Communication
Automation
Corporation

1180 McDermott Dr ♦ West Chester, PA 19380-4022
Tel: (610) 692-9526 ♦Toll-free: (800) 367-6735 ♦ Fax: (610) 436-8258 ♦http://www.cacdsp.com♦ Email: sales@cacdsp.com

EMC12

Audio Interface for the EmPack System

Mezzanine Board Technical Reference Version 1.4

email: support@cacdsp.com

License Agreement

Warranty

Trademarks

Table of Contents

TABLES

FIGURES

1. EMC12 Audio Interface for the EmPack System

1.1 EmPack EMC12 Overview

1.1.1 EMC12 Introduction

This document specifies the characteristics of the EMC12, an EmPack mezzanine board with six CS4231A CODECs with a total of 12 audio channels, an optional Agere DSP32C processor, its memory, and the software libraries and drivers to support this board. Some of the detailed specifications in this manual reflect logic available in version 1.2 of the EMC12 FPGA configuration. This FPGA configuration is included in version 3.4.7 of the EmPack FPGA flash file (fpga347.bin).

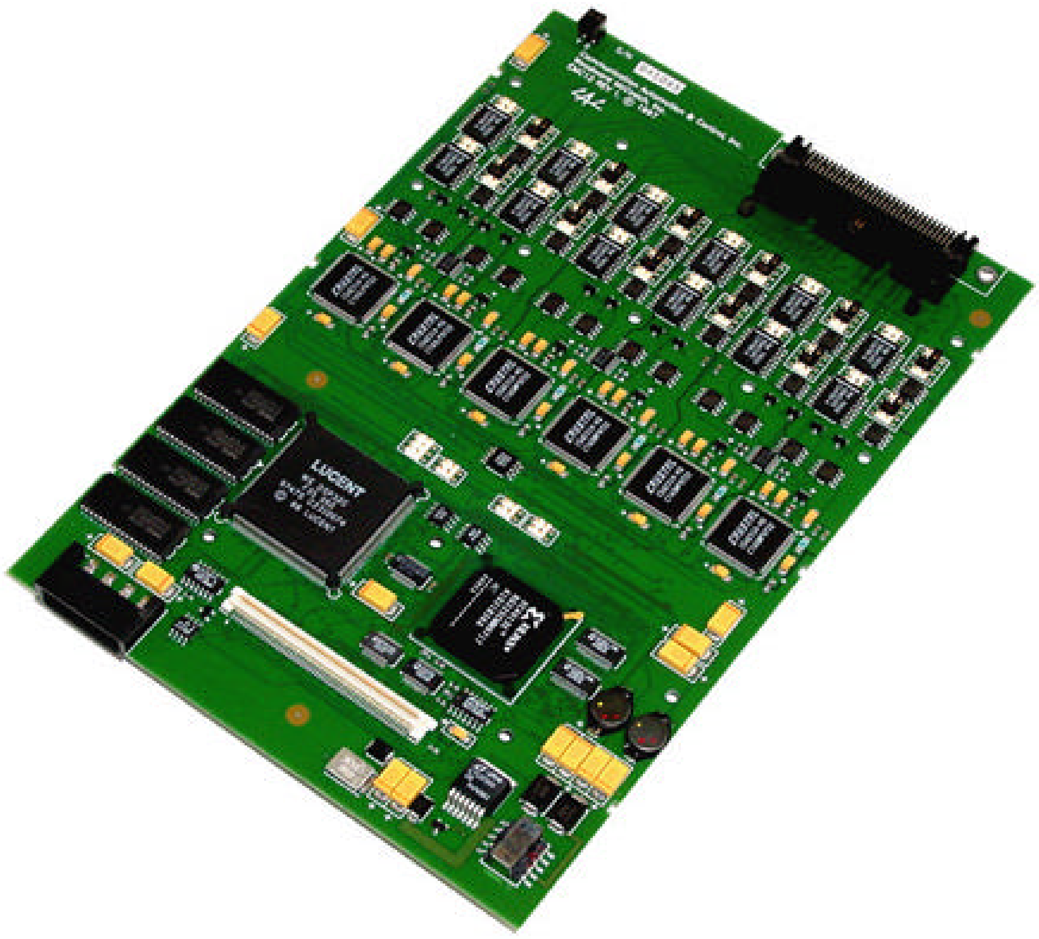


Figure 1-1: EMC12 Mezzanine Board with DSP

1.1.2 EMC12 Top Level Description

The EMC12 is a mezzanine board for the EmPack system. It contains the following major functional blocks:

An EmPack mezzanine bus interface
Six CS4231A CODECs with a total of 12 audio channels
An optional Agere DSP32C processor operating at 74 MHz
512k bytes or 2 Mbytes of zero wait-state (0ws) static RAM for the DSP
6 TDM buses connecting serial ports of the DSPs and other resources in the EmPack system
2 FIFOs storing TDM control information
2 LEDs under DSP program control
A global programmable frequency generator
An EEPROM containing board specific information
A temperature sensor
Software drivers for Solaris, Windows NT and Win2K are provided, together with a software interface library to allow access to and control of the EMC12 from user-developed programs.
Diagnostics are provided which are sufficient to verify correct operation of all functional blocks on the EMC12.

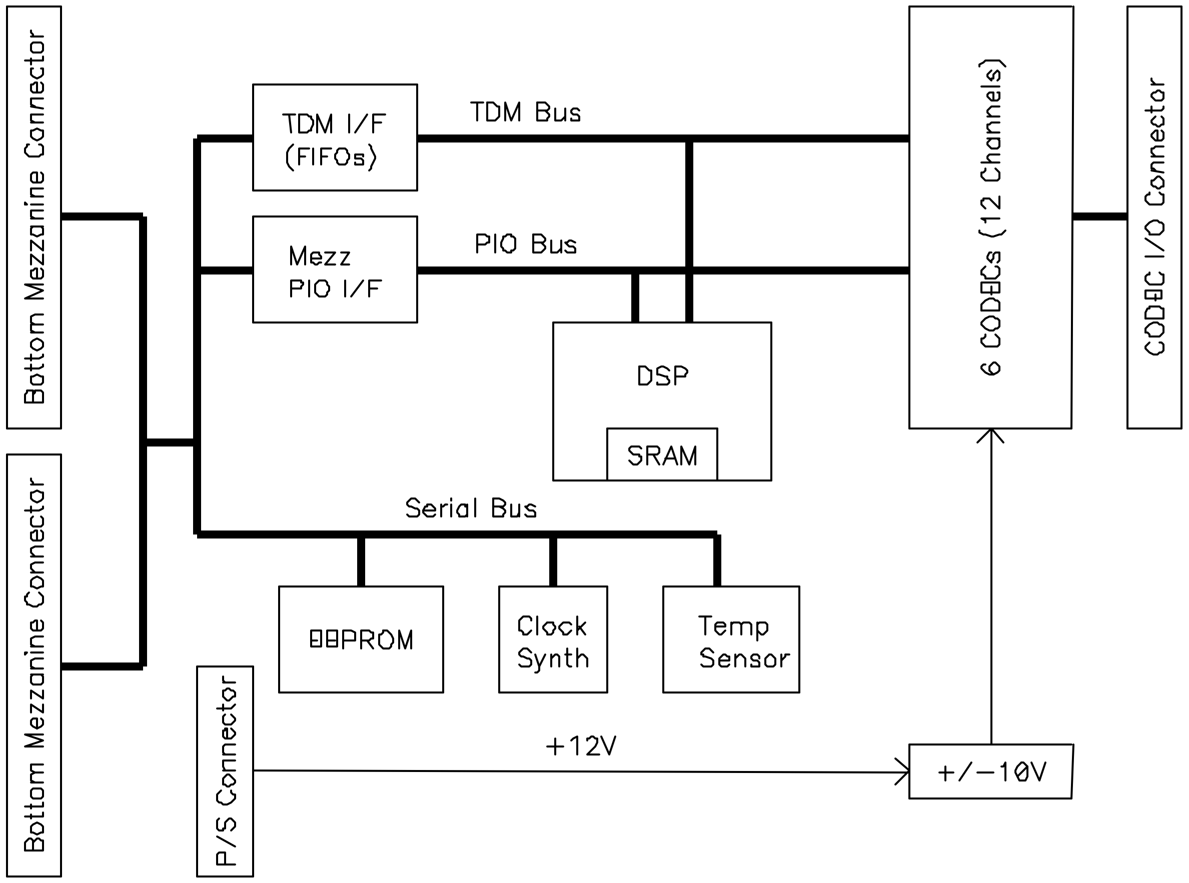


Figure 1-2: EMC12 Block Diagram

1.2 EMC12 Hardware Specification

The electrical hardware portions of the characteristics in Section 3 are elaborated in the following subsections.

1.2.1 EmPack Mezzanine Interface

The EmPack mezzanine connector provides the basic communication path between the base board and mezzanine boards. Signals used by the EMC12 are grouped by function and discussed in subsequent sections.

