Introduction

The SIC3xDSP Board is a development tool that lets you execute and debug your applications program by using the TMS320C3x C Source Debugger. When you connect analog input and output (such as microphone and speaker) to the system, the SIC3xDSP becomes a simple signal analysis tool. You can also transfer your analog data to and from the host PC through the 16-bit communications port of the SIC3xDSP.

The sheer power of the TM320C3x 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 TMS320C3x well suited for these high-end applications also make it appropriate for products like the SIC3xDSP Board, which meets the goal of providing medium to high performance with minimal logic and a low product cost.

The SIC3xDSP Board 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 full size ISA card at a price that eliminates all cost barriers for evaluating the TMS320C3x.

This manual describes the SIC3xDSP board, its features, design details, and the associated application interface software. Use this manual in conjunction with the TMS320C3x C Source Debugger User's Guide, literature number SPRUO53, and associated addenda for instructions on installing and setting up the board.

1.1 General Description

Key Features

* The industry-standard TMS320C3x floating-point DSP, configured for 27-60 MFLOPS.
* 64K-256K words of zero wait-state SRAM on the primary bus.
* Standard 9 and 15 pin DSUB connectors, one 96 pin DIN connector containing all C3x signals, and two 16 pin headers containing both serial ports for expansion or prototyping.
* 32 bit bidirectional PC host communications port.
* IBM PC/AT compatible 16 bit full size ISA card, mappable in one of four I/O locations.

Functional Overview

The interconnects include the emulation interface, the host interface, an expansion interface for mostly analog I/O daughtercards, a serial port interface, and memory.

All code for the SIC3xDSP board can be loaded through the TMS320C3x emulation port or by means of the C31's onchip boot loader. This allows for the ommition of onboard boot PROM/EPROM. Once code has been loaded, the host and TMS320C3x communicate by means of a shared 32 bit bidirectional register.

The register-based interface between the host and TMS320C3x is simple, keeps costs low, and provides a moderate transfer band-width of approximately 500K bytes per second (KBPS). This interface has been tailored for use with the TMS320C3x DMA channel and is fully interrupt driven.

Host I/O Map Requirements

The SIC3xDSP 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 SIC3xDSP will not work in systems that do not conform to the 1K I/O decoding scheme.*

IO Map Switch Settings

You can map the SIC3xDSP into one of four I/O base address ranges by setting switches MAPSEL0 and MAPSEL1. 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. Below is Table 1 that shows the host I/O map usage for the default switch settings, and table 2 shows the SIC3xDSP register address offsets.

Table 1

MAPSEL1	MAPSEL0	Base I/O range
C	A	0x240 - 0x25F
C	B	0x280 - 0x29F
D	A	0x320 - 0x33F
D	B	0x340 - 0x35F

Table 2

Address	Name	Description	Width (bits)	Direction
Base+0x00	CNTRLWD	DSP Function Control	8	R/W
Base+0x04	CONTROL/STATUS	Hardware Control/Status	8	R/W
Base+0x08	LO WORD	Lower 16 bit DSP Data	16	R/W
Base+0x10	HI WORD	Upper 16 bit DSP Data	16	R/W

Base+0x18	RESET	Reset Line	8	W

I/O Base Address Generation

The host accesses the SIC3xDSP registers over the PC/AT I/O expansion bus with the x86 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);
}
```

Test Bus Controller (TBC) Interface

Complete details for the TBC interface can be found in the TMS320C3x Evaluation Module Technical Reference manual, TI document SPRU069.

The most significant aspect of the SIC3xDSP is the emulation support for the TMS320C3x that is embedded into the target system. The SN74ACT8990 test bus controller (TBC) and the TMS320C3x 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 TMS320C3x 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.

The TBC provides a means for the host processor to communicate with a target device through the device emulation port. This is done with the Texas Instruments modular port scan device (MPSD) format.

The host interface to the TBC is quite simple and consists of two registered bidirectional 74FCT652 data transceivers (16 bit interface), two 74ALS245 bidirectional transceivers (16 bit interface), and three PALS.

The TBC provides a direct connection to the TMS320C3x emulation port.

The TBC supports four external events, EVT0 through EVT3. They can be programmed as interrupt sources to the TBC or general purpose bit I/O. The event pins are set up with the EVM_reset function.

Data Transfer Synchronization

Data transfer synchronization between the TMS320C3x and the host is an integral part of the communications protocol. See Chapter 3 of TI's EVM manual for details about the communications protocol between the TMS320C3x 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 polling is recommended, it is not required. The data transfer rate with polling is approximately 200 Kbytes 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 TMS320C3x and the host to ensure that data is not lost in unsynchronized transfers.

> *CAUTION: 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 TMS320C3x operation. Access the TBC only with the software routines provided in this manual or with compatible routines.*

Steps for Synchronized Writes

Execute the following steps to synchronize writes to the SIC3xDSP communications register. Below are the synchronized write parameters for each step.

Step 1: Clear the previous EVT2 write acknowledge.
Step 2: Write the data into the communications register.
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.

Steps for Synchronized Reads

Execute the following steps to synchronize reads to the SIC3xDSP communications register. Below are the synchronized read parameters for each step.
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).

Using the TBC to Reset The TMS320C3x

The TMS320C3x may be reset by toggling the RESET bit in the hardware control/status register (CONTROL/STATUS) when no TBC is present. If the TBC is present, resetting the TMS320C3x via the TBC is implemented by inverting the EVT3 signal and generating the TMS320C3x reset signal when EVT3 is active. Below are the required steps to reset the TMS320C3x through the TBC.

Step 1: Set EVT3 = 1.
Step 2: Set EVT3 = 0.

Host/TMS320C3x Communications Port Interface

In general, host communications to a system such as the SIC3xDSP are provided by means of FIFOs, dual-port SRAMs, or direct interface to SIC3xDSP memory space. These interfaces have there strengths; however, they consume a significant amount of hardware and printed circuit board real estate. Further they are fairly expensive, depending on the implementation.

The SIC3xDSP has a simpler host port interface. This reduces product cost, while maintaining moderate throughput (up to 500 KBPS). One of the main reasons for having elaborate host interfaces in a system such as the SIC3xDSP is for program loading. With either the TBC handling program loading or the on chip TMS320C31 boot loader handling the COFF loading chores, the interface requirements reduce to those of passing data to and from the application software.

On the SIC3xDSP, a 32 bit bidirectional register based host interface was implemented to meet these interface requirements. This interface is similar to that of the TBC interface. The major difference between the TBC and TMS320C3x interface schemes is that communications to the TMS320C3x can be interrupt driven. This means that host accesses to the communications register will generate interrupts to the TMS320C3x by means of the INT0 - INT2 interrupt flags. TMS320C3x accesses to the communications registers generate the TBC EVT1 - EVT2 event flags, which the host polls if the TBC is present. If not present, it can pole the CONTROL/STATUS register.

TMS320C3x Interrupts

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

The SIC3xDSP board uses hardware driven interrupts on the TMS320C3x side of the communications register to maximize performance. This allows the TMS320C3x DMA controller to service data read/write interrupts.

Software-driven interrupts cannot be used by the DMA controller, because the TMS320C3x DMA does not execute code that clears the interrupts.

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 1) that data in the register is properly read or written before an interrupt is generated, and 2) that only one interrupt is generated per host access.

Host Event Polling

While the TMS320C3x side of the communications register is interrupt driven, the host side is polled. When the TMS320C3x reads or writes the communications register, the TBC generates the EVT1

- EVT2 event flags. Additionally, the CONTROL/STATUS register emulates this event if the TBC is absent. 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 SIC3xDSP is passed in blocks. Using host interrupts to service individual accesses to the communications register significantly reduces the data throughput.

TMS320C3x Communications Port Mapping

Table 1 showed the SIC3xDSP register address offsets mapping in the host address space. Table 2-11 shows the communications register mapping in the TMS320C3x address space. The COM_DATA register resides on the entire TMS320C30 I/O expansion bus, while it resides in 0xFF0000 on the TMS320C3X.

System Resets

System reset is essential for proper SIC3xDSP operation. The SIC3xDSP is reset by the host's !RESET signal during system power up. However, you can also reset the SIC3xDSP with software by writing to the SOFT_RESET I/O address location shown in table 2-12.

Protocol Overview

The TMS320C3x 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 TMS320C3x is the slave. This is because the TMS320C3x side of the communications is interrupt-driven, while the host side is polled.

> *NOTE: TMS320C3x 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 when the TMS320C3x is the master and the host PC is the slave.*

The communications protocol presented in this manual capitalizes on the SIC3xDSP's best features. You may wish to modify 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 TMS320C3x is the slave.

The basic elements of the protocol are as follows:

System initialization
    Host
    TMS320C3x
Command preprocessor

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

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

System Initialization

The host initializes the SIC3xDSP to support communications by doing the following:

* It resets the SIC3xDSP, then holds the TMS320C3x in a reset condition.
* It initializes the TBC for default operation.
* It downloads the TMS320C3x monitor/communications code into the SIC3xDSP by means of the TBC.
* It removes the TMS320C3x reset condition.

The TMS320C3x debugger or the stand alone loader is used to accomplish these operations.

Once host initialization is completed, the TMS320C3x initializes the SIC3xDSP. The TMS320C3x is initialized by the source code implemented in the SICINIT.ASM routine. 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 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.

The Command Preprocessor

The command preprocessor consists or two separate operations:

1) COMMAND TRANSFERS, transfer commands to the command structure.
2) COMMAND FORMATTING, formats command parameters and starts command execution.

Host/TMS320C3x Command Transfer and Formatting

The following explains host/TMS320C3x command transfer and formatting sequences:

Step 1: The host initiates a command transfer by writing a command send request to the SIC3xDSP. This generates an INT0 to the TMS320C3x, which prepares the TMS320C3x to accept command parameters.
Step 2: The host sends command parameters to the SIC3xDSP as standard data transfers. Each data transfer generates an INT1 to the TMS320C3x. The TMS320C3x stores the command parameters in the command structure.
Step 3: The host terminates the command parameter transfer by sending a command end to the SIC3xDSP, which generates an INT0 to the TMS320C3x.
Step 4: The TMS320C3x formats the command parameters. Command parameter formatting is necessary because the host sends the parameters as 16-bit values; the TMS320C3x requires 32 bit values.
Step 5: Once the command parameters are formatted, the TMS320C3x 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 TMS320C3x sends a command ready acknowledge back to the host. The host begins data transfer.

Host Writes

The host writes to the communications register. This generates an INT1 to the TMS320C3x. The TMS320C3x responds by reading the data from the communications register, which generates an EVT2 to the TBC.

Host Reads

The host reads data from the communications register and generates an INT2 to the TMS320C3x. The TMS320C3x 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.

SIC3xDSP Command Structure

The SIC3xDSP command structure is defined as follows:

```
com_stat      .word  00000000h  ;command status
com_cmd       .word  00000000h  ;command code

com_counth    .word  00000000h  ;transfer count, high
com_countl    .word  00000000h  ;transfer count, low

com_saddrh    .word  00000000h  ;source address, high
com_saddrl    .word  00000000h  ;source address, low

com_daddrh    .word  00000000h  ;destination address, high
com_daddrl    .word  00000000h  ;destination address, low
```

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 sicinit.asm.

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.

Types of Data Transfers

Four types of data transfers are supported by the communications protocol: 16 bit, 32 bit, 16 bit DMA, and 32 bit DMA. In each case, data is transferred to and from sequential locations in TMS320C3x memory.

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.

DMA Commands

DMA transfers are well suited for SIC3xDSP applications because data is generally 16 bits or less, thereby allowing the concatenation of data into a single 32 bit word. 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. 16 bit DMA data may require special processing before a mathematical computation is performed.

Command Execution Examples

Example 3-1 through Example 3-6 show the command execution sequence for each of the commands on the TMS320C3x side of the communications interface.

Example 3-1. 16-Bit Writes - hwrite16
This example shows the sequence for transferring data from the communications register to the TMS320C3x 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_addrl.

Example 3-2. 16-Bit Reads - hread16
This example shows the sequence for transferring data from TMS320C3x 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 TMS320C3x 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 TMS320C3x 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 example shows the sequence for transferring data from the communications register to the TMS320C3x 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 the 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 TMS320C3x 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_daddrl.
Step 2: Set the DMA destination address register to @hostport.
Step 3: Set the DMA count to @com_countl.
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.

ADDENDUM

Overview

In the enhanced SI-C3xDSP-ISA cards from Sheldon Instruments, there are two distinct DSP communication protocols: discrete Interrupt based transfers and DMA based transfers. However, both of these DSP protocols are implemented with programmed I/O on the host side, without the use of DMA or IRQ resources.

Discrete Interrupt transfers usually take place during transfers of function commands, or during transfers involving small amounts of data, usually less than ten 32bit data words, or transfers of data that is not mapped contiguously in the DSP's memory. A discrete DSP Interrupt must be invoked every time the host completes a transfer, which helps ensure a reliable communications link. However, the overhead may be substantial when transfers require more than ten 32 bit words. All communications are initiated by the host, which sets the C3xRDY bit to a "HI", and will remain "HI" until the DSP responds by clearing this bit. The host must pole this bit before proceeding with the next transfer.

DMA based transfers are performed with the DSP's DMA controller. First, the DMA controller is configured with discrete Interrupt transfers, but the actual transfer of data is performed by the DSP's DMA controller. DMA based transfers are useful for transfers of at least more than ten 32 bit words. The host must perform the first transfer in order to invoke a DSP DMA transfer cycle, and may continue to access both the high and lower parts of the DSP's 32 bit word without any software polling. Before a DMA transfer is to take place however, the host must enable the hardware wait signal on the ISA bus (IORDY) in order to avoid software polling.

Since the ISA bus is 16 bits wide, and the DSP has a 32 bit wide interface, two ISA bus cycles must take place. The host must always access the upper half of the DSP's 32 bit word (Base + 0x10 = HI WORD), and then access the lower half of the DSP's 32 bit word (Base + 0x08 = LO WORD). Notice that synchronization with the DSP is implemented by interrupting the DSP every time the host performs an access to the LO WORD register. The host is synchronized by poling the C3xRDY bit.

When using discrete Interrupt transfer, the DSP is interrupted on every host access to LO WORD. When this interrupt occurs, the DSP performs one 32 bit read/write operation. In order to accommodate the 32 bit access, the host first accesses the HI WORD, then the LO WORD. This ensures that the DSP gets the full 32 bits of information when the interrupt occurs upon the host's access to the LO WORD.

The mapping for the board can be jumper configured to support four distinct I/O mapped base address locations. All registers are 16 bits wide.

I/O Mapping Base Address

MAPSEL1	MAPSEL0	Base I/O range

C	A	0x240 - 0x25F
C	B	0x280 - 0x29F
D	A	0x320 - 0x33F
D	B	0x340 - 0x35F

Register Summary

Address	Name	Description	Width (bits)	Direction
Base+0x00	CNTRLWD	DSP Function Control (causes DSP INT0)	8	R/W
Base+0x04	CSR	Board Logic Control/Status Register	8	R/W
Base+0x08	LO_WORD	Lower 16 bit DSP Data (causes DSP INT1 for Host writes and DSP INT2 for host reads)	16	R/W
Base+0x10	HI_WORD	Upper 16 bit DSP Data	16	R/W
Base+0x18	RESET	Reset Line	8	W

Register Details

1) Base + 0x00 = **CNTRLWD** register.
DSP Function Control; 8 bit wide, R/W. Host accesses to this I/O address causes DSP INT0.

2) Base + 0x04 = **CSR** register.
Board Logic CONTROL/STATUS register; 8 bit wide. CSR:CONTROL register during a host write, but CSR:STATUS register during a host read. No DSP interrupts are generated by host accesses to this I/O address. However, host writes to this register controls the onboard logic where writing certain bits are used to directly RESET the DSP, put the DSP into microcomputer/microprocessor mode (for COFF file loading), or control DSP interrupt handling during initialization and during the command phase of a data transaction. On the other hand, host reads to this register provides a method of software handshaking to ensure proper synchronization during data transfers between the host and the DSP.

CSR:CONTROL Register Write only bits at Base + 0x04:

Bit [7:5] = unused.

Bit [4:3] = TMC[1:0]; Transfer Mode Configuration bits.

00= NULL: Normal operation, set during the data phase of a communications cycle between the DSP and host. Allows DSP INT1 and INT2 during host accesses to the LO WORD.

01= CLEAR_EN: Enables the Clearing of the control logic that arbitrates the

communications. By clearing the communications logic, the arbitration is set to a known initial state so as to avoid any false device status signals before a transfer is to take place.

NOTE: The CLEAR_EN does not place the DSP into a reset condition. The DSP is placed into a reset condition by toggling CSR_RES bit instead, as described below.

10=CMD_EN: Enable Command Mode, active during the final transaction in the command phase of a communications transfer between the host and DSP. Before any communication is to take place, the host must initiate the command phase before the DSP can participate in the data phase portion of the transfer. This command phase has a unique communications protocol in the final transaction, which must be clearly distinguished by the communications logic in order to avoid any false device status signals. Specifically, the DSP must not be interrupted by INT0, thus allowing it to proceed parsing the command parameters without misinterpreting this last host access as a CNTRLWD access. Host accesses to CNTRLWD do **NOT** cause DSP interrupt in this mode.

11=IORDY_EN: Enable IORDY line on the ISA bus. If software polling is not used to synchronize transfers, the ISA bus needs to be held in cases where the DSP has not completed a transfer before the ISA bus proceeds with the next transfer cycle. Normally, the ISA bus signal IORDY is held "HI" to force hardware wait states; when "LO", the ISA bus operates with no wait states. This mode is only available during the data phase of a communications cycle and is a substitute for software polling of the CSR:STATUS register.

Bit 2 = BLSRC: Boot loader Interrupt source. This signal must be toggled LO-HI-LO when the DSP is in microcomputer mode. Toggling this line activates the DSP's on-chip Boot Loader used to load a COFF file to DSP memory. Please refer to the C3x reference manual for more details.

1= To be toggled HI for a single cycle only. Causes an Interrupt to the DSP in microcomputer mode only, thus activating the on-chip Boot Loader.

0= To be left LO while DSP is in processor mode for normal operation.

Bit 1 = MC/BLMP: DSP operating mode. Please refer to the C3x reference manual for more details.

1= C3x is in Microcomputer mode, which activates the DSP on-chip boot loader for COFF file loading.

0= Sets the DSP into Processor mode for normal operation.

Bit 0 = CSR_RES: Reset C3x DSP.

1= Reset C3x DSP.

0= Remove C3x DSP reset for normal operation.

NOTE: The CSR_RES only resets the DSP; not the communications logic. Initializing of the control logic is done by setting the TMC[1:0] bits to the CLEAR_EN mode described above.

CSR:STATUS Register Read only bits at Base + 0x04:

Bit [7:1] = unused, always "0".

Bit 0 = C3xRDY: Status bit to indicate if the DSP is ready for another transfer. This line is set HI by a host access to the LO WORD (triggering either a DSP INT1 or INT2), and then cleared to a LO by a subsequent DSP response.

1= C3x is busy.

0= C3x is ready for another transfer.

3) Base + 0x08 = **LO_WORD** register.

Lower 16 bit data word, R/W. During data phase, the DSP is interrupted every time this register is accessed by the host. Additionally, all host accesses to the LO WORD register set the CSR Status bit 0 to a HI, which indicates to the host that the DSP is busy, where the host must wait until it is cleared to a LO by a DSP response.

4) Base + 0x10 = **HI_WORD** register.

High 16 bit data word, R/W. No DSP interrupt is caused by host accesses. In order for the host to complete a 32 bit transfer, it must access this register before the LO WORD in order to avoid data being overwritten by a DSP interrupt.

5) Base + 0x18 = **RESET** register.

8 bit reset byte invoked exclusively by the TBC, whose functionality is also duplicated in bit 0 of the CSR: CONTROL Register located at Base + 0x04. Not to be used if the TBC is absent.

Examples of Host Accesses

To access the DSP, the communication cycle consists of two phases: a) command phase, and b) data phase. During the command phase, the communications parameters must first be established using the discrete steps, which includes the host writing each parameter separately, thereby invoking DSP interrupts on every parameter word transferred. After every host parameter write, the host must poll the CSR:STATUS register in order to ensure proper synchronization. Note that polling is simply the host checking to see if the DSP has responded by clearing the C3xRDY bit in the CSR:STATUS register.

Example of Host to DSP Transfer (Host Write)

1. Clear the communication link between the host and the DSP, by writing TMC[1:0]:01= CLEAR_EN mode in the CSR:CONTROL register. This is just a command to clear the onboard arbitration logic, place the hardware into a known state, and does not affect the DSP in any way.

2. Return to normal operating mode, by writing TMC[1:0]:00= NULL mode in the CSR: CONTROL register. This command does not affect the DSP in any way.

3. Alert the DSP that the command phase of a communication cycle is to **begin**, by writing the BEGIN_CMD_SEND/BEGIN_CMD_SEND_32 value (0x1/0x4) to the CNTRLWD

register. This causes a DSP INT0 to occur.
NOTE: Check if the DSP is ready (C3xRDY bit) after sending this message.

4. Send the first half of the Command Type to the DSP, by writing the upper 16 bits of the 32 bit Command Type to the HI_WORD register. For the SI example program, the command types for host to DSP transfers (host writes) are i) CMD_HOST_MW16 (d12/0xC), ii) CMD_HOST_MW32 (d14/0xE), iii) CMD_HOST_DMAW (d16/0x10), iv) CMD_HOST_DMAW32 (d16/0x10), and v) CMD_HOST_MW32W (d32/0x20), as shown in the table below. Therefore, the value '0x0000' should always be written to the HI_WORD register. Note that this access does **NOT** cause any interrupts to the DSP, and is only intended to clear any residue that may be present in the upper 16 bit registers when the DSP reads the command type after the next step.

5. Send second half of the Command Type to the DSP, by writing the lower 16 bits of 32 bit Command Type to the LO_WORD register. For the SI example program, the actual values should be written, depending on what type of access is desired. Note that this causes a DSP INT1 to occur after placing the data into the lower 16 bit registers, thereby alerting the DSP of their presence. The DSP always performs a 32 bit access, and will use the 32 bit Command Type value it just read to be interpreted in its ISR1.

NOTE: Since the DSP was interrupted, check for DSP ready (C3xRDY bit) after sending this word.

The following values are valid types:

Command Type	Value	Description
CMD_HOST_MR16	11* (0xB)	16/32 bit, DSP to host transfer (host read), S/W or H/W polled, 16/32 bit wide registers
CMD_HOST_MW16	12 (0xC)	16 bit, host to DSP transfer (host write), S/W or H/W polled, 16/32 bit wide registers
CMD_HOST_MR32	13 (0xD)	32 bit, DSP to host transfer (host read), polled mode, 16 bit wide registers
CMD_HOST_MW32	14 (0xE)	32 bit, host to DSP transfer (host write), polled mode, 16 bit wide registers
CMD_HOST_DMAR	15* (0xF)	16/32 bit, 'DSP DMA' to host transfer (host read), DSP DMA enabled, S/W or H/W polled, 16/32 bit wide registers
CMD_HOST_DMAW	16* (0x10)	16/32 bit, host to 'DSP DMA' transfer (host write), DSP DMA enabled, S/W or H/W polled, 16/32 bit wide registers

CMD_HOST_DMAR32	15 (0xF)	32 bit, 'DSP DMA' to host transfer (host read), DSP DMA enabled, S/W or H/W polled, 32 bit wide registers
CMD_HOST_DMAW32	16 (0x10)	32 bit, host to 'DSP DMA' transfer (host write), DSP DMA enabled, S/W or H/W polled, 32 bit wide registers
CMD_HOST_ MR32W	11* (0xB)	32 bit, DSP to host transfer (host read), S/W or H/W polled, 32 bit wide registers
CMD_HOST_MW32W	32* (0x20)	32 bit, host to DSP transfer (host write), S/W or H/W polled, 32 bit wide registers

6. Send the first half of the Count value or quantity of 32 bit words (Dwords) to be transferred to the DSP, by writing the upper 16 bits of the Count to the HI_WORD register. This is equivalent to the total number of DSP accesses to take place after all command parameters are sent. The DSP is not affected in any way during this step.

7. Send the second half of the Count value or quantity of 32 bit words (Dwords) to be transferred to the DSP, by writing the lower 16 bits of the Count to the LO_WORD register. ***NOTE: Since the DSP was interrupted, check for DSP ready (C3xRDY bit) after sending this word.***

8. Send the first half of the DSP Source Address to the DSP, by writing the upper 16 bits of the DSP Source Address to the HI_WORD register.
NOTE: The DSP Source Address is only useful for DSP to host transfers (host reads). The entire 32 bits of the DSP Source Address is ignored by the DSP for host to DSP transfers (host write).

9. Send the second half of the DSP Source Address to the DSP, by writing the lower 16 bits of the DSP Source Address to the LO_WORD register.
NOTE: Since the DSP was interrupted, check for DSP ready (C3xRDY) after sending this word.

10. Send the first half of the DSP Destination Address to the DSP, by writing the upper 16 bits of the DSP Source Address to the HI_WORD register.
NOTE: The DSP Destination Address is only useful for host to DSP transfers (host writes). The entire 32 bits of the DSP Destination Address is ignored by the DSP for DSP to host accesses (host read).

11. Send the second half of the DSP Destination Address to the DSP, by writing the lower 16 bits of the DSP Destination Address to the LO_WORD register.
NOTE: Since the DSP was interrupted, check for DSP ready (C3xRDY) after sending this word.

12. Alert the DSP that the command phase of a communication cycle is to **end**, by writing

the END_CMD_SEND/END_CMD_SEND_32 value (0x2/0x8) to the CNTRLWD register. This causes a DSP INT0 to occur, after which all parameters read by the DSP are to take effect.
NOTE: Check if the DSP is ready (C3xRDY) after sending this message.

13. Temporarily disable DSP interrupt INT0, by writing TMC[1:0]:10= CMD_EN mode in the CSR:CONTROL register. This command disables interrupt to the DSP while the host accesses the CNTRLWD register in order to verify that the DSP has properly processed all parameters, as defined in the next step. This command does not affect the DSP in any way.

14. Verify that the DSP has properly processed the command parameters, by reading the CNTRLWD register at Base+0x0. The DSP should return a 0x0 to this register if the parameters have been properly processed. If this value is anything other than 0x0, the DSP has detected an error in reading the command parameters.

15. Clear the communication link between the host and the DSP, by writing TMC[1:0]:01= CLEAR_EN mode in the CSR:CONTROL register. This is just a command to clear the onboard arbitration logic, place the hardware into a known state, and does not affect the DSP in any way. At this point, the entire command phase of a communication cycle has been completed.

16. Before proceeding with the data phase of the communication cycle, the synchronization mode must be first configured so as to ensure correct data transfers between the host and the DSP. The synchronization mode may be performed either in software or in hardware, by writing one of two commands into the TMC[1:0] bits of the CSR:CONTROL register. Writing TMC[1:0]: 00= NULL enables software synchronization, in which the host MUST poll the CSR:STATUS register after every single data access so as to wait for the DSP to respond, before the host may proceed to the next data access. Writing TMC[1:0]:11= IORDY_EN enables hardware synchronization, in which the host is automatically held by ISA bus hardware logic if it tries to proceed to the next data transfer before the DSP has had a chance to respond. Hardware synchronization is generally more desirable than software polling as their is no software overhead.

17: Begin the actual data phase by sending the upper half of the first data word to the DSP, by writing the upper 16 bits of the data to the HI_WORD register. This step does NOT cause DSP interrupts.
NOTE: This step is to be ignored if the host to DSP transfers (host writes) is only 16 bits.

18. Send the lower half of the first data word to the DSP, by writing the lower 16 bits of the data to the LO_WORD register. This step causes a DSP interrupt and serves to alert the DSP that the first data word is ready to be accessed.
NOTE: Since the DSP was interrupted, check for DSP ready (C3xRDY) after sending this word, before proceeding to the next data transfer.

19: Repeat steps 17 and 18 for the number of words (count) specified in DSP command

parameter.

Summary for host to DSP transfers (host writes):

1. Perform the command phase by sending the command parameters to the DSP.
2. For 32 bit host writes, write upper 16 bits of data to HI_WORD. Ignore this step when performing 16 bit transfers.
3. Write lower 16 bits of data to LO_WORD. Note that a DSP INT1 is caused and the C3xRDY bit must be checked.
4. Repeat steps 2 and 3 for the number of words specified in DSP command parameter.

Example of a DSP to Host Transfer (Host Read)

The number of steps in the command phase are identical to those outlined above, except the command types for DSP to host transfers (host reads) are i) CMD_HOST_MR16 (d11/0xB), ii) CMD_HOST_MR32 (d13/0xD), iii) CMD_HOST_DMAR (d15/0xF), iv) CMD_HOST_DMAR32 (d15/0xF), and v) CMD_HOST_MR32W (d11/0xB), as shown in the table above.

NOTE: Before the data phase is to begin, a "dumby" or additional host read of the LO_WORD register is required to generate a DSP interrupt, which alerts the DSP to initiate the first data transfer. Note that a host access must always precede a DSP response, since it is the host access to the LO_WORD register that causes a DSP interrupt. Secondly, for the unique case of DSP DMA to host transfers (host reads), the transfer Count (in the command phase) must be incremented by a single value in order to account for this host dumby access.

Summary for DSP to host transfers (host reads):

1. Perform the command phase by sending the command parameters to the DSP. Remember to increment the Count by one (1) when using the DSP's DMA to perform the transfer.
2. Host must perform a "dumby" or additional read to the LO_WORD register, in order to cause a DSP interrupt. Note that a DSP INT2 is caused and the C3xRDY bit must be checked.
3. For 32 bit host reads, read upper 16 bits of data from HI_WORD. Ignore this step when performing 16 bit transfers.
4. Read lower 16 bits of data from LO_WORD. Note that a DSP INT2 is caused and the C3xRDY bit must be checked.
5. Repeat steps 3 and 4 for the number of words specified in DSP command parameter.

DSP Reset Function

There are two methods by which the DSP can be placed into a REST condition:

1) If TBC is present, the RESET register located at Base+0x18 will be available to place the DSP

in a REST condition. If the TBC is absent, this register is ignored.
2) The DSP can always be placed into a reset condition by accessing Bit 0 of the CSR:CONTROL register as indicated above.
NOTE: When the DSP has been placed into a reset condition, be sure to clear the onboard logic by writing a TMC[1:0]:01= CLEAR_EN to the CSR:CONTROL register as described above.

Writing DSP code to Communicate with the Host.

The DSP program must be written to handle all conditions for communicating with the host. This requires reset, DSP INT0, DSP INT1, DSP INT2, and DSP DMA interrupt. On each interrupt, the DSP should examine the data on address 0xFF0000; which is the DSP's address where the onboard registers are mapped.

The reset portion of the code should be written to set all interrupt vectors to appropriate values, and then and enable the interrupts.

ISR0 should be written to take the command parameter from the host. This is triggered by a host write to CNTLWD. If BEGIN_CMD_SEND/BEGIN_CMD_SEND32 is received, the DSP ISR should prepare the interrupt service routines for the WRITE function (ISR1) to receive four 32 bits of data: 1) Command Type, 2) Count, 3) Source Address, and 4) Destination Address.

When END_CMD_SEND/END_CMD_SEND32 is received, the ISR should examine the command parameters and prepare the ISR1, ISR2, and DMA ISR according to the received parameters.

ISR1 should take the data received on the Host Port and copy it to another DSP memory location specified by the Destination Address field. It should increment the Destination Address so as to have the next interrupt write to the new destination address.

ISR2 should take the data in the Source Address field and copy it to the Host Port. It should increment the source address so the next interrupt will read from the new source address.

The DSP's DMA interrupt should be set to occur once all the data transaction is complete with DMA. The ISR should disable any existing DMA transfers and disable the DMA controller. It should reset the Interrupt Enable field so as to allow a new command phase of a new command cycle to take place.