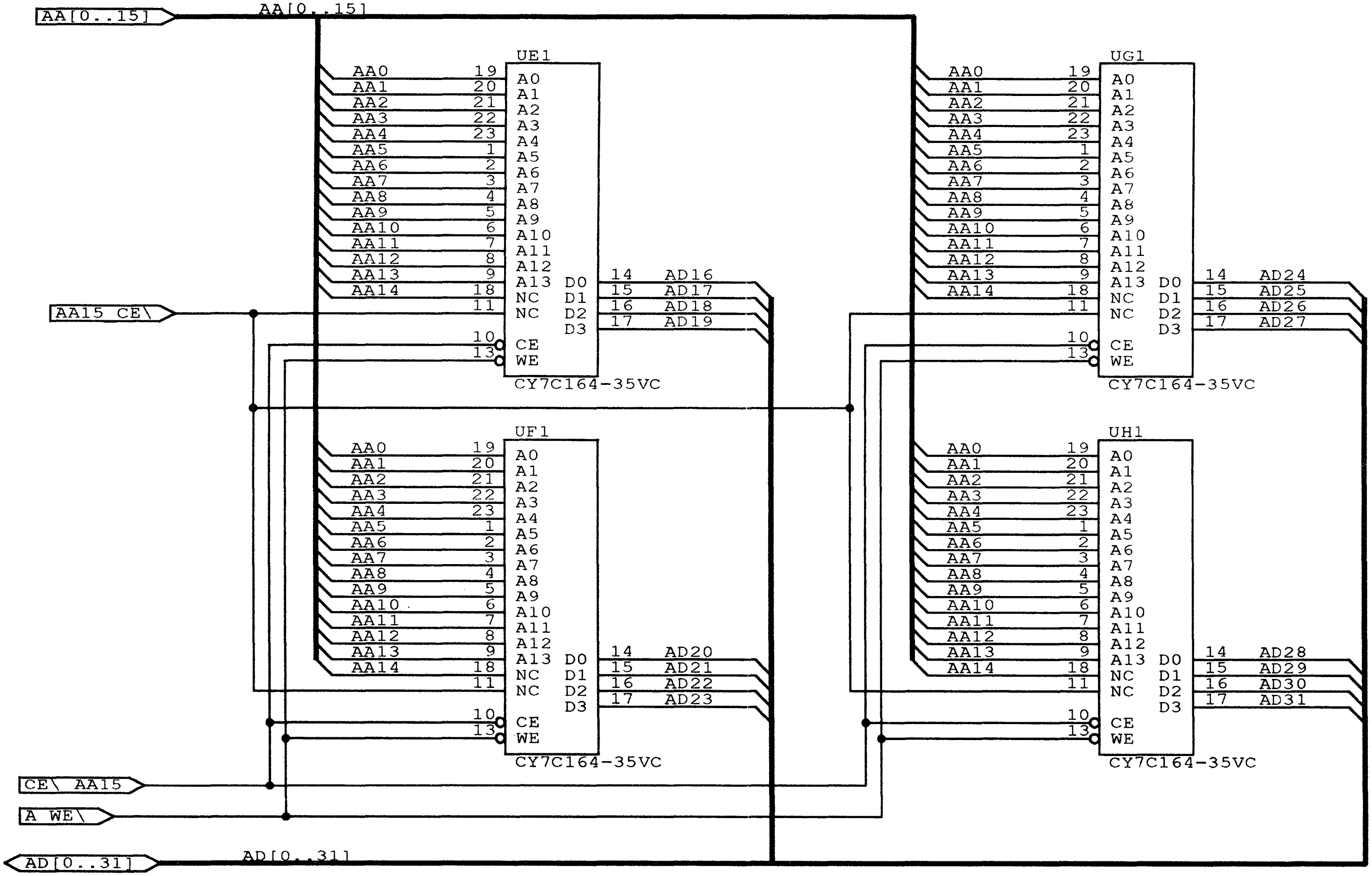


PLACE ON OUTER LAYER COMPONENT SIDE
* JUMPER TRACE BETWEEN PINS 1 AND 2
** JUMPER TRACE BETWEEN PINS 5 AND 6

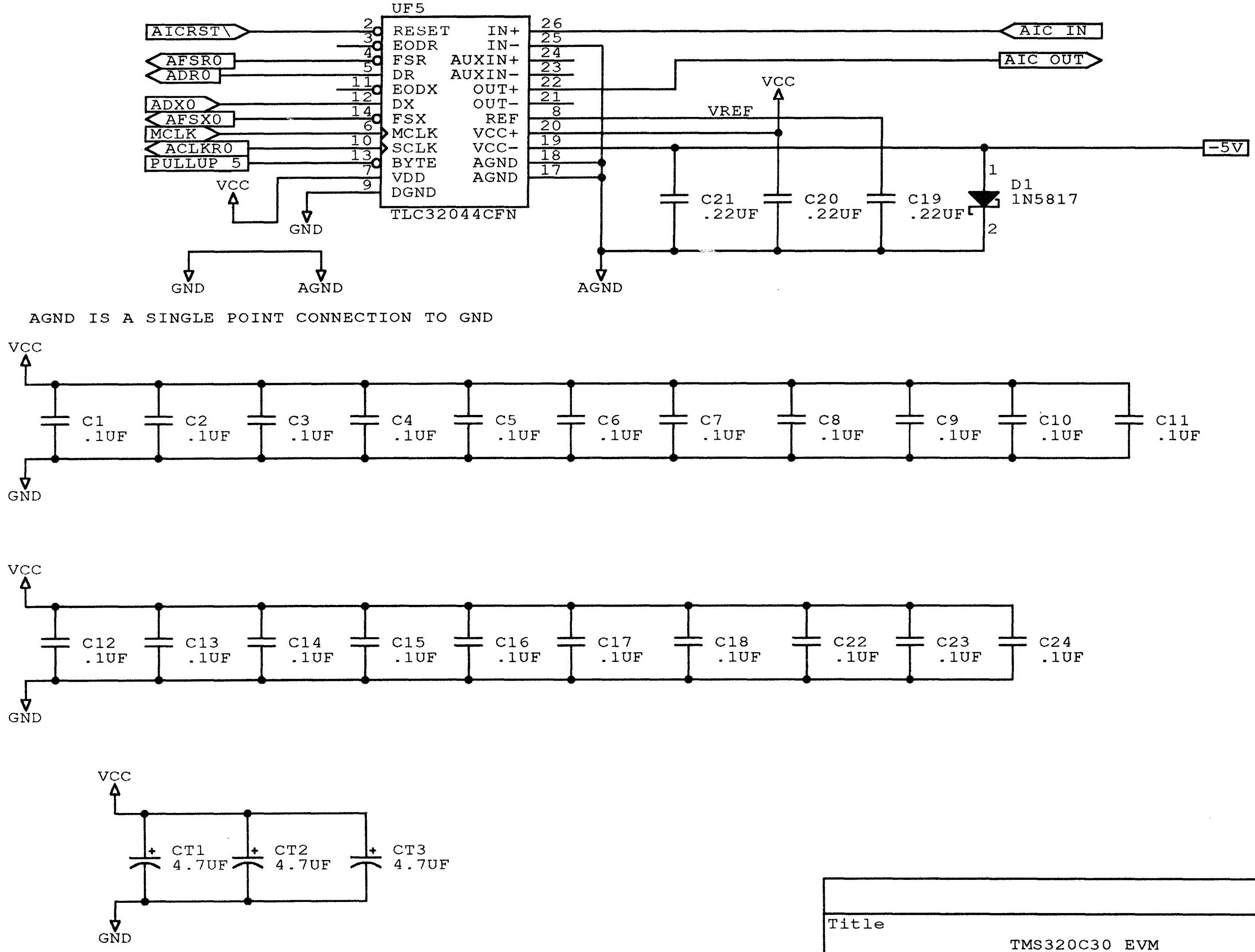
Title	TMS320C30 EVM	
Size A	Document Number 2563912	REV *
Date: January 13, 1990	Sheet 6 of	11



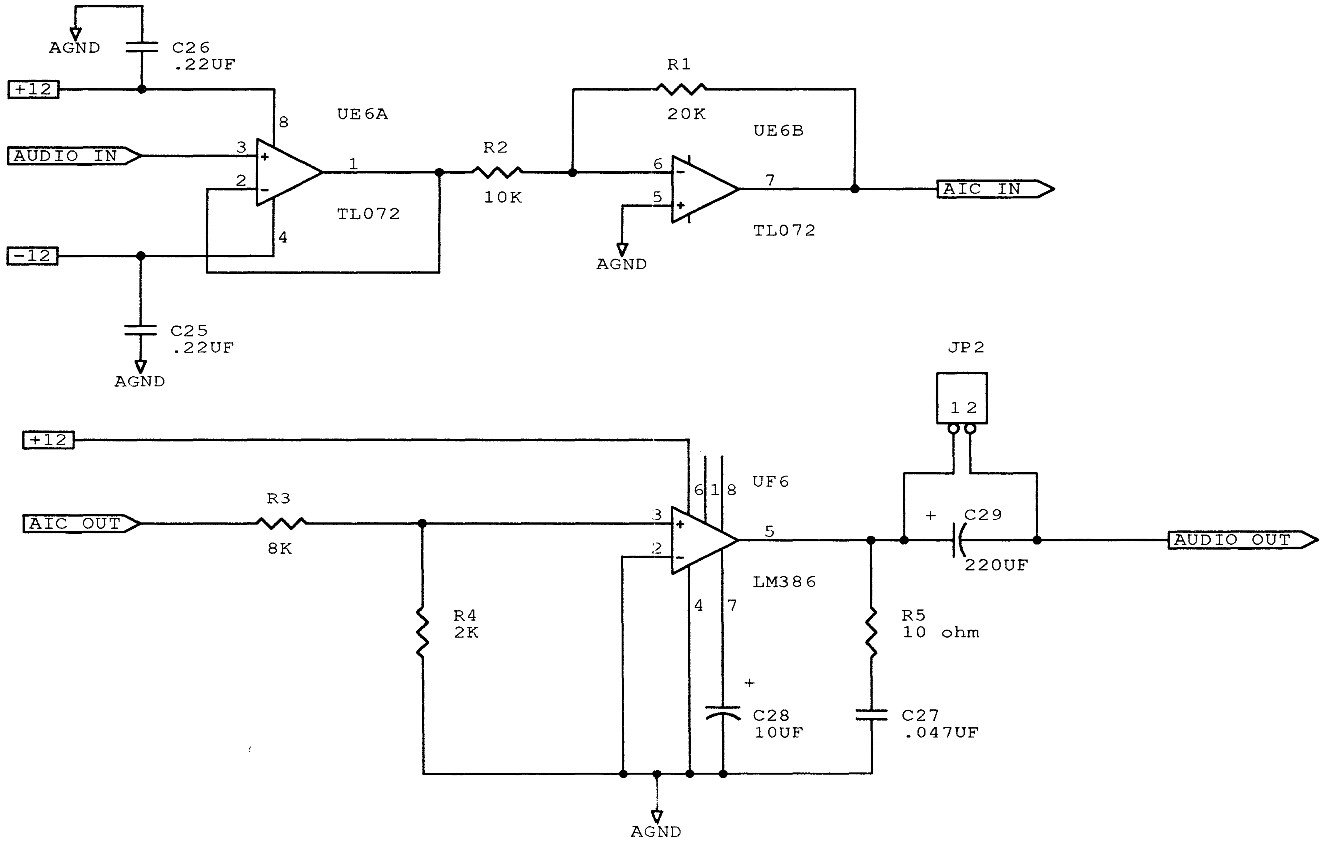
Title	TMS320C30 EVM	
Size A	Document Number 2563912	REV *
Date: January 25, 1990	Sheet 7 of 11	



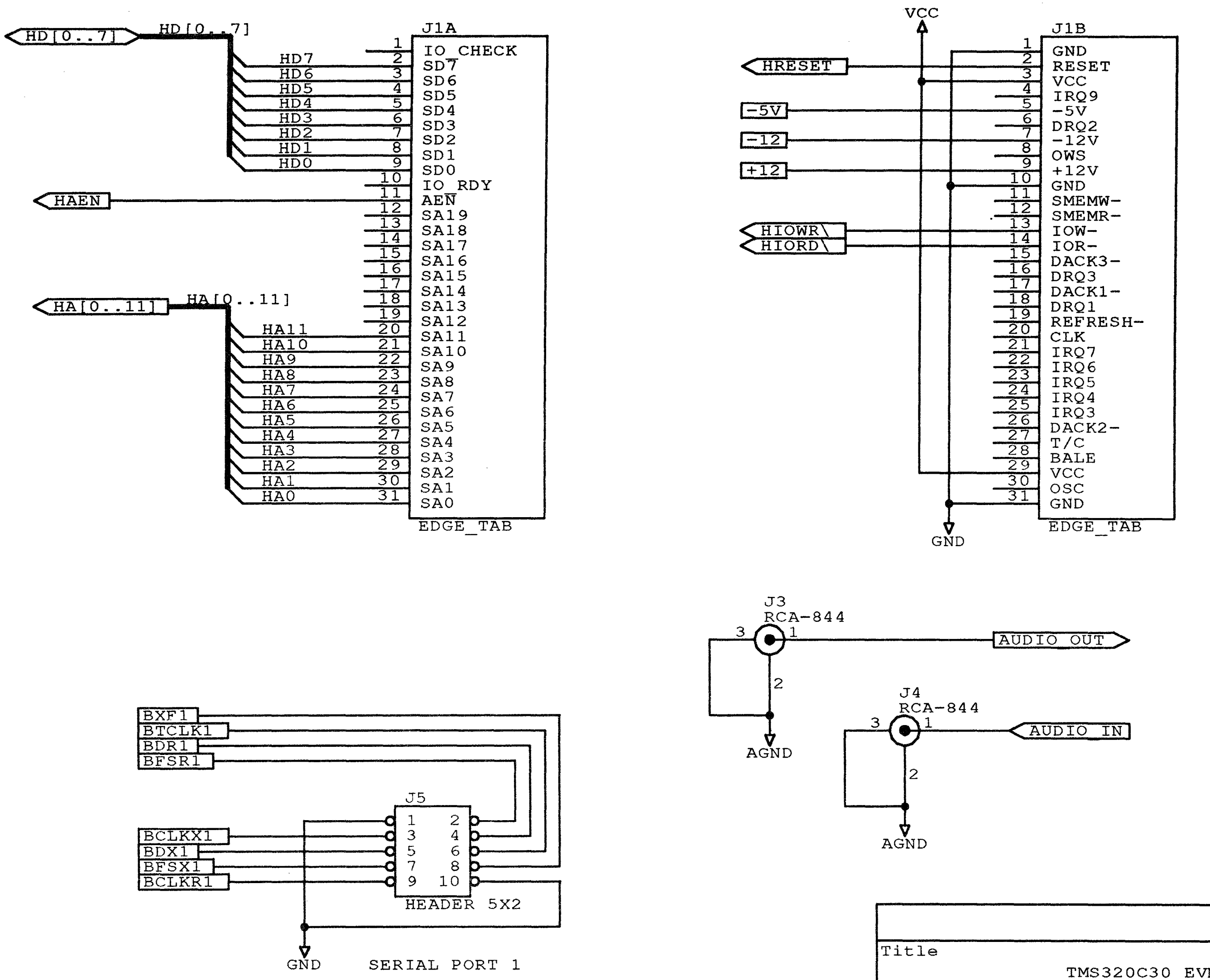
Title		
	TMS320C30 EVM	
Size	Document Number	REV
A	2563912	*
Date: January 13, 1990	Sheet 8 of	11



Title	TMS320C30 EVM		
Size A	Document Number 2563912		REV *
Date: January 13, 1990	Sheet 9 of		11



Title TMS320C30 EVM			
Size A	Document Number 2563912		REV *
Date: January 13, 1990		Sheet 10 of	11



Title TMS320C30 EVM		
Size A	Document Number 2563912	REV *
Date: January 7, 1990	Sheet 11 of 11	

Appendix E

Glossary

Acronyms

AIC: Analog interface controller.

ALU: Arithmetic logic unit.

ASCII: American Standard Code for Information Interchange.

CMOS: Complementary MOS technology.

CPU: Central processing unit.

EPROM: Erasable programmable read-only memory.

EEPROM: Electrically erasable programmable read-only memory.

EVM: Evaluation module.

LSB: Least significant bit.

MSB: Most significant bit.

PROM: Programmable read-only memory.

RAM: Random-access memory.

ROM: Read-only memory.

SRAM: Static RAM.

TBC: Test bus controller.

Commonly Used Terms

A

address: A location in an array of bits, bytes, or words of information.

analog interface controller (AIC): A single channel input/output voice quality analog interface.

arithmetic logic unit (ALU): The section of the computer that carries out all arithmetic operations (addition, subtraction, multiplication, division, or comparison) and logic functions.

archiver: A software program that allows the collection of several individual files into a single file called an archive library.

ASCII: American Standard Code for Information Interchange, 1968. The standard set of 7-bit coded characters (8-bit including parity check) used for information interchange among data processing systems, communications systems, and associated equipment. The ASCII set consists of control characters and graphics characters.

assemble: To prepare a machine-language program from a symbolic language program by substituting absolute operation codes for symbolic operation codes and absolute or relocatable addresses for symbolic addresses.

assembler: A software program that creates a machine-language program from a source file that contains assembly language instructions, directives, and macro directives. The assembler substitutes absolute operation codes for symbolic operation codes, and absolute or relocatable addresses for symbolic addresses.

assembly language: A low-level symbolic programming language, closely resembling machine code language and composed of groups of letters — each group representing a single instruction; allows a computer user to write a program using mnemonics instead of numeric instructions.

attribute: A parameter specifying some characteristic or feature to be applied to subsequent pictorial information.

B

base: 1. A reference value. 2. A number that is multiplied by itself as many times as indicated by an exponent. 3. Same as radix.

bit: A binary digit; usually 1 or 0.

breakpoint: A place in a routine specified by an instruction, instruction digit, or other condition, where the routine may be interrupted by external intervention or by a monitor routine.

byte: An 8-bit sequence of adjacent binary digits operated as a unit.

C

central processing unit (CPU): Part of a computer system that contains the main storage, arithmetic unit, and special register groups. It performs arithmetic operations, controls instruction processing, and provides timing signals and other housekeeping operations.

character: A letter, digit, or symbol that is used as part of the organization, control, or representation of data.

compiler: A translation program that converts a high-level language set of instructions into a target machine's assembly language.

configured memory: Memory that is allocated.

D

data: 1. General term for numbers, letters, symbols, and analog quantities that serve as input for computer processing. 2. Any representations of characters or analog quantities to which meaning, if not information, may be assigned.

display: A visual representation of data.

download: To call for and receive a file from another computer storage medium.

dump: To copy the contents of all or part of a storage, usually internal storage.

E

electrically erasable read-only memory (EEPROM): An integrated-circuit memory that has an internal switch to permit a user to erase the contents of the chip and write new contents into it by means of electrical signals.

erasable programmable read-only memory (EPROM): A read-only memory in which stored data can be erased by ultraviolet light or other means and reprogrammed bit by bit with appropriate voltage pulses.

embedded in-system emulation (EISE): Full emulation support that is embedded in the target system.

evaluation module (EVM): A low-cost evaluation tool that includes a processor, memory, and a host interface.

F

fetch: That portion of a computer cycle during which the next instruction is retrieved from memory.

flag: A binary status indicator whose state indicates whether a particular condition has occurred or is in effect.

H

housekeeping: Those operations or routines that do not contribute directly to the solution of a computer program, but rather to the organization of the program.

I

icon: A graphic symbol representing a menu item.

instruction: A statement that specifies an operation and the values or locations of operands.

instruction set: A set of operation codes for a particular computer or family of processors.

interrupt: To stop a process in such a way that it can be resumed.

L

language: A set of representations, conventions, and rules used to convey information.

linker: A software tool that combines object files to form an object module, which can be loaded into memory and executed.

load: To enter data into storage or working registers.

loop: A sequence of instructions executed repeatedly until a terminal condition prevails.

LSB: Least significant bit.

M

macro: A program made up of one or more sequences of statements or instructions, each sequence represented by a symbolic name and grouped into single instructions.

map file: An output file created by the linker that shows the memory configuration, section composition and allocation, and symbols with the addresses where they are defined.

memory: The section of the computer where instructions and data are stored; synonymous with storage.

memory map: A map of target system memory space that is partitioned into functional blocks.

microcomputer: An integrated circuit that consists of a microprocessor, controller, storage registers, some sort of ALU, and memory.

microprocessor: An integrated circuit that can be programmed with stored instructions to perform a wide variety of functions.

mnemonic: An instruction name that the assembler translates into machine code.

MSB: Most significant bit.

O

object file: A file that has been assembled or linked and contains machine-language object code.

operand: Any one of the quantities entering into or arising from an operation, such as the arguments or parameters of an assembly language instruction, assembler directive, or macro directive.

operation: 1. A defined action; namely, the act of obtaining a result from one or more operands in accordance with a rule that completely specifies the result of any permitted combination of operands. 2.The set of such acts specified by rule, or the rule itself. 3. The act specified by a single computer instruction. 4. A program step undertaken or executed by a computer, i.e., addition, multiplication, extraction, comparison, shift, transfer, etc. 5. The specific action performed by a logic element.

P

programmable read-only memory (PROM): A large-scale integrated circuit chip for storing digital data. It can be erased with ultraviolet light and reprogrammed, or it can be programmed only once, either at the factory or in the field.

R

random-access memory (RAM): A memory element that can be written to, as well as read.

read-only memory (ROM): A semiconductor storage element containing permanent data that cannot be changed.

real time: The actual time during which the physical process of a computation transpires in order that results of the computation interact with the physical process.

register: Temporary storage area for digital data.

S

scrolling: Moving through text strings or graphic strings vertically or horizontally.

simulator: A device, system, or computer program that represents certain features of the behavior of a physical or abstract system.

software: A set of computer programs, procedures, and associated documentation concerned with the operation of a data processing system, e.g., compilers, library routines, manuals, circuit diagrams, etc.

static random-access memory (SRAM): A read-write random-access memory whose storage cells are made up of four to six transistors forming flip-flop elements that indefinitely remain in a given state until the information is intentionally changed, or the power to the memory circuit is turned off.

symbol: A programmer-defined letter, numeral, sign, or other mark that represents the location of a particular datum item, instruction, routine, value, or address.

syntax: The grammatical and structural rules of a language. All higher-level programming languages possess a formal syntax.

T

target memory: Physical memory in a device into which executable object code is loaded.

test bus controller (TBC): A device that is embedded in the target system to support in-system emulation. The TBC provides scan path control of a processor and access to all its registers and memory.

U

unconfigured memory: Memory that is not defined as part of the device's memory map.

W

window: A specified rectangular area of virtual space shown on the display screen.

word: The fundamental unit of storage capacity, usually considered to be more than eight bits in length; a set of bits composing the smallest unit of addressable computer memory.

X

XDS: A hardware development system that provides hardware debugging capabilities in a target system.

Index

A

address decoder, UA3, 2-6
address space switch settings, 2-3
AIC
  initialization, 2-20
  input circuitry, 2-18
  output circuitry, 2-18
  UF5, 2-17
analog I/O characteristics, 2-19
analog interface controller, UF5, 2-17
audio data processing, 2-19

C

communications port, mapping, 2-15
communications port interface, description, 2-10
Cypress CY7C164-35VC, UA1,UB1,UC1,UD1,UE1,UF1,UG1,UH1, 1-3

D

data transfer synchronization
  description, 2-8
  synchronized reads, 2-9
  synchronized writes, 2-8
default switch settings, 2-3
  host I/O memory mapping, 2-3

E

embedded emulation, definition, 1-3
emulation port connections, TBC to TMS320C30, 2-6
equations
  generator state equations, 2-13
  I/O control pal, UA5, C-5
  interrupt control pal, UB5, C-7
  read control pal, UA4, C-4
  write control pal, UA3, C-2
  write interrupt generator, 2-13
EVM analog circuitry, 1-3
EVM analog I/O characteristics, 2-19
EVM board, description, 2-2
EVT0 — EVT3, description, 2-7
EVT3, resetting the TMS320C30, 2-16
examples, I/O base address generation, 2-4
external events, EVT0 — EVT3, 2-7
external serial port interface, 2-20

F

format, MPSD, 2-5

H

hardware-driven interrupts, 2-11
host I/O, read/write control signals, 2-6
host I/O mapping, requirements, 2-3
host I/O memory mapping, default switch settings, 2-3

I

I/O base address generation, 2-4
  example, 2-4
initialization, AIC, 2-20
INT0—INT2, TMS320C30 interrupts, 2-10
  description, 2-11
interface
  host/TBC, 2-5
    8-bit to 16-bit conversion, 2-5
  TBC, 2-5
interface connections, TMS320C30 to AIC, 2-17

interfaces
  EVM analog I/O characteristics, 2-19
  external serial port, 2-20
  TMS320C30 memory, 2-17

interrupt generator, UB5, 2-13

interrupts
  flag generation, 2-14
  generation and detection, 2-13
  generator state equations, 2-13
  hardware-driven, 2-11
  INT0—INT2, 2-10
    description, 2-11
  level sensitive, 2-13
  software-driven, 2-11
  timing, 2-12
  write equations, 2-13

J

J3, output jack, 1-3

J4, input jack, 1-3

J5 pinouts, 2-20

jacks
  J3, 2-2
    output, 1-3
  J4, input, 1-3

jacks J4, 2-2

M

mapping, communications port, 2-15

memory interface, TMS320C30, 2-17

memory mapping, offsets, 2-4, 2-16

O

offsets, EVM register mapping, 2-4, 2-16

op amps
  TL072
    UE6A, 2-18
    UE6B, 2-18
  UE6A, 2-18
  UE6B, 2-18

P

PALs
  equations
    UA3, C-2
    UA4, C-4
    UA5, C-5
    UB5, C-7
  host interface logic, 2-6
  TIBPAL16L8-15
    UA3, 2-6
    UA4, 2-6
  UA5, IO_CONTROL, 2-9
  UB5, INT_CONTROL, 2-11

parameters, TMS320C30 reset, 2-9

pinouts, J5, 2-20

power amp, UF6, LM386, 2-18

power amps, LM386, UF6, 2-18

R

read/write
  host I/O control, 2-6
  TBC, 2-6

READ_CONTROL, UA4, 2-6

reset parameters, TMS320C30, 2-9

resetting the TMS320C30, 2-9
  EVT3, 2-16

S

serial port 0, 1-3

serial port 1, 1-3

SN74ACT8990, TBC, 1-3

software-driven interrupts, 2-11

SRAM, Cypress CY7C164-35VC,
  UA1,UB1,UC1,UD1,UE1,UF1,UG1,UH1, 1-3

SW1, address space switch settings, 2-3

synchronization
  synchronized reads, 2-9
  synchronized writes, 2-8

synchronized reads, 2-9

synchronized writes, 2-8

sytem resets, 2-16

T

TBC
  description, 2-5
  emulation port connections, 2-6
  interface, 2-5
  read/write signals, 2-6

  SN74ACT8990, 1-3
timing, interrupts, 2-12
TLC32044, UF5, 2-17
TMS320C30, emulation port connections, 2-6

U

UA3
  address decoder, 2-6
  control signal generator, 2-6
  PAL equation, C-2
  TIBPAL16L8-15, 2-6
  WRITE_CONTROL, 2-6
UA4
  PAL equation, C-4
  READ_CONTROL, 2-6
  TIBPAL16L8-15, 2-6
UA5
  IO_CONTROL, 2-9
  PAL equation, C-5
  TIBPAL16L8-15, IO_CONTROL, 2-9
UB5
  INT_CONTROL, 2-11
  interrupt generator, 2-13
  PAL equation, C-7
  TIBPAL16L8-15, INT_CONTROL, 2-11
UD5
  low-byte communications register, 2-10
  SN74ALS652, 2-10
UD6
  high-byte communications register, 2-10
  SN74ALS652, 2-10
UE6A, AIC op amp, 2-18
UE6B, AIC op amp, 2-18
UF5, TLC32044, 2-17
UF6, AIC power amp, LM386, 2-18

W

WRITE_CONTROL, UA3, 2-6

TI North American Sales Offices

ALABAMA:
Huntsville: (205) 837-7530

ARIZONA:
Phoenix: (602) 995-1007
Tucson: (602) 292-2640

CALIFORNIA:
Irvine: (714) 660-1200
Roseville: (916) 786-9208
San Diego: (619) 278-9601
Santa Clara: (408) 980-9000
Torrance: (213) 217-7010
Woodland Hills: (818) 704-8100

COLORADO:
Aurora: (303) 368-8000

CONNECTICUT:
Wallingford: (203) 269-0074

FLORIDA:
Altamonte Springs: (407) 260-2116
Fort Lauderdale: (305) 973-8502
Tampa: (813) 885-7411

GEORGIA:
Norcross: (404) 662-7900

ILLINOIS:
Arlington Heights: (708) 640-2925

INDIANA:
Carmel: (317) 573-6400
Fort Wayne: (219) 482-3311

IOWA:
Cedar Rapids: (319) 395-9550

KANSAS:
Overland Park: (913) 451-4511

MARYLAND:
Columbia: (301) 964-2003

MASSACHUSETTS:
Waltham: (617) 895-9100

MICHIGAN:
Farmington Hills: (313) 553-1500
Grand Rapids: (616) 957-4200

MINNESOTA:
Eden Prairie: (612) 828-9300

MISSOURI:
St. Louis: (314) 994-2100

NEW JERSEY:
Iselin: (201) 750-1050

NEW MEXICO:
Albuquerque: (505) 345-2555

NEW YORK:
East Syracuse: (315) 463-9291
Fishkill: (914) 897-2900
Melville: (516) 454-6600
Pittsford: (716) 385-6770

NORTH CAROLINA:
Charlotte: (704) 527-0933
Raleigh: (919) 876-2725

OHIO:
Beachwood: (216) 464-6100
Beavercreek: (513) 427-6200

OREGON:
Beaverton: (503) 643-6758

PENNSYLVANIA:
Blue Bell: (215) 825-9500

PUERTO RICO:
Hato Rey: (809) 753-8700

TENNESSEE:
Johnson City: (615) 461-2192

TEXAS:
Austin: (512) 250-7655
Dallas: (214) 917-1264
Houston: (713) 778-6592

UTAH:
Murray: (801) 266-8972

WASHINGTON:
Redmond: (206) 881-3080

WISCONSIN:
Waukesha: (414) 782-2899

CANADA:
Nepean: (613) 726-1970
Richmond Hill: (416) 884-9181
St. Laurent: (514) 335-8392.

TI Regional Technology Centers

CALIFORNIA: Irvine: (714) 660-8140
Santa Clara: (408) 748-2220
GEORGIA: Norcross: (404) 662-7950
ILLINOIS: Arlington Heights: (708) 640-2909
INDIANA: Indianapolis: (317) 573-6400
MASSACHUSETTS: Waltham: (617) 895-9196
MEXICO: Mexico City: 491-70834
MINNESOTA: Minneapolis: (612) 828-9300
TEXAS: Dallas: (214) 917-3881
CANADA: Nepean: (613) 726-1970

TI Authorized North American Distributors

Alliance Electronics, Inc.
Almac Electronics
Arrow/Kierulff Electronics Group
Arrow (Canada)
Future Electronics (Canada)
GRS Electronics Co., Inc.
Hall-Mark Electronics
Marshall Industries
Newark Electronics
Schweber Electronics
Wyle Laboratories
Zeus Components
Rochester Electronics, Inc. (obsolete product only)

TI Distributors

ALABAMA: Arrow/Kierulff (205) 837-6955; Hall-Mark (205) 837-8700; Marshall (205) 881-9235; Schweber (205) 895-0480.

ARIZONA: Arrow/Kierulff (602) 437-0750; Hall-Mark (602) 437-1200; Marshall (602) 496-0290; Schweber (602) 431-0030; Wyle (602) 437-2088.

CALIFORNIA: Los Angeles/Orange County: Arrow/Kierulff (818) 701-7500, (714) 838-5422; Hall-Mark (818) 773-4500, (714) 727-6000; Marshall (818) 407-4100, (714) 458-5301; Schweber (818) 880-9686, (714) 863-0200; Wyle (818) 880-9000, (714) 863-9953; Zeus (714) 921-9000, (818) 889-3838;

Sacramento: Hall-Mark (916) 624-9781; Marshall (916) 635-9700; Schweber (916) 364-0230; Wyle (916) 638-5282;

San Diego: Arrow/Kierulff (619) 565-4800; Hall-Mark (619) 268-1201; Marshall (619) 578-9600; Schweber (619) 495-0015; Wyle (619) 565-9171; Zeus (619) 277-9681;

San Francisco Bay Area: Arrow/Kierulff (408) 745-6600; Hall-Mark (408) 432-4000; Marshall (408) 942-4600; Schweber (408) 432-7171; Wyle (408) 727-2500; Zeus (408) 629-4789.

COLORADO: Arrow/Kierulff (303) 790-4444; Hall-Mark (303) 790-1662; Marshall (303) 451-8383; Schweber (303) 799-0258; Wyle (303) 457-9953.

CONNECTICUT: Arrow/Kierulff (203) 265-7741; Hall-Mark (203) 271-2844; Marshall (203) 265-3822; Schweber (203) 264-4700.

FLORIDA: Fort Lauderdale: Arrow/Kierulff (305) 429-8200; Hall-Mark (305) 971-9280; Marshall (305) 977-4880; Schweber (305) 977-7511;

Orlando: Arrow/Kierulff (407) 333-9300; Hall-Mark (407) 830-5855; Marshall (407) 767-8585; Schweber (407) 331-7555; Zeus (407) 365-3000;

Tampa: Hall-Mark (813) 541-7440; Marshall (813) 573-1399; Schweber (813) 541-5100.

GEORGIA: Arrow/Kierulff (404) 497-1300; Hall-Mark (404) 447-8000; Marshall (404) 923-5750; Schweber (404) 449-9170.

ILLINOIS: Arrow/Kierulff (708) 250-0500; Hall-Mark (312) 860-3800; Marshall (312) 490-0155; Newark (312)784-5100; Schweber (708) 330-2888.

INDIANA: Arrow/Kierulff (317) 299-2071; Hall-Mark (317) 872-8875; Marshall (317) 297-0483; Schweber (317) 843-1050.

IOWA: Arrow/Kierulff (319) 395-7230; Schweber (319) 373-1417.

KANSAS: Arrow/Kierulff (913) 541-9542; Hall-Mark (913) 888-4747; Marshall (913) 492-3121; Schweber (913) 492-2922.

MARYLAND: Arrow/Kierulff (301) 995-6002; Hall-Mark (301) 988-9800; Marshall (301) 622-1118; Schweber (301) 596-7800; Zeus (301) 997-1118.

MASSACHUSETTS: Arrow/Kierulff (508) 658-0900; Hall-Mark (617) 667-0902; Marshall (508) 658-0810; Schweber (508) 694-9100; Wyle (617) 272-7300; Zeus (617) 863-8800.

MICHIGAN: Detroit: Arrow/Kierulff (313) 462-2290; Hall-Mark (313) 462-1205; Marshall (313) 525-5850; Newark (313) 967-0600; Schweber (313) 525-8100;

Grand Rapids: Arrow/Kierulff (616) 243-0912.

MINNESOTA: Arrow/Kierulff (612) 830-1800; Hall-Mark (612) 941-2600; Marshall (612) 559-2211; Schweber (612) 941-5280.

MISSOURI: Arrow/Kierulff (314) 567-6888; Hall-Mark (314) 291-5350; Marshall (314) 291-4650; Schweber (314) 739-0526.

NEW HAMPSHIRE: Schweber (603) 625-2250.

NEW JERSEY: Arrow/Kierulff (201) 538-0900, (609) 596-8000; GRS (609) 964-8560; Hall-Mark (201) 515-3000, (609) 235-1900; Marshall (201) 882-0320, (609) 234-9100; Schweber (201) 227-7880, (609) 273-7900.

NEW MEXICO: Alliance (505) 292-3360.

NEW YORK: Long Island: Arrow/Kierulff (516) 231-1000; Hall-Mark (516) 737-0600; Marshall (516) 273-2424; Schweber (516) 231-2500; Zeus (914) 937-7400;

Rochester: Arrow/Kierulff (716) 427-0300; Hall-Mark (716) 425-3300; Marshall (716) 235-7620; Schweber (716) 424-2222;

Syracuse: Marshall (607) 798-1611.

NORTH CAROLINA: Arrow/Kierulff (919) 876-3132; (919) 725-8711; Hall-Mark (919) 872-0712; Marshall (919) 878-9882; Schweber (919) 876-0000.

OHIO: Cleveland: Arrow/Kierulff (216) 248-3990; Hall-Mark (216) 349-4632; Marshall (216) 248-1788; Schweber (216) 464-2970;

Columbus: Hall-Mark (614) 888-3313;

Dayton: Arrow/Kierulff (513) 435-5563; Marshall (513) 898-4480; Schweber (513) 439-1800; Zeus (513) 293-6162.

OKLAHOMA: Arrow/Kierulff (918) 252-7537; Hall-Mark (918) 254-6110; Schweber (918) 622-8000.

OREGON: Almac (503) 629-8090; Arrow/Kierulff (503) 645-6456; Marshall (503) 644-5050; Wyle (503) 643-7900.

PENNSYLVANIA: Arrow/Kierulff (215) 928-1800; GRS (215) 922-7037; Marshall (412) 788-0441; Schweber (412) 963-6804.

TEXAS: Austin: Arrow/Kierulff (512) 835-4180; Hall-Mark (512) 258-8848; Marshall (512) 837-1991; Schweber (512) 339-0088; Wyle (512) 345-8853;

Dallas: Arrow/Kierulff (214) 380-6464; Hall-Mark (214) 553-4300; Marshall (214) 233-5200; Schweber (214) 247-6300; Wyle (214) 235-9953; Zeus (214) 783-7010;

El Paso: Marshall (915) 593-0706;

Houston: Arrow/Kierulff (713) 530-4700; Hall-Mark (713) 781-6100; Marshall (713) 895-9200; Schweber (713) 784-3600; Wyle (713) 879-9953.

UTAH: Arrow/Kierulff (801) 973-6913; Marshall (801) 485-1551; Wyle (801) 974-9953.

WASHINGTON: Almac (206) 643-9992, (509) 924-9500; Arrow/Kierulff (206) 575-4420; Marshall (206) 486-5747; Wyle (206) 881-1150.

WISCONSIN: Arrow/Kierulff (414) 792-0150; Hall-Mark (414) 797-7844; Marshall (414) 797-8400; Schweber (414) 784-9451.

CANADA: Calgary: Future (403) 235-5325;

Edmonton: Future (403) 438-2858;

Montreal: Arrow Canada (514) 735-5511; Future (514) 694-7710; Marshall (514) 694-8142;

Ottawa: Arrow Canada (613) 226-6903; Future (613) 820-8313; **Quebec City:** Arrow Canada (418) 871-7500;

Toronto: Arrow Canada (416) 670-7769; Future (416) 638-4771; Marshall (416)458-8046;

Vancouver: Arrow Canada (604) 291-2986; Future (604) 294-1166.

TEXAS INSTRUMENTS

Important Notice: Texas Instruments (TI) reserves the right to make changes to or to discontinue any semiconductor product or service identified in this publication without notice. TI advises its customers to obtain the latest version of the relevant information to verify, before placing orders, that the information being relied upon is current.

D590

Printed in the U.S.A.

TI Worldwide Sales Offices

ALABAMA: Huntsville: 4960 Corporate Drive, Suite N-150, Huntsville, AL 35805, (205) 837-7530.

ARIZONA: Phoenix: 8825 N. 23rd Avenue, Suite 100, Phoenix, AZ 85021, (602) 995-1007; **Tucson:** 818 W. Miracle Mile, Suite 43, Tucson, AZ 85705, (602) 292-2640.

CALIFORNIA: Irvine: 17891 Cartwright Drive, Irvine, CA 92714, (714) 660-1200; **Roseville:** 1 Sierra Gate Plaza, Suite 255B, Roseville, CA 95678, (916) 786-9208; **San Diego:** 5625 Ruffin Road, Suite 100, San Diego, CA 92123, (619) 278-9601; **Santa Clara:** 5353 Betsy Ross Drive, Santa Clara, CA 95054, (408) 980-9000; **Torrance:** 690 Knox Street, Building A, Suite 100, Torrance, CA 90502, (213) 217-7010; **Woodland Hills:** 21550 Oxnard Street, Suite 700, Woodland Hills, CA 91367, (818) 704-8100.

COLORADO: Aurora: 1400 S. Potomac Street, Suite 101, Aurora, CO 80012, (303) 368-8000.

CONNECTICUT: Wallingford: 9 Barnes Industrial Park Road, Wallingford, CT 06492, (203) 269-0074.

FLORIDA: Altamonte Springs: 370 S. North Lake Boulevard, Suite 1008, Altamonte Springs, FL 32701, (407) 260-2116; **Fort Lauderdale:** 2950 N.W. 62nd Street, Suite 100, Fort Lauderdale, FL 33309, (305) 973-8502; **Tampa:** 4803 George Road, Suite 390, Tampa, FL 33634, (813) 885-7411.

GEORGIA: Norcross: 5515 Spalding Drive, Norcross, GA 30092, (404) 662-7900.

ILLINOIS: Arlington Heights: 515 W. Algonquin, Arlington Heights, IL 60005, (708) 640-2925.

INDIANA: Carmel: 550 Congressional Drive, Suite 100, Carmel, IN 46032, (317) 573-6400; **Fort Wayne:** 118 E. Ludwig Road, Suite 102, Fort Wayne, IN 46825, (219) 482-3311.

IOWA: Cedar Rapids: 373 Collins Road N.E., Suite 201, Cedar Rapids, IA 52402, (319) 395-9550.

KANSAS: Overland Park: 7300 College Boulevard, Lighton Plaza, Suite 150, Overland Park, KS 66210, (913) 451-4511.

MARYLAND: Columbia: 8815 Centre Park Drive, Suite 100, Columbia, MD 21045, (301) 964-2003.

MASSACHUSETTS: Waltham: 950 Winter Street, Suite 2800, Waltham, MA 02154, (617) 895-9100.

MICHIGAN: Farmington Hills: 33737 W. 12 Mile Road, Farmington Hills, MI 48018, (313) 553-1500; **Grand Rapids:** 3075 Orchard Vista Drive S.E., Grand Rapids, MI 49506, (616) 957-4200.

MINNESOTA: Eden Prairie: 11000 W. 78th Street, Suite 100, Eden Prairie, MN 55344, (612) 828-9300.

MISSOURI: St. Louis: 11816 Borman Drive, St. Louis, MO 63146, (314) 994-2100.

NEW JERSEY: Iselin: Parkway Towers, 485 E. Route 1 South, Iselin, NJ 08830, (201) 750-1050.

NEW MEXICO: Albuquerque: 2820 D Broadbend Parkway N.E., Albuquerque, NM 87207, (505) 345-2555.

NEW YORK: East Syracuse: 6365 Collamer Drive, East Syracuse, NY 13057, (315) 463-9291; **Fishkill:** 300 Westage Business Center, Suite 140, Fishkill, NY 12524, (914) 897-2900; **Melville:** 1895 Walt Whitman Road, P.O. Box 2936, Melville, NY 11747, (516) 454-6600; **Pittsford:** 2851 Clover Street, Pittsford, NY 14534, (716) 385-6770.

NORTH CAROLINA: Charlotte: 8 Woodlawn Green, Charlotte, NC 28217, (704) 527-0933; **Raleigh:** 2809 Highwoods Boulevard, Suite 100, Raleigh, NC 27525, (919) 876-2725.

OHIO: Beachwood: 23775 Commerce Park Road, Beachwood, OH 44122, (216) 464-6100; **Beavercreek:** 4200 Colonel Glenn Highway, Suite 600, Beavercreek, OH 45431, (513) 427-6200.

OREGON: Beaverton: 6700 S.W. 105th Street, Suite 110, Beaverton, OR 97005, (503) 643-6758.

PENNSYLVANIA: Blue Bell: 670 Sentry Parkway, Blue Bell, PA 19422, (215) 825-9500.

PUERTO RICO: Hato Rey: 615 Mercantil Plaza Building, Suite 505, Hato Rey, PR 00918, (809) 753-8700.

TENNESSEE: Johnson City:3000 Bill Garland Road, Johnson City, TN 37601, (615) 461-2192.

TEXAS: Austin: 12501 Research Boulevard, Austin, TX 78759, (512) 250-7655; **Dallas:** 7839 Churchill Way, Dallas, TX 75251, (214) 917-1264; **Houston:** 9301 Southwest Freeway, Commerce Park, Suite 360, Houston, TX 77074, (713) 778-6592.

UTAH: Murray: 5201 South Green Street, Suite 200, Murray, UT 84123, (801) 266-8972.

WASHINGTON: Redmond: 5010 148th Avenue N.E., Building B, Suite 107, Redmond, WA 98052, (206) 881-3080.

WISCONSIN: Waukesha: 20825 Swenson Drive, #900, Waukesha WI 53186, (414) 782-2899.

CANADA: Nepean: 301 Moodie Drive, Mallorn Center, Nepean, Ontario, Canada K2H 9C4, (613) 726-1970; **Richmond Hill:** 280 Centre Street East, Richmond Hill, Ontario, Canada L4C 1B1, (416) 884-9181; **St. Laurent:** 9460 Trans Canada Highway, St. Laurent, Quebec, Canada H4S 1R7, (514) 335-8392.

ARGENTINA: Texas Instruments Argentina Viamonte 1119, 1053 Capital Federal, Buenos Aires, Argentina, 541/748-3699.

AUSTRALIA (& NEW ZEALAND): Texas Instruments Australia Ltd., 6-10 Talavera Road, North Ryde (Sydney), New South Wales, Australia 2113, 2 887-1122; 5th Floor, 418 Street, Kilda Road, Melbourne, Victoria, Australia 3004, 3 267-4677; 171 Philip Highway, Elizabeth, South Australia 5112, 8 255-2066.

AUSTRIA: Texas Instruments GmbH., Hietzinger Kai 101-105, A-1130 Wien, (0222) 9100-0.

BELGIUM: S.A. Texas Instruments Belgium N.V., 11, Avenue Jules Bordetlaan 11, 1140 Brussels, Belgium, (02) 242 30 80.

BRAZIL: Texas Instruments Electronicos do Brasil Ltda., Rua Paes Leme, 524-7 Andar Pinheiros, 05424 Sao Paulo, Brazil, 0815-6166.

DENMARK: Texas Instruments A/S, Marielundvej 46E, 2730 Herlev, Denmark, (42) 91 74 00.

FINLAND: Texas Instruments OY, Ahertajantie 3, P.O. Box 81, 02101 Espoo, Finland, (90) 461-422.

FRANCE: Texas Instruments France, 8-10 Avenue Morane Saulnier-B.P. 67, 78141 Velizy Villacoublay Cedex, France, (1) 30 70 10 03.

GERMANY (Federal Republic of Germany): Texas Instruments Deutschland GmbH., Haggertystrasse 1, 8050 Freising, (08161) 801; Kurfurstendamm 195-196, 1000 Berlin 15, (030) 8 82 73 65; Dusseldorfer Strasse 40, 6236 Eschborn 1, (06196) 80 70; III, Hagen 43/Kibbelstrasse 19, 4300 Essen 1, (0201) 24 25-0; Kirchhorster Strasse 2, 3000 Hannover 51, (0511) 64 68-0; Maybachstrasse II, 7302 Ostfildern 2 (Nellingen), (0711) 34 03-0.

HOLLAND: Texas Instruments Holland B.V., Hogehilweg 19, Postbus 12995, 1100 AZ Amsterdam-Zuidoost, Holland, (020) 5602911.

HONG KONG: Texas Instruments Hong Kong Ltd., 8th Floor, World Shipping Center, 7 Canton Road, Kowloon, Hong Kong, 852-7351223.

TEXAS INSTRUMENTS

IRELAND: Texas Instruments Ireland Ltd., 7/8 Harcourt Street, Dublin 2, Ireland, (01) 78 16 77.

ITALY: Texas Instruments Italia S.p.A., Centro Direzionale Colleoni, Palazzo Perseo-Via Paracelso 12, 20041, Agrate Brianza (Mi), (039) 63221; Via Castello della Magliana, 38, 00148 Rome, (06) 5222651; Via Amendola, 17, 40100 Bologna, (051) 554004.

JAPAN: Texas Instruments Japan Ltd., Aoyama Fuji Building 3-6-12 Kita-aoyama Minato-ku, Tokyo, Japan 107, 03-498-2111; MS Shibaura Building 9F, 4-13-23 Shibaura, Minato-ku, Tokyo, Japan 108, 03-769-8700; Nissho-iwai Building 5F, 2-5-8 Imabashi, Chuou-ku, Osaka, Japan 541, 06-204-1881; Daini Toyota West Building 7F, 4-10-27 Meieki, Nakamura-ku, Nagoya, Japan 450, 052-583-8691; Kanazawa Oyama-cho Daiichi Seimei Building 6F, 3-10 Oyama-cho, Kanazawa, Ishikawa, Japan 920, 0762-23-5471; Matsumoto Showa Building 6F, 1-2-11 Fukashi, Matsumoto, Nagano, Japan 390, 0263-33-1060; Daiichi Olympic Tachikawa Building 6F, 1-25-12, Akebono-cho, Tachikawa, Tokyo, Japan 190, 0425-27-6760; Yokohama Nishiguchi KN Building 6F, 2-8-4 Kita-Saiwai, Nishi-Ku, Yokohama, Kanagawa, Japan 220, 045-322-6741; Nihon Seimei Kyoto Yasaka Building 5F, 843-2, Higashi Shiokohjicho, Higashi-iru, Nishinotoh-in, Shiokohji-dori, Shimogyo-ku, Kyoto, Japan 600, 075-341-7713; Sumitomo Seimei Kumagaya Building 8F, 2-44 Yayoi, Kumagaya, Saitama, Japan 360, 0485-22-2440; 2597-1, Aza Harudai, Oaza Yasaka, Kitsuki, Oita, Japan 873, 09786-3-3211; 3-18-36, Minami, Hatogaya, Saitama, Japan 334, 0482-82-2211; 4260 Aza-takao, Ohaza-kawasaki, Hiji-machi, Hayami-gun, Oita, Japan 879-15, 0977-72-1111; 2350 Kihara, Miho-mura, Inashiki-gun, Ibaragi, Japan 300-04, 0298-85-3311.

KOREA: Texas Instruments Korea Ltd., 28th Floor, Trade Tower, 159, Samsung-Dong, Kangnam-ku Seoul, Korea, 2 551 2800.

MEXICO: Texas Instruments de Mexico S.A., Alfonso Reyes 115, Col. Hipodromo Condesa, Mexico, D.F., Mexico 06120, 525/525-3860.

MIDDLE EAST: Texas Instruments, No. 13, 1st Floor Mannai Building, Diplomatic Area, P.O. Box 26335, Manama Bahrain, Arabian Gulf, 973 274681.

NORWAY: Texas Instruments Norge A/S, PB 106, Refstad (Sinsenveien 53), 0513 Oslo 5, Norway, (02) 155090.

PEOPLE'S REPUBLIC OF CHINA: Texas Instruments China Inc., Beijing Representative Office, 7-05 Citic Building, 19 Jianguomenwai Dajje, Beijing, China, (861) 5002255, Ext. 3750.

PHILIPPINES: Texas Instruments Asia Ltd., Philippines Branch, 14th Floor, Ba-Lepanto Building, Paseo de Roxas, Makati, Metro Manila, Philippines, 2 817 6031.

PORTUGAL: Texas Instruments Equipamento Electronico (Portugal) Ltda., Eng. Frederico Ulricho, 2650 Moreira Da Maia, 4470 Maia, Portugal (2) 948 1003.

SINGAPORE (& INDIA, INDONESIA, MALAYSIA, THAILAND): Texas Instruments Singapore (PTE) Ltd., Asia Pacific Division, 101 Thomson Road, #23-01, United Square, Singapore 1130, 350 8100.

SPAIN: Texas Instruments Espana S.A., c/Gobelas 43, Ctra de la Coruna km 14, La Florida, 28023, Madrid, Spain, (1) 372 8051; c/Diputacion, 279-3-5, 08007 Barcelona, Spain, (3) 317 91 80.

SWEDEN: Texas Instruments International Trade Corporation (Sverigefilialen), (visit address: Isafjordsgatan 7, Kista), Box 30, S-164 93 Kista, Sweden, (08) 752 58 00.

SWITZERLAND: Texas Instruments Switzerland AG, Riedstrasse 6, CH-8953 Dietikon, Switzerland, (01) 740 22 20.

TAIWAN: Texas Instruments Supply Company, Taiwan Branch, Room 903, 9th Floor, Bank Tower, 205 Tun Hwa N. Road, Taipei, Taiwan, Republic of China, 2 713 9311.

UNITED KINGDOM: Texas Instruments Ltd., Manton Lane, Bedford, England, MK41 7PA, (0234) 270 111.

Important Notice: Texas Instruments (TI) reserves the right to make changes to or to discontinue any semiconductor product or service identified in this publication without notice. TI advises its customers to obtain the latest version of the relevant information to verify, before placing orders, that the information being relied upon is current.

A590

Printed in the U.S.A.

TEXAS
INSTRUMENTS

Printed in U.S.A., October 1990
2547305-9761 revision B