Table 1-1: EmPack Mezzanine Connector Pinout

Pin Number	Pin Name	I/O	Pin Number	Pin Name	I/O
1	+12V		2	+12V	
3	+12V		4	+12V	
5	N/C		6	N/C	
7	TDI	I	8	TDO	O
9	TCK	I	10	EECS	I
11	GND		12	GND	
13	MODID0 (bottom) MODID1 (top)	I O	14	MODID1 (bottom) MODID2 (top)	I O
15	MODID3 (bottom) ~MODID0 (top)	I O	16	MODID2 (bottom) MODID1 (top)	I O
17	GND		18	VCC	
19	PIO_D0	I/O	20	PIO_D1	I/O
21	PIO_D2	I/O	22	PIO_D3	I/O
23	GND		24	GND	
25	PIO_D4	I/O	26	PIO_D5	I/O
27	PIO_D6	I/O	28	PIO_D7	I/O
29	GND		30	VCC	
31	PIO_D8	I/O	32	PIO_D9	I/O
33	PIO_D10	I/O	34	PIO_D11	I/O
35	GND		36	GND	
37	PIO_D12	I/O	38	PIO_D13	I/O
39	PIO_D14	I/O	40	PIO_D15	I/O
41	GND		42	VCC	
43	PIO_A0(HBS-)	I	44	PIO_A1	I
45	PIO_A2	I	46	PIO_A3	I
47	GND		48	GND	
49	PIO_A4	I	50	PIO_A5	I
51	PIO_A6	I	52	PIO_A7	I
53	GND		54	VCC	
55	PIO_A8	I	56	PIO_A9	I
57	PIO_A10	I	58	PIO_A11	I
59	GND		60	GND	

Table 1-1 - EmPack Mezzanine Connector Pinout (continued)

Pin Number	Pin Name	I/O	Pin Number	Pin Name	I/O
61	PIO_A12	I	62	PIO_A13	I
63	PIO_A14	I	64	PIO_A15	I
65	GND		66	VCC	
67	PIO_A16	I	68	PIO_A17	I
69	PIO_A18	I	70	PIO_A19	I
71	GND		72	GND	
73	PIO_A20	I	74	PIO_A21	I
75	PIO_LBS-	I	76	PIO_CS-	I
77	GND		78	VCC	
79	PIO_IRQ0-	O	80	PIO_IRQ1-	I
81	PIO_WAIT-	O	82	PIO_ACK(bottom) PIO_ACK (top)	I O
83	GND		84	GND	
85	LTDM0	I/O	86	LTDM1	I/O
87	LTDM2	I/O	88	LTDM3	I/O
89	GND		90	VCC	
91	LTDM4	I/O	92	LTDM5	I/O
93	LTDM6	I/O	94	LTDM7	I/O
95	GND		96	GND	
97	LTDM8	I/O	98	LTDM9	I/O
99	LTDM10	I/O	100	LTDM11	I/O
101	GND		102	VCC	
103	LFSYNC-	I	104	LMSYNC	I
105	PROGRAM-	I	106	RESET-	I
107	GND		108	GND	
109	VCC (bottom) LASTMOD (top)	 O	110	REFCLK	I
111	PIO_RD-	I	112	GND	
113	GND		114	LCLK	I
115	PIO_WR-	I	116	GND	
117	GND		118	PRCLK	I
119	MEZZLCLK	I	120	GND	

1.2.2 Parallel I/O

The primary means of communication between the base board and the mezzanine is through the parallel I/O interface. This group of signals consists of PIO_D[0-15], PIO_A[0-21], PIO_CS-, MODID[0-3], PIO_WAIT-, PIO_RD-, PIO_WR-, PRCLK, PIO_IRQ0-.

1.2.2.1 MODID

Each mezzanine in an EmPack system has a unique Module ID (MODID) at any given time. The MODID signals on the bottom connector of the first mezzanine (closest to the base board) are controlled directly by the base board. The MODIDs of two adjacent mezzanines have the following relationship:

If, for a particular mezzanine, $MODID_n = J_n$,
then, for the mezzanine above it, $MODID_{n+1} = J_{n+1}$,
where J_n is a number in a sequence generated by the 4-bit Johnson counter. The sequence used is {0x7, 0x3, 0x1, 0x0, 0x8, 0xC, 0xE, 0xF}. Notice the sequence is circular. If, for example, the first mezzanine sees a MODID of 0xE, then the second and third mezzanine will see 0xF, and 0x7 respectively. For normal PIO, the base board always drives 0x7 on the MODID bits.

1.2.2.2 PIO Address Map

A mezzanine is selected in a PIO read/write operation if PIO_CS- is low, and the MODID matches the address bits PIO_A[18-21]. Address bits PIO_A[14-17] select the resource on the mezzanine, and PIO_A[1-5] select the register to be read or written. All PIO are 16-bit wide and PIO_A[0] is not decoded.. Table 1-2 below shows the mezzanine PIO resource map.

See the CS4231A datasheet for register mapping of the CODEC and BROADCAST resources. Note: Access to the BROADCAST resource is read only.

Table 1-2: PIO Resource Map

PIO_A[14-17]	Resource
0x0	CODEC 0
0x1	CODEC 1
0x2	CODEC 2
0x3	CODEC 3
0x4	CODEC 4
0x5	CODEC 5
0x6	BROADCAST
0x7	FIFO (TDM)
0x8	SYSTEM
0x9	DSP
0xa - 0xF	Reserved

Table 1-3: Register Map For DSP Resources

PIO_A[1-5]	Access	Register	Location
0x00 - 0x0D	Read/Write	See AT&T DSP32C Information Manual	DSP
0x0E - 0x11		Reserved	
0x12	Read/Write	PDF-conditional PDR access (PDRW)	FPGA
0x13 - 0x1D		Reserved	
0x1E	Read/Write	PDF-conditional PDR2 access (PDR2W)	FPGA
0x1F		Reserved	

Table 1-4: Register Map For TDM Resource

PIO_A[1-5]	Access	Register	Location
0x00	Read/Write	TDM Control Data Register (TCDR)	FIFO
0x01 - 0x1F		Reserved	

Table 1-5: Register Map For SYSTEM Resource

PIO_A[1-5]	Access	Register	Device
0x00	Read/Write	Board Control Register 0 (BCR0)	FPGA
0x01	Read/Write	Board Control Register 1 (BCR1)	FPGA
0x02	Read	PIO Interrupt Status (PISR)	FPGA
0x03	Read/Write	PIO Interrupt Mask (PIMR)	FPGA
0x04	Write	DSP Interrupt Control (DICR)	FPGA
0x05	Read/Write	Last TDM Control Word (LTCW)	FPGA
0x06	Read/Write	CODEC test points select (bits 0 – 2)	FPGA
0x07	Read/Write	Test point diagnostic signal (bit 0)	
0x08	Read/Write	CODEC 0 Control Status (CCS0)	FPGA
0x09	Read/Write	CODEC 1 Control Status (CCS1)	FPGA
0x0a	Read/Write	CODEC 2 Control Status (CCS2)	FPGA
0x0b	Read/Write	CODEC 3 Control Status (CCS3)	FPGA
0x0c	Read/Write	CODEC 4 Control Status (CCS4)	FPGA
0x0d	Read/Write	CODEC 5 Control Status (CCS5)	FPGA
0x0e - 0x1F		Reserved	

Table 1-6: Board Control Register 0 (BCR0)

Bit(s)	Name	Access	Function
0	DSP_RESET	Read/Write	Control RESTN (active low) pin of each DSP
1	DSP_MMODE	Read/Write	DSP memory mode: 0 = mode 6 1 = mode 7
2	TDM_RESET	Read/Write	TDM reset
3	Reserved		
4	ACTIVE_FIFO	Read Only	Currently active FIFO
5 - 10	CODEC_RESET	Read/Write	Assert Power-Down to Codecs 0 - 5
11 - 14	Reserved		
15	DIGI_LOOP	Read/Write	Experimental Digital Loop-back mode

Table 1-7: Board Control Register 1 (BCR1)

Bit(s)	Name	Access	Function
0	AV9110_EN	Read/Write	AV9110 Chip Select: 0 = chip deselected 1 = chip selected
1	DS1620_EN	Read/Write	0 = chip deselected 1 = chip selected
2	FIFO_RESET	Write Only	Reset inactive FIFO

3	FIFO_RT	Write Only	Retransmit inactive FIFO
4	SWITCH_FIFO	Write Only	Switch active FIFO
5	CLR_FINT	Write Only	Clear FIFO Switch Interrupt
6	CLR_TINT	Write Only	Clear PIO Timeout Interrupt
7 - 15	Reserved		

Table 1-8: PIO Interrupt Status Register (PISR)

Bit(s)	Name	Access	Function
0	DSP	Read Only	DSP PIF signal
6	FIFO	Read Only	FIFO switched
7	TIMEOUT	Read Only	PIO Timeout
8 - 15	Reserved		

Table 1-9: PIO Interrupt Mask Register (PIMR)

Bit(s)	Name	Access	Function
0	DSP	Read/Write	0 = disable interrupt 1 = enable interrupt
6	FIFO	Read/Write	0 = disable interrupt 1 = enable interrupt
7	TIMEOUT	Read/Write	0 = disable interrupt 1 = enable interrupt
8 - 15	Reserved		

Table 1-10: DSP Interrupt Control Register (DICR)

Bit(s)	Name	Access	Function
0 - 2	INT_CTRL	Write Only	0 = Reserved 1 = set External INTREQ1 2 = set External INTREQ2 3 = clear INTREQ1 and INTREQ2, disable LMSYNC interrupt 4 = enable setting INTREQ1 by the rising edge of LMSYNC 5 = disable LMSYNC interrupt 6= enable setting INTREQ1 by both edges of LMSYNC 7 = Reserved
3 - 15	Reserved		

Table 1-11: CODEC Control Status Register (CCSR)

Bit(s)	Name	Access	Function
0	ADC16	Read/Write	ADC 16-bit conversion
1	DAC16	Read/Write	DAC 16-bit conversion
2	SR16	Read/Write	16kHz sample rate
3	SR32	Read/Write	32kHz sample rate

4	SR48	Read/Write	48kHz sample rate
5	LOCK	Read Only	CODEC sync locked
6	LOSTLOCK	Read Only	CODEC sync lost lock
7 - 15	Reserved		

1.2.2.3 PIO Read/Write

The PIO_RD-, PIO_WR- and address decoding are used by the FPGA to generate appropriate read, write and enable signals to the DSPs, the FIFOs and internal registers. As soon as the board is selected (PIO_CS- low and MODID matches PIO_A[18-21]), PIO_WAIT- is asserted. When the PIO access is decoded and no additional wait states are needed, the PIO_WAIT- signal is de-asserted, and the appropriate read/write pulses are generated.

If the PDRW or PDR2W is addressed, then the PDF signal from the corresponding DSP is used to generate extra wait states, if necessary. A high PDF in a write cycle or a low PDF in a read cycle will insert extra wait states until the condition goes away. A maximum of 15 wait states are allowed before the state machine terminates the read/write cycle by releasing the PIO_WAIT- signal, and set the TIMEOUT bit in the PISR. If the corresponding bit in the PIMR is also set, the PIO_IRQ0- is asserted (driven low).

1.2.2.4 PIO Interrupt

The mezzanine board can interrupt the base board via the PIO_IRQ0- signal. There are 3 interrupt sources on the mezzanine each of which has a corresponding bit in the PIMR and PISR. Bits 0 of PISR reflect the signal levels of the DSP PIF pins. Bit 6 of PISR is set when a FIFO switch has occurred. Bit 7 is set by a PIO timeout.

An interrupt source is enabled when the corresponding bit in the PIMR is set and disabled if the bit is reset. The contents of PISR and PIMR are bit-wise AND'ed, and the result is OR'ed to generate the PIO_IRQ0- signal. The PIO_IRQ0- is asserted until all the interrupt sources are either turned off or disabled. Reading the PIR clears the corresponding DSP interrupt. To clear the FIFO Switch and PIO Timeout interrupts, set the CLR_FINT and CLR_TINT bits in the BCR1 respectively.

1.2.3 Reset and Configuration

The EEPROM (NM93C46M8) contains board specific information for the mezzanine. The TCK and TDO signals are connected to the SK and DO pins of the EEPROM. MODID0 and MODID3 are AND'ed together and connected to the DI pin. The base board can use the information stored in the EEPROM to configure the EMC12 appropriately.

Note that the sequence of numbers used for MODID is {0x7, 0x3, 0x1, 0x0, 0x8, 0xC, 0xE, 0xF}. 0xF is the only number whose MODID0 and MODID3 bits AND'ed together gives '1'. This means to shift a '1' into the DI pin of the EEPROM, the base board needs to drive a value to MODID such that the targeted mezzanine sees 0xF as the MODID. Any other value will present a '0' at the DI pin. For example, if the base board wants to interrogate the EEPROM of the second mezzanine, it needs to send a sequence of '1's and '0's to the EEPROM. To send a '1', it should drive 0xE to MODID, and to send a '0', it can drive 0x7.
Activating the PROGRAM- signal clears the FPGA (XC4013E) internal configuration memory and the FPGA is ready for reprogramming. The TDI, TDO, TCK signals are connected to the respective pins on the FPGA. The MODID0 and MODID3 signals are NAND'ed and connected to the TMS pin of the FPGA. These four pins are used to program the device with the JTAG Configure command. Note the INIT- pin of the FPGA is pulled low to prevent the device from entering normal configuration through the Mode pins.

Since the signals are shared between the FPGA and EEPROM, EECS is used to select the intended device. When EECS is active, the EEPROM is enabled. The EECS signal is also OR'ed with TMS so that when EECS is a '1' the TMS input of the FPGA is also a '1'. This keeps the JTAG in the idle state when the EEPROM is being accessed.
The RESET- signal is connected to the FPGA Global Set/Reset net, and the ZN pin of each DSP. When RESET- is active (low), all DSP outputs are tri-stated and the FPGA internal registers are put in a known state. The DSP_RESET bit in BCR0 becomes '0' which halts all the DSPs on the mezzanine. In addition, the FIFOs are reset and all TDM control information is cleared.

1.3 EMC12 TDM

Serial data passing between resources in an EmPack system are routed through 6 time-division-multiplexed (TDM) buses They are called buses A, B, C, D, E and F. Each contains a data signal and a valid signal. The Table below shows how LTDM lines are allocated among the 6 buses.

Table 1-12: LTDM[0-11] Signal Allocation

TDM BUS	TDM Data	TDM Valid
A	LTDM0	LTDM2
B	LTDM1	LTDM3
C	LTDM4	LTDM6
D	LTDM5	LTDM7
E	LTDM8	LTDM10
F	LTDM9	LTDM11

All 6 buses run synchronously at the same bit clock rate, with identical slot definition, frame duration and frame sync pulse. The buses are connected to the serial port of each DSP. The TDM system operates independently of the parallel I/O system.

Frames of data are passed through each serial TDM bus. The number of slots per frame is fixed at 128. Each time slot has a fixed length of 8 bits. A buffered version of LCLK (SLCK) is used as the TDM bit clock. The LFSYNC- signal is used to indicate frame boundary. It goes low on the falling edge of LCLK for one LCLK period and the frame boundary falls within that pulse.

The TDM subsystem also has the concept of "Superframe" or "Multiframe" timing. The signal, LMSYNC is a high active pulse that lasts for one TDM frame with transitions on a frame boundary. The number of frames per Multiframe is controlled by the foundation board using the API function, **EmSetTdmMode**. The LMSYNC signal is connected to the SY pin of the DSP. A DSP program can test the state of the SY pin (using the **sys** flag to test if LMSYNC is high and the **syc** flag to test if LMSYNC is low). A DSP program can synchronize its serial input and output to the TDM frames by testing for a transition on the LMSYNC signal.

1.3.1 TDM Connections

The interconnection of TDM data to various resources in the system is dynamically controlled by the contents of the TDM map. For each time slot, a set of control words in the map determines the source and destination(s) of the data on each TDM bus. The control words are recycled in every TDM frame.

For the EMC12 mezzanine, eight control words are used to establish connection for each time slot. Tables 1- 13, 1-14, and 1-15, below, show the format of the control words.

The TDM Map control words are controllable using functions in the host API. These are described in section 3.4.8 of this manual.

Table 1-13: TDM Source Control Word Definition (Words 0-2)

Word	Bits 4 - 7	Bits 0 - 3
0	Bus A Source	Bus B Source
1	Bus C Source	Bus D Source
2	Bus E Source	Bus F Source

Table 1-14: TDM Destination Control Words Definition (Words 3-6)

Word	Bits 8 - 6	Bits 5 - 3	Bits 0-2
3	CODEC0 Destination	CODEC1 Destination	CODEC2 Destination
4	CODEC3 Destination	CODEC4 Destination	CODEC5 Destination
5	CODEC6 Destination	CODEC7 Destination	CODEC8 Destination
6	CODEC9 Destination	CODEC10 Destination	CODEC11 Destination

Table 1-15: TDM Destination Control Word Definition (Word 7)

Word	Bits 0 - 3
7	DSP Destination

Tables 1-16, 1-17 and 1-18 below shown the meanings of the values for each type of control word.

Table 1-16: Bus Source Encoding

Value	Meaning
0x0	CODEC0 drives TDM data
0x1	CODEC1 drives TDM data
0x2	CODEC2 drives TDM data
0x3	CODEC3 drives TDM data
0x4	CODEC4 drives TDM data
0x5	CODEC5 drives TDM data
0x6	CODEC6 drives TDM data
0x7	CODEC7 drives TDM data
0x8	CODEC8 drives TDM data
0x9	CODEC9 drives TDM data
0xA	CODEC10 drives TDM data
0xB	CODEC11 drives TDM data
0xC	DSP sends TDM valid control
0xD	DSP sends TDM data
0xE	Continuation slot
0xF	No Connection

Table 1-17: CODEC Destination Encoding

Value	Meaning
0x0	CODEC receives from Bus A
0x1	CODEC receives from Bus B
0x2	CODEC receives from Bus C
0x3	CODEC receives from Bus D
0x4	CODEC receives from Bus E
0x5	CODEC receives from Bus F
0x6	Continuation slot
0x7	No connection

Table 1-18: DSP Destination Encoding

Value	Meaning
0x0	DSP receives unconditionally from Bus A
0x1	DSP receives unconditionally from Bus B
0x2	DSP receives unconditionally from Bus C
0x3	DSP receives unconditionally from Bus D
0x4	DSP receives unconditionally from Bus E
0x5	DSP receives unconditionally from Bus F
0x6 - 0x7	Reserved
0x8	DSP receives conditionally from Bus A
0x9	DSP receives conditionally from Bus B
0xA	DSP receives conditionally from Bus C
0xB	DSP receives conditionally from Bus D
0xC	DSP receives conditionally from Bus E
0xD	DSP receives conditionally from Bus F
0xE	Continuation slot (no ILD)
0xF	No connection

1.3.2 Continuation Slots

Normally a time-slot carries a single, 8-bit data item across each TDM bus. However, the TDM system allows the use of consecutive time-slots on the TDM bus to carry larger data items. Four time-slots, for example, can be combined to carry a 32-bit word. The technique of using multiple time-slots to carry a larger data item is called “slot continuation”. When slot continuation is selected for a TDM bus, the FPGA continues driving DSP serial data onto the TDM bus without generating an OLD pulse. Similarly, the TDM bus data is driven to the DSP serial input without a ILD pulse. The same concept is used for transferring 16-bit data items to or from the Codecs, using pairs of consecutive 8-bit time slots.

1.3.3 TDM Validity and Conditional Transfers

Each TDM bus has a data signal and a valid signal. The valid signal is used to indicate whether a given time-slot contains valid data. When a DSP is selected as the source of a TDM bus for a time-slot, the OBE signal from the DSP is sampled before OLD pulse is generated. If OBE is low, indicating that the DSP's serial buffer has data to send, the OLD pulse is generated and the valid signal is driven high for one bit clock. The valid signal is also driven high for one bit clock when a Codec is selected as the source of a TDM bus. If OBE is high, indicating that the DSP's serial output buffer is empty, the OLD pulse is suppressed, and the valid signal is driven low for 1 bit clock.

When a DSP is selected to conditionally receive from a TDM bus, the valid line of the that bus is sampled before the ILD pulse is generated. If the valid signal is low, the ILD pulse is suppressed and the data on the bus is ignored.

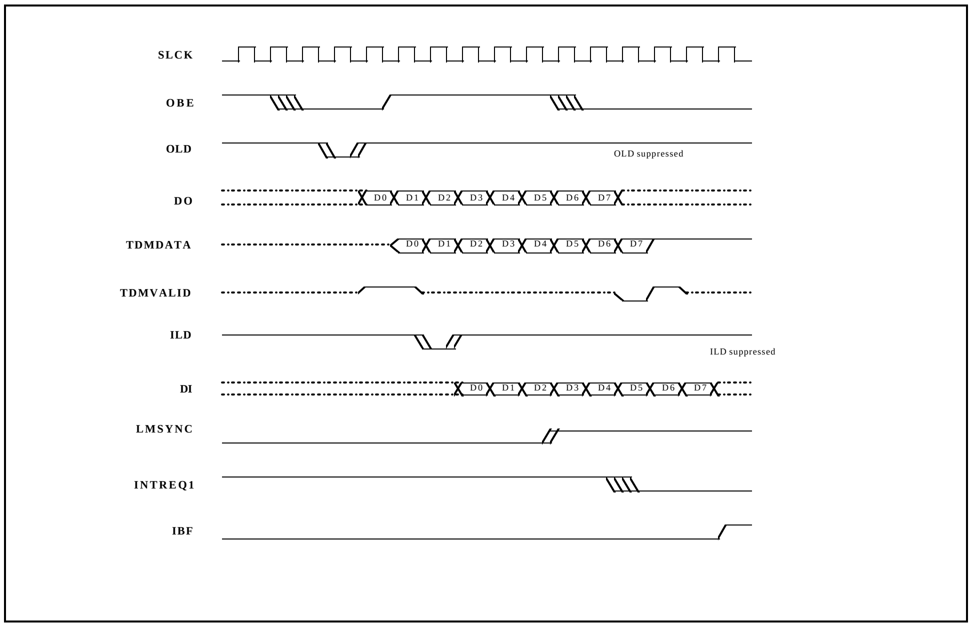


Figure 1-3: TDM Timing

1.3.4 Programmable TDM Validity

For the conditional TDM transfers described above, the hardware between the DSP and the TDM subsystem determines whether a time slot's data is valid based on the state of the DSP's OBE signal. Some applications may require the DSP to determine when the data it has to send is valid or not. For example when dealing with isochronously generated data (such as E1 or T1 framers) the DSP will be sending data at a constant rate and on a fixed and multiple number of time slots per TDM frame. In order to maintain synchronization with the TDM frame, the DSP must be allowed to transmit serial data for every time slot on which it is connected to the TDM bus. However there may be times when the

DSP has no valid data to send on one or more time slots such as when a frame slip is detected on an incoming E1 or T1 stream.

The EMC12 hardware provides a mechanism for the DSP to control the TDM valid signal that accompanies each TDM data bus. This is accomplished by using pairs of consecutive TDM time slots. For each pair of time slots to be connected in this way, the first time slot is given a Source control value of 0xC (DSP Sends TDM Valid Control) and the second time slot is given a Source control value if 0xD (DSP Sends TDM Data). These control word pairs must be encoded for the same TDM bus on consecutive time slots.

When pairs of time slot control words are encoded this way, they behave as though the second time slot was a continuation slot. The DSP receives one OLD signal for the pair, at the beginning of the first time slot. The DSP is then expected to transmit 16 bits of serial data. The first 8 bits contain the TDM valid information and the second 8 bits is the byte of data to be driven as TDM data on the selected TDM bus. The 2nd bit of the valid information is sampled by the hardware and used to control the TDM valid signal for the second time slot of the pair.

The examples below show how to build a 16-bit serial data word in the DSP program for a data byte that is valid and one that is invalid. The bit ordering assumes that the DSP's IOC register is set to transmit the most significant bit of the serial output buffer first and that it will be transmitting serial data from the memory referenced by **serial_out_buffer** using serial DMA output.

```
unsigned short serial_out_buffer[64];
char out_data[64]
int data_valid[64];

for (i = 0; i < 64; ++i)
{
    serial_out_buffer[i] = out_data[i];
    if (data_valid[i])
        serial_out_buffer[i] |= 0x4000;
}
```

1.3.5 TDM FIFO

The TDM control words for an entire TDM frame are stored in one of two FIFOs on the board. The FIFO being used for the current TDM frame is the active FIFO, the other is called the inactive FIFO. The inactive FIFO can be read/written by the base board through the TCDR using PIO read/write operations. Writing a '1' to the FIFO_RT (FIFO Retransmit) bit in BCR1 resets the FIFO read pointer. The word read from the TCDR after a FIFO Retransmit is the first word in the FIFO.

To load a new TDM control map, write a '1' to the FIFO_RESET bit in BCR1. This resets the read/write pointers of the FIFO. Then write each word (9-bit) of the map to the TCDR. A maximum of 1024 words can be written. The 1024th word is written to the LTCW register instead of the FIFO. When the entire map is loaded, write a '1' to the SWITCH_FIFO bit in BCR1. The inactive FIFO will become active at the next frame boundary. The deactivated FIFO can be read and written by the base board after the switch.

This scheme of using two FIFOs and switching at frame boundary ensures that TDM control maps are always valid and consistent.

1.3.6 TDM Last Control Word

Due to hardware limitations, the last TDM control word of the FIFO is not actually used for TDM control. It is replaced on the fly with the contents of the "Last TDM Control Word" register of the FPGA. Therefore, all software that affects the EmC12 TDM map must take this into account and make sure that the "Last TDM Control Word" register is accessed instead of the last FIFO word.

1.3.7 TDM Reset

After power-on or board reset, the TDM_RESET bit in BCR0 is '0'. This bit can also be read/written by the base board using the PIO interface.

When the TDM_RESET bit is '0', the EMC12's TDM system becomes idle. It does not drive any of the LTDM lines or DSP SIO signals. Before writing a '1' to the TDM_RESET bit, the base board should setup the FIFOs with valid TDM control words. The TDM becomes active at the second frame boundary after TDM_RESET become '1'.

1.3.8 API Functions for TDM Control

The Empack host API includes several functions for controlling the TDM timing and connection map. This section describes how these functions pertain to the EMC12 mezzanine. See the Empack API Reference manual for more details about these functions.

1.3.8.1 TDM Timing and Operation Functions

The basic timing is fixed with a bit clock of 8.192 MHz, basic slot size of 8-bits and frame of 128 time slots. The clock source and the number of frames per Multiframe is controlled by the Empack foundation board and using the host API with the **EmSetTdmMode** function:

```
int
EmSetTdmMode (
    EMPACK_HANDLE Empack,
    int Mode,
    int Multiframe,
    int ModuleId
)
```

The ***Mode*** argument specifies the timing reference and clock configuration, specifying the source of TDM clock, whether or not the Empack is supply the clock, sync and 8kHz reference clock to the SCSA expansion bus. The ***Multiframe*** argument specifies the number of TDM frames per Multiframe and the ***ModuleId*** argument specifies which mezzanine is supplying TDM clock if the ***Mode*** argument specifies that the clock is supplied from a Mezzanine.

In case an application is required to adopt the current TDM configuration of an Empack the following function is used to determine the current mode settings and initialize the software state to the current hardware settings:

```
int
EmGetTdmMode (
    EMPACK_HANDLE Empack,
    int *Mode,
    int *Multiframe,
    int *ModuleId
)
```

This function will always initialize the API's internal flags to match the current settings. If desired, the values may be returned to the application by specifying pointers to store the values for ***Mode***, ***Multiframe*** and ***ModuleId***. Specify **NULL** as the pointer for any of the data not required.

The TDM subsystem for a module is enabled or disabled with the following two functions:

```
int
EmStartTdm (
    MODULE_HANDLE Module
)
```

```
int
EmStopTdm (
    MODULE_HANDLE Module
)
```

The TDM subsystem is initially disabled when the Empack first powers up and will also be disabled after a call to the **EmInitModule** or **EmInitEmpack** functions. The **EmStartTdm** function will also updates the hardware TDM map if any changes have been made to the API's software copy since the last time the Map was updated.

An application can determine if the TDM subsystem is currently enabled and if the hardware map is current using the following functions:

```
int
EmIsTdmStarted (
    MODULE_HANDLE Module,
    int *Status
)
```

```
int
EmIsTdmMapCur (
    MODULE_HANDLE Module,
    int *Status
)
```

Both of these functions store their result (0 for no or 1 for yes) in the integer pointed to by the ***Status*** argument.

1.3.8.2 TDM Map Functions

The API maintains a software copy of the TDM map for each module. Incremental modifications to the map are made only to the software copy. The active, hardware copy of the map is updated when specified and becomes active at the next TDM frame boundary. The TDM Map for a module is cleared using the following function:

```
int
EmClearTdmMap (
    MODULE_HANDLE Module
)
```

This function clears both the software and hardware copy of the TDM map. The cleared map becomes active on the next TDM frame boundary. The TDM map is also cleared by the ***EmInitModule*** and ***EmInitEmpack*** functions if they are called with the **EM_INIT_FORCED** option. On power-up the TDM map contents are undefined so it is necessary to call one of these functions to clear the TDM map before creating the desired map.

The following function is used to update the hardware copy of the map after making one or more incremental changed to the map:

> int
> **EmWriteTdmMap** (
>   MODULE_HANDLE ***Module***
>)

This function writes the software copy of the TDM map to the hardware if the hardware copy is not current. The new map will become active on the next TDM frame boundary. Note that the **EmStartTdm** function will also update the hardware map if it is not current.

In case an application is required to adopt the current configuration of the TDM map, the following function is used to initialize the software copy of the map to reflect the current hardware map form the module.

> int
> **EmReadTdmMap** (
>   MODULE_HANDLE ***Module***
>)

Note that in such cases, the application should not call **EmInitModule**, **EmInitEmpack** or **EmClearTdmMap** as doing so would destroy the current configuration of the hardware. After calling **EmReadTdmMap** the application may modify and update the map normally.

The following functions are used to make incremental additions, adding TDM sources or destinations to the software copy of theTDM map.

> int
> **EmAddTdmSrc** (
>   MODULE_HANDLE ***Module***,
>   RESOURCE_HANDLE ***Resource***,
>   int ***Slot***,
>   int ***Bus***,
>   int ***Option1***,
>   int ***Option2***
>)

> int
> **EmAddTdmDst** (
>   MODULE_HANDLE ***Module***,
>   RESOURCE_HANDLE ***Resource***,
>   int ***Slot***,
>   int ***Bus***,
>   int ***Option1***,
>   int ***Option2***
>)

EmAddTdmSrc is used to add a device that will drive data onto a specified TDM bus during a specified TDM time slot. **EmAddTdmDst** is used to add a device to receive data from a specified TDM bus during a specified TDM time slot. Both functions require arguments to specify the Empack ***Module*** and ***Resource*** for the TDM device.

The ***Slot*** argument specifies the TDM time slot with valid values in the range of 0 through 127. The ***Bus*** argument specifies the TDM bus with valid values in the range of 0 through 5 or macros **TDM_BUSA**, **TDM_BUSB**, **TDM_BUSC**, **TDM_BUSD**, **TDM_BUSE** or **TDM_BUSF**.

The ***Option1*** argument specifies additional information for the type of connection to be made. For the DSP resource valid values are:

TDM_DATA	Normal TDM data connection
TDM_CONTINUE	Continuation slot for 16 or 32 bit data
TDM_CONDITIONAL	Conditional receive slot (EmAddTdmDst only)
TDM_DSP_VALID	Valid information slot to precede a data slot (EmAddTdmSrc only)

Valid values for Codec resources are:

TDM_DATA	Normal TDM data connection
TDM_CONTINUE	Continuation slot for 16 bit audio samples

The ***Option2*** argument does not pertain to the EMC12 mezzanine and should be 0.

The following functions are used to make incremental additions, deleting TDM sources or destinations to the software copy of theTDM map.

```
int
EmDelTdmSrc (
      MODULE_HANDLE Module,
      RESOURCE_HANDLE Resource,
      int  Slot,
      int  Bus,
      int  Option1
)
```

```
int
EmDelTdmDst (
      MODULE_HANDLE Module,
      RESOURCE_HANDLE Resource,
      int  Slot,
      int  Option1
)
```

EmDelTdmSrc is used to delete (or disconnect) a device from driving data onto a specified TDM bus during a specified TDM time slot. **EmDelTdmDst** is used to delete (or disconnect) a device from receiving data from a specified TDM bus during a specified TDM time slot. Both functions require arguments to specify the Empack ***Module*** and ***Resource*** for the TDM device.

The ***Slot*** argument specifies the TDM time slot with valid values in the range of 0 through 127. The ***Bus*** argument specifies the TDM bus with valid values in the range of 0 through 5 or macros **TDM_BUSA**, **TDM_BUSB**, **TDM_BUSC**, **TDM_BUSD**, **TDM_BUSE** or **TDM_BUSF**. Note that there is no ***Bus*** argument for **EmDelTdmDst**, it disconnects the device from whatever bus is was receiving data from.

The ***Option1*** argument does not pertain to the EMC12 mezzanine and should be 0.

1.3.8.3 Examples Using TDM_DSP_VALID Mode

The normal TDM map connections using the ***Option1*** modes of TDM_DATA, TDM_CONTINUE and TDM_CONDITIONAL are self explanatory. However the TDM_DSP_VALID and TDM_C12DSP_OLD modes may require some further explanation.

Section 3.4.4 describes how the programmable TDM validity mode, **TDM_DSP_VALID**, works at the hardware level. Below is an example of setting up a TDM map for this mode using the API functions. In the example, a TDM map is built for an EMC12 DSP to transmit data and control the valid line on 4 TDM slots. The DSP will synchronize to the TDM frame and begin sending 4 16-bit words per frame. Each 16-bit word has the TDM valid information in the upper 8-bits and the TDM data is in the lower 8-bits.

```
MODULE_HANDLE    emc12;
RESOURCE_HANDLE  dsp;
int              tdm_bus = TDM_BUSA;

EmAddTdmSrc(emc12, dsp,  4, tdm_bus, TDM_DSP_VALID, 0);
EmAddTdmSrc(emc12, dsp,  5, tdm_bus, TDM_DATA, 0);
EmAddTdmSrc(emc12, dsp,  8, tdm_bus, TDM_DSP_VALID, 0);
EmAddTdmSrc(emc12, dsp,  9, tdm_bus, TDM_DATA, 0);
EmAddTdmSrc(emc12, dsp, 12, tdm_bus, TDM_DSP_VALID, 0);
EmAddTdmSrc(emc12, dsp, 13, tdm_bus, TDM_DATA, 0);
EmAddTdmSrc(emc12, dsp, 16, tdm_bus, TDM_DSP_VALID, 0);
EmAddTdmSrc(emc12, dsp, 17, tdm_bus, TDM_DATA, 0);
EmStartTdm(emc12);
```

With this TDM map loaded, the DSP will receive 4 OLD pulses per frame, at the beginning of time slots 4, 8, 12 and 16. The EMC12 hardware receive 16 bits from the DSP's serial output port during time slot pairs 4/5, 8/9, 12/13 and 16/17. The first 8 bits of each pair is the validity information, which is actually encoded in the 2nd bit received. This information is held until the second slot of each pair when it is used to drive the TDM valid line to coincide with the TDM data. No TDM data or valid is driven from the DSP during the first slot of each pair and, although the second time slot of each pair is encoded as TDM_DATA mode, the DSP only receives an OLD pulse for the time slots encoded as TDM_DSP_VALID.

Example 2:

There may arise a need to temporarily replace one or more of the time slots transmitted from the DSP with data from another source such as a Codec on the same EMC12 mezzanine or a DSP or Codec on another mezzanine. However, with DSP transmitting on a fixed number of time slots, synchronized to the TDM frame, the OLD pulses to the DSP should remain constant. When using **TDM_DSP_VALID** mode it is possible to maintain the OLD pulses that occur at the beginning of the time slots designated for the valid information, yet not have the valid information or the data portion of the DSP's 16-bit serial output driven onto the TDM bus.

In the example, below, the map using TDM_DSP_VALID mode from the previous example will be modified. Sometime after that map has been in use the data from the DSP will be replaced on time slot 9 by data from a Codec and on time slot 13 by data from a device on another mezzanine. The following function calls make the necessary changes to the map on the EMC12 mezzanine.

Note that for replacing the DSP data with Codec data, as long as the same TDM bus is used it is not necessary to delete the normal TDM slot for the DSP. Simply specifying the new source device is sufficient to properly modify the TDM map. None of the changes actually take affect until **EmWriteTdmMap** is called.

```
/* Replace DSP source on slot 9 with Codec */
EmAddTdmSrc(emc12, codec, 9, tdm_bus, TDM_DATA, 0);

/* Discontinue driving slot 13 with DSP data and valid*/
EmDelTdmSrc(emc12, dsp,  13, tdm_bus, 0);
EmWriteTdmMap(emc12);
```

The DSP will continue to receive the OLD pulses at the beginning of time slots 8 and 12, as directed by the TDM_DSP_VALID mode on those time slots. The DSP will also continue transmitting 16-bits of serial data during time slot pairs 8/9 and 12/13, but the data and valid lines are only driven from the DSP during time slots 5 and 17. Data from the Codec is now being driven onto the TDM bus during time slot 9 and nothing on this mezzanine is driving data on the TDM bus during time slot 17.

1.4 EMC12 DSP

The EMC12 is optionally populated with one Agere DSP32C, an 80 MHz, 20 MIPs, 40 MFlops digital signal processor chip. It is packaged in a 164-pin bumpered quad flat pack (BQFP). The DSP32C may be operated from 50MHz to 80 MHz.

The DSP is controlled by an FPGA (Xilinx 4013E) which provides address decoding for the parallel ports, external interrupt control, TDM interface, and interface to the status LEDs.

1.4.1 DSP Reset

The RESTN pin of the DSP is controlled by the DSP_RESET bit in BCR0. Writing a ‘0’ to this bit halts the DSP. This bit is ‘0’ after power-up and board reset.

A transition from ‘0’ to ‘1’ of this bit initiates the DSP reset sequence. Refer to the DSP32C Information Manual for in-depth discussion of halt and reset operations of the DSP.

1.4.2 DSP Interrupts

The two external interrupt pins of the DSP32C, INTREQ1 and INTREQ2, are controlled by the DSP Interrupt Control Register (DICR) in the FPGA. The various external interrupt operations and their control values are shown in Table 10.

1.4.3 DSP Clock

The serially programmable frequency generator ICS AV9110-02CS14 operates from a reference clock of 14.318 MHz (REFCLK) available on the mezzanine connector. The output clock (CLK/X) from the AV9110 is then distributed to the local clock multipliers ICS AV9170-01CS8 which quadruple the frequency. The programmability of the AV9110 lets the DSP clock frequency be selected in software. The local clock multipliers allow the AV9110 output clock to be distributed over a long distance at relatively low frequencies. TDI is connected to the DATA pin of the AV9110, and TCK is inverted and connected to the SCLK pin. To program the AV9110, set the AV9110_EN bit of BCR1 to select the AV9110, shift the serial data using TDI and TCK, and reset the AV9110_EN bit in BCR1 to deselect the AV9110.

Table 1-19 shows the format of the serial data shifted into the AV9110.
Note that bit 0 is the first bit shifted into the device.

Table 1-19: AV9110 Serial Data Format

Bit(s)	Field	Function	Default
0 - 6	N	VCO Frequency Divider	1111111b
7 - 13	M	Reference Frequency Divider	0010010b
14	V	VCO Pre-scale Divider 0b = divide by 1 1b = divide by 8	0b
15 - 16	X	CLK/X Output Divider 00b = divide by 1 01b = divide by 2 10b = divide by 4 11b = divide by 8	10b
17 - 18	R	VCO Output Divider 00b = divide by 1 01b = divide by 2 10b = divide by 4 11b = divide by 8	10b
19	CLK_EN	CLK Output Enable 0b = tristate	1b
20	CLK/X_EN	CLK/X Output Enable 0b = tristate	1b
21	Reserved	Should be 1	1b
22	CLK_SEL	Reference clock select on CLK 1b = reference frequency	0b
23	Reserved	Should be 1	1b

The output frequency of CLK/X is given by:

$$f_{CLK/X} = (f_{REF} \bullet N \bullet V) / (M \bullet R \bullet X)$$

where f_{REF} is the input reference clock.

For a reference clock of 14.318MHz, Table 20 shows the bit stream associated with several commonly used DSP clock frequencies. Note that CLK/X is multiplied by 4 to generate the DSP clock.

Table 1-20: Common AV9110 Data Stream

DSP Clock Frequency (MHz)	Bit Stream (Bit 23 - Bit 0)
50	0xB305CD
74	0xB304DD
80	0xB288DF

Refer to ICS AV9110 data sheet for more information on programming the device.

1.4.4 DSP Memory

The DSP can operate in either memory mode 6 or 7. The mode is controlled by the DSP_MMODE bit in BCR0. A '0' selects mode 6 and a '1' selects mode 7. This bit is '0' after power-up and board reset. New memory mode will not take effect until a DSP reset sequence is performed by a '0' to '1' transition of the DSP_RESET bit of BCR0.

The DSP has up to 2 Mbytes of 0-wait-state SRAM. Table 1-21 shows the address map for the DSP.

Table 1-21: DSP Memory Configurations

DSP Memory Address	Physical Memory Accessed	
	Mode 6	Mode 7
0x000000 0x0007FF	0x000000	Internal RAM0
0x000800 0x1FFFFF	0x1FFFFF	0x000800 0x1FFFFF
0xFFE000 0xFFE7FF	Internal RAM2	Internal RAM2
0xFFF000 0xFFF7FF	Internal RAM0	<none>
0xFFF800 0xFFFFFF	Internal RAM1	Internal RAM1

1.4.5 Sync Input

The Multi-Frame (LMSYNC) signal from the EmPack mezzanine connector is buffered and connected to the SY pin of the DSP. The DSP program can read the level of the signal at any time.
The LMSYNC pulse lasts one full TDM frame and occurs every N frames, where N is 1, 2, 3, ..., 32. Figure XX below shows the timing of the LMSYNC rising edge. The falling edge timing is identical but occurs one TDM frame later.

1.5 EMC12 CODECS

This section describes the basic operation of the Crystal CS4231A CODECs, as they are integrated on the EMC12 mezzanine. Additional details on the operation of the CODECs may be found in the data sheet for the CS4231A CODEC which is available on CAC's FTP site (ftp://ftp.cacdsp.com/pub/datasheets/crystal/cs4231a.pdf).

The EmPack application interface library includes several functions for controlling various settings for the EMC12 CODECs. All of them operate on a RESOURCE HANDLE obtained by calling the **EmOpenResource** function. The functions for setting parameters (**EmSetCodec…**) may be used to control individual channels or all 12 channels simultaneously if the RESOURCE HANDLE references the Broadcast resource, resource 'm' of an EMC12 mezzanine. The functions for reading current parameters (**EmGetCodec…**) may only be used for individual channels. Brief descriptions of these functions appear here. More details may be found in the EmPack Software Reference Manual.

1.5.1 Analog I/O

1.5.1.1 Gain Stages and Analog Characteristics

The figure below shows a functional diagram of the input and output gain stages for each of the 12 analog channels (2 channels per CODEC).

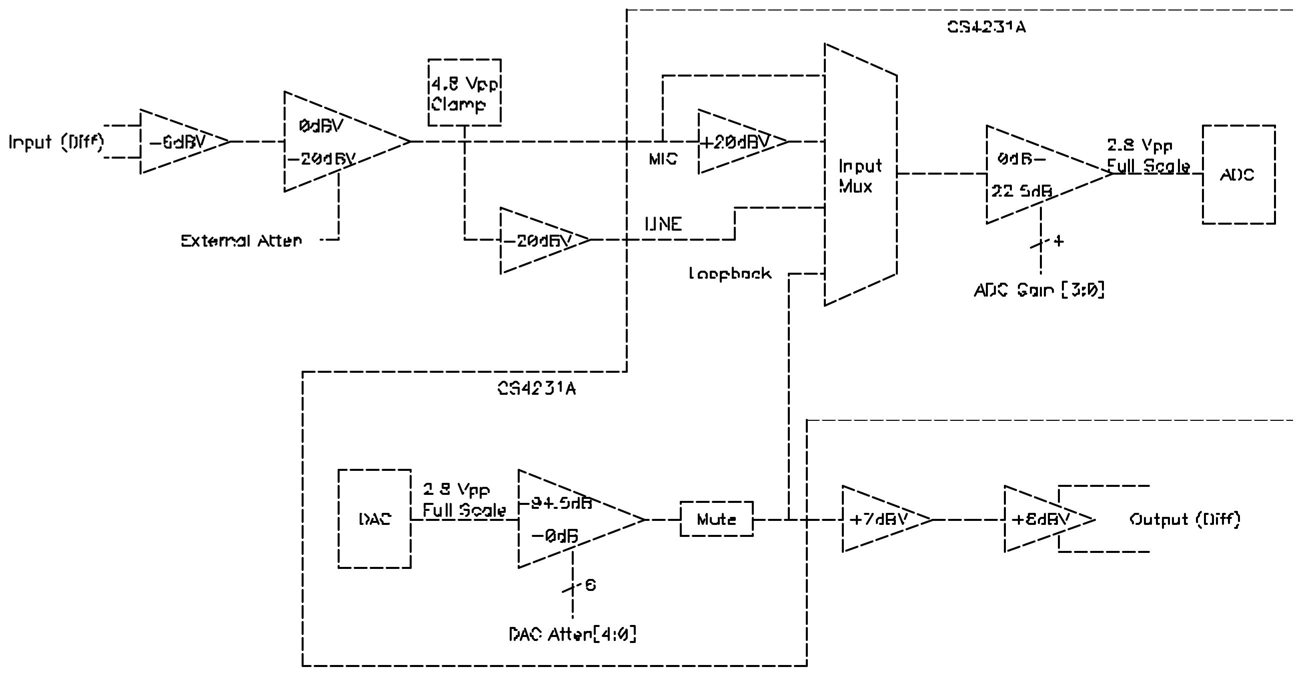


Figure 1-4: CODEC Channel Gain Stages

INPUT CHARACTERISTICS

1. The differential input amplifier introduces a -6dB attenuation, whether the input is configured as differential or single-ended (see Section 1.5.3).
2. Maximum input is 23 Vp-p.
3. “External Atten” selects either 0 or –20 dB .
4. The clamp limits the voltage to protect the CODEC inputs. Care must be taken in setting the external attenuator such that its output does not exceed ~4.8 Vp-p, at which point the signal be clamped.
5. The CODEC Input Mux can select either MIC, MIC +20dB , LINE, or the output looped back.
6. The ADC gain can be set to 0 to 22.5 dB in 16 steps of 1.5 dB
7. The overall range of input gain/attenuation is –46 dB to +36.5 dB , where 0 dB results in a full scale ADC conversion for 2.8 Vp-p at the differential input.

OUTPUT CHARACTERISTICS

1. The DAC attenuation can be set from 0 to -94.5 dB in 64 steps of 1.5 dB
2. The output has a +7 dB stage for compatibility with previous CAC products.
3. The final output gain stage introduces a +6dB gain, whether the output is configured as differential or single-ended (see Section 1.5.3).
4. The overall range of output gain/attenuation is –81.5 dB to +13 dB , where 0 dB results in a full scale voltage of 2.8 Vp-p at the differential output.

1.5.1.2 Gain Control API Functions

Output Muting:

```
int
EmSetCodecMute (
      RESOURCE_HANDLE Resource,
      BOOL Mute
)
```

```
int
EmGetCodecMute (
      RESOURCE_HANDLE Resource,
      BOOL *Mute
)
```

The **EmSetCodecMute** function controls the analog output muting. Muting is turned on (output silent) if the ***Mute*** argument is TRUE (non-zero) and turned off if it is FALSE. The **EmGetCodecMute** function returns the current mute setting in the memory pointed to by its ***Mute*** argument.

Overall Input and Output Level Settings:

```
int                                          int
EmSetCodecLevels (                           EmGetCodecLevels (
      RESOURCE_HANDLE Resource,                    RESOURCE_HANDLE Resource,
      float Input                                  float *Input
      float Output                                 float *Output
)                                            )
```

The **EmSetCodecLevels** function provides a simple method to control the overall gain / attenuation settings of the analog input and output. It adjusts all aspects of the input and output gain stages to achieve the specified gain or attenuation, selecting which CODEC input to use and taking into account the clipping level of the CODEC's input clamp. The ***Input*** argument specifies the overall input gain or attenuation in the range of –46.0 dBV to +36.5 dBV. The ***Output*** argument specifies the overall output gain or attenuation in the range of –81.5 dBV to +13 dBV. The values specified are relative to the nominal full scale level of 2.8 Vp-p. Note that the actual settings will be rounded to the nearest 1.5 dB step.

The **EmGetCodecLevels** function returns the current input and output level settings, storing the results in the memory pointed to by its ***Input*** and ***Output*** arguments.

Converting Levels from VME6U6-C12 to EMC12:

The gain stages of the EMC12 Mezzanine were designed to provide a similar range of input and output levels as the C12 Mezzanine for the VME6U6 boards. However, some differences remain. Most notably, the EMC12 gain/attenuation settings are specified in decibels and the VME6U6-C12 levels are specified as integer values used to set the digitally controlled voltage dividers in its CODECs. There is also a different nominal full scale voltage reference; 2.8 Vp-p for the EMC12 compared to 6.3 Vp-p for the VME6U6-C12.

These differences can be accommodated for with the following formulas to convert input and output settings for the VME6U6-C12 to the decibel values for the EMC12. For both formulas, ***value*** refers to the digital potentiometer values used for the VME6U6-C12 and ***dB*** refers to the EMC12 values to be passed to the **EmSetCodecLevels** function.

Input level conversion:

$$dB = 20 * \log(value / 128) - 7.0$$

Output level conversion:

$$dB = 20 * \log(value / (256 - value)) + 7.0$$

Controlling Individual Gain / Attenuation Components:

The following functions in the EMAPI are available should you find it necessary to control the various analog gain stages and signal paths separately. Please refer to the EmPack Software Reference Manual for an explanation of the functions' usage and arguments.

The functions below control the input signal path and gain stages; enabling the external –20 dB attenuator, selecting the Mic or Line input to the CODEC, enabling the CODEC's internal 20 dB gain for the Mic input and the value of the CODEC's register that sets its ADC gain.

int
EmSetCodecExtInputAtten (
    RESOURCE_HANDLE ***Resource***,
    BOOL ***ExtInputAtten***
)

int
EmGetCodecExtInputAtten (
    RESOURCE_HANDLE ***Resource***,
    BOOL ****ExtInputAtten***
)

int
EmSetCodecInputSel (
    RESOURCE_HANDLE ***Resource***,
    int ***InputSel***
)

int
EmGetCodecInputSel (
    RESOURCE_HANDLE ***Resource***,
    int ****InputSel***
)

int
EmSetCodecMicGain (
    RESOURCE_HANDLE ***Resource***,

int
EmGetCodecMicGain (
    RESOURCE_HANDLE ***Resource***,

BOOL ***MicGain***	BOOL ****MicGain***
)	)

int	int
EmSetCodecAdcGain (	**EmGetCodecAdcGain** (
RESOURCE_HANDLE ***Resource***,	RESOURCE_HANDLE ***Resource***,
int ***AdcGain***	int ****AdcGain***
)	)

The functions below control the value of the CODEC's register that sets its DAC attenuation:

int	int
EmSetCODECDacAtten (	**EmGetCODECDacAtten** (
RESOURCE_HANDLE ***Resource***,	RESOURCE_HANDLE ***Resource***,
int ***DacAtten***	int ****DacAtten***
)	)

1.5.2 Data Conversion Mode and Sampling Rate

The following two functions in the EmPack API control the analog-to-digital and digital-to-analog conversion process. The two parameters are the data conversion format and the sampling rate. The functions operate on registers inside the CS4231 CODECs as well as setting register bits in the EMC12's FPGA controlling the interface between the CODEC and the TDM subsystem.

These functions may be applied to an individual resource or the EMC12 broadcast resource. Note, however, that these operations apply to both channels in the CODEC for an individual resource. Therefore, setting the data conversion format or sampling rate for CODEC resource 'a' will set the same parameters for channel 'b'. Setting these parameters for channel 'c' will also set them for channel. 'd', etc.

int	int
EmSetCodecCvMode (	**EmGetCodecCvMode** (
RESOURCE_HANDLE ***Resource***,	RESOURCE_HANDLE ***Resource***,
int ***InputCvMode***	int ****InputCvMode***
int ***OutputCvMode***	int ****OutputCvMode***
)	)

The **EmSetCodecCvMode** function controls the digital data format (conversion mode) used by the CODEC. The modes for input and output are specified as separate arguments, ***InputCvMode*** and ***OutputCvMode***, and may be different if desired. The valid values for the conversion modes are defined as macros in the **emapi.h**. They are:

EM_CS4231_LIN8	Unsigned 8-bit Linear
EM_CS4231_LIN8_D	Unsigned 8-bit Linear with Dither (for Input only)
EM_CS4231_LIN16	Signed 16-bit Linear
EM_CS4231_LIN16_B	Signed 16-bit Linear, Big Endian (byte reversed)
EM_CS4231_ULAW	µ-Law Companded, 8-bit
EM_CS4231_ALAW	A-Law Companded, 8-bit

The **EmGetCodecCvMode** function reads the current conversion modes and stores the results in the memory pointed to by its ***InputCvMode*** and ***OutputCvMode*** arguments.

When using the 16-bit conversion modes, it is necessary to configure the TDM map to use pairs of consecutive TDM slots to carry each 16-bit sample. This is accomplished by configuring the TDM MAP using first a normal TDM slot connection, followed by a CONTINUATION slot in the succeeding time slot.

Note that the linear 8-bit conversion for the CS4231A CODEC deals with unsigned values. This is not directly compatible with the DSP32C's **iconv** and **oconv** instructions which deal with signed 8-bit values.

```
int
EmSetCodecSampleRate (
    RESOURCE_HANDLE Resource,
    int SampleRate
)
```

```
int
EmGetCodecSampleRate (
    RESOURCE_HANDLE Resource,
    int *SampleRate
)
```

The **EmSetCodecSampleRate** function controls the sample rate of the CODEC. Valid values for the ***SampleRate*** argument are defined as macros in the **emapi.h**. They are:

EM_CS4231_SR8	8 kHz Sample Rate
EM_CS4231_SR16	16 kHz Sample Rate
EM_CS4231_SR32	32 kHz Sample Rate
EM_CS4231_SR48	48 kHz Sample Rate

The TDM connections for the CODECs must be configured in accordance with the selected sample rate. The EmPack TDM subsystem runs with a fixed frame rate of 8 kHz with 128 slots per frame and 8-bits per slot. The simplest scenario is when the CODEC is set for 8 kHz sampling rate; data must be transferred between the TDM subsystem and each CODEC channel at a rate of one sample per TDM frame. However, at higher sampling rates multiple samples must be transferred during each TDM frame and the transfers for each sample must occur within certain fixed time slot ranges. The ranges of time slots for each sample depend on the selected sample rate. Table 22, below, shows the valid TDM time slot ranges for each CODEC data sample at the various sampling rates.

Table 1-22: CODEC / TDM Transfer Mapping

Sample Rate	Samples per Frame	Time Slot Range for Sample:					
		1	2	3	4	5	6
8 kHz	1	0 - 127	-	-	-	-	-
16 kHz	2	0 - 63	64 - 127	-	-	-	-
32 kHz	4	0 - 31	32 - 63	64 - 95	96 - 127	-	-
48 kHz	6	0 - 19	20 - 41	42 - 63	64 - 83	84 - 105	106 - 127

The **EmGetCodecSampleRate** function reads the current sample rate and stores the result in the memory pointed to by its ***SampleRate*** argument.

1.5.3 I/O Jumper Settings

For each of the 12 CODEC channels, there are two jumper blocks that configure the analog I/O circuitry. Figure 5 shows a diagram representing one section of the EMC12 PCB. This section is repeated 12 times in a 2 x 6 array on the board near the front audio connector.

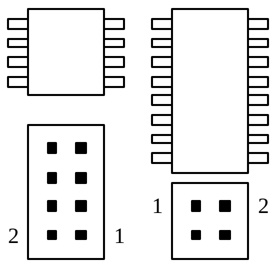


Figure 1-5: CODEC Channel Jumper PCB Section

One jumper block is 4x2 and is used as follows:

Table 1-23: 4 x 2 Jumper Block

Jumper Pair	Function
1-2	Microphone Power
2-4	DC Coupled Input
5-6	DC Differential Input
3-5	AC Coupled Single-Ended Input
5-7	DC Coupled Single-Ended Input
6-8	Single-Ended Input

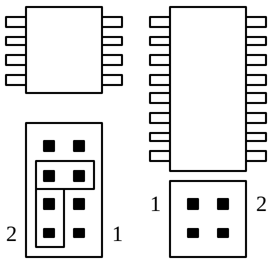
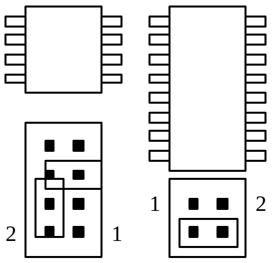
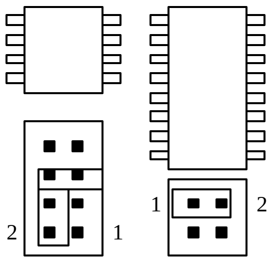
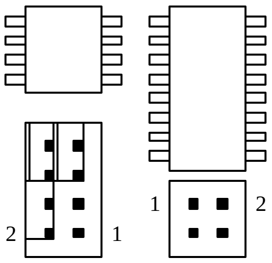
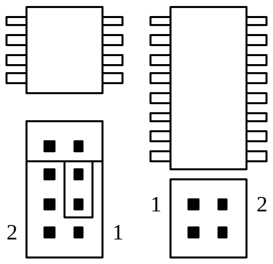
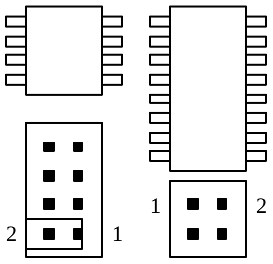
The second jumper block is 2x2 and is used as follows:

Table 1-24: 2 x 2 Jumper Block

Jumper Pair	Function
1-2	Input Terminated with 604 Ohms
3-4	Single-Ended Output

Table 1-25 shows a variety of common configurations for reference.

Table 1-25: CODEC Jumper Block Examples

DC Differential Input, Differential Output	DC Differential Input, Single-Ended Output
	
DC Differential Input, 600 Ohm Input	**DC Single-Ended Input**
	
AC Single-Ended Input	**Microphone Power**
	

1.5.4 EmPack Audio Signals

The input and output audio channels as routed through the EmPack system are differential, i.e. a pair of wires (+ and -) carry each signal. The numbering of the twelve channels start with 0 and ends with 11. Therefore, the input side of the first channel consists of the signals IN00+ and IN00-. Likewise, the outputs are labeled OUT00+ and OUT00-. In addition to the 48 signal wires, there are 20 pins that are either grounded or no-connections making a total of 68-pins. Two standard 68-pin SCSI connectors (one from AMP and the other from Thomas&Betts) and standard 68-pin SCSI cable are used for interconnect of the audio components of the EmPack system.

Figures 3-4 and 3-5 can be used to distinguish between the AMP and T&B styles of connectors and to locate pin 1 for each type. Note that these figures represent the connectors soldered onto the circuit board and not the cable connector.

Typically, Thomas&Betts (T&B) connectors are used between the EmPack codec cards mounted inside the card cage and the rear-panel circuit board. Flat ribbon cable is used for the interconnection.

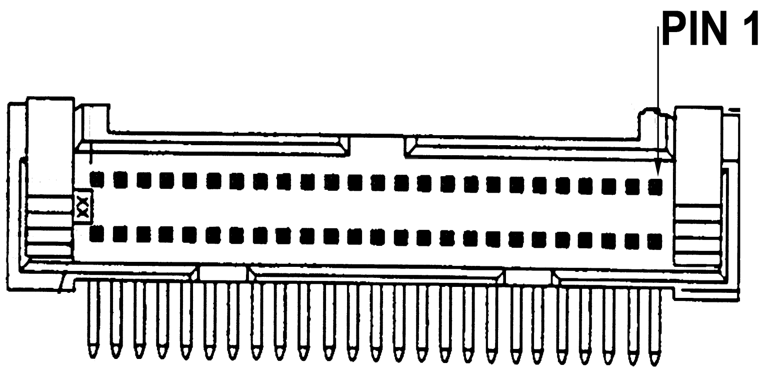


Figure 1-6: Thomas & Betts connector

AMP connectors are used between the outside of the rear-panel and the audio patch panel. Standard external SCSI cables with thumbscrews or latches are used to make these connections.

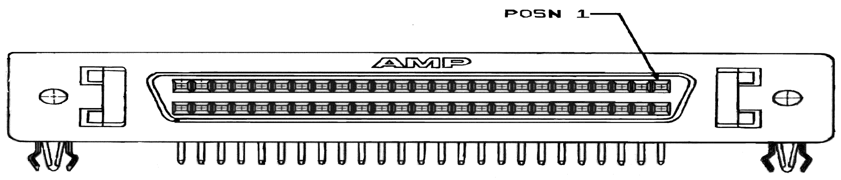


Figure 1-7: AMP connector

Table 1-26: AMP and T&B OUT Signals

Signal	AMP	T&B
OUT00+	35	65
OUT00-	1	66
OUT01+	37	61
OUT01-	3	62
OUT02+	41	53
OUT02-	7	54
OUT03+	43	49
OUT03-	9	50
OUT04+	46	43
OUT04-	12	44
OUT05+	48	39
OUT05-	14	40
OUT06+	54	27
OUT06-	20	28
OUT07+	57	21
OUT07-	23	22
OUT08+	59	17
OUT08-	25	18
OUT09+	61	13
OUT09-	27	14
OUT10+	63	9
OUT10-	29	10
OUT11+	66	3
OUT11-	32	4
GND	5	--
GND	6	--
GND	11	--
GND	21	--
GND	30	--
GND	34	--
NC	16	7
NC	17	8
NC	18	25
NC	19	26
NC	--	29
NC	--	30
NC	--	31
NC	--	32
NC	--	33
NC	--	34

Table 1-27: AMP and T&B IN Signals

Signal	AMP	T&B
IN00+	2	64
IN 00-	36	63
IN 01+	4	60
IN 01-	38	59
IN 02+	8	52
IN 02-	42	51
IN 03+	10	48
IN 03-	44	47
IN 04+	13	42
IN 04-	47	41
IN 05+	15	38
IN 05-	49	37
IN 06+	22	24
IN 06-	56	23
IN 07+	24	20
IN 07-	58	19
IN 08+	26	16
IN 08-	60	15
IN 09+	28	12
IN 09-	62	11
IN 10+	31	6
IN 10-	65	5
IN 11+	33	2
IN 11-	67	1
GND	39	--
GND	40	--
GND	45	--
GND	55	--
GND	64	--
GND	68	--
NC	50	35
NC	51	36
NC	52	45
NC	53	46
NC	--	55
NC	--	56
NC	--	57
NC	--	58
NC	--	67
NC	--	68

1.6 Display LEDs

The EMC12 has a pair of status LEDs controlled by the DSP. The DSP can access these indic ators by writing to any address between 0x800000 and 0xBFFFFF. The pair of LEDs display the value of the two least significant bits written. The green LEDs corresponds to D1 and the yellow LED corresponds to D0.
Note that, due to the physical memory wrapping around the 24-bit address space, writing to the LEDs will also write data to the corresponding location in external RAM.

1.7 Temperature Sensor

The EMC12 has a DS1620 temperature sensor. The TCK and TDI signals are connected to the CLK and DQ pins of the DS1620. To send/receive serial data to/from the device, set the DS1620_EN bit in BCR1. Reset the bit to finish the data transmission. Refer to the Dallas DS1620 Data Sheet for information on the serial data format.

1.8 Mechanical Characteristics

The EMC12 is implemented as a mezzanine in the EmPack system. It is secured to a heat spreader which is, in turned, mounted on the EmPack card cage. The overall dimension of the board with the spreader is 8.1" long, 5.5" wide, 0.3" high. The bottom mezzanine connector (P2) is a 60x2 position plug and the top connector (J1) is a matching 60x2 position receptacle. The EMC12 board must be separated from the EmPack baseboard and any other EMC12 mezzanine by either an EM6X32C mezzanine or an EBX1 spacer.

A separate 4-pin standard disk drive power connector (P1) supplies +5V. The +12V is not used on this board.

1.9 Power Consumption

The EMC12 requires +5V, and +12V. The power consumption requirement depends on the clock speed and the activity on the board. Power consumption has been empirically found to be as follows for boards populated with and without a DSP:

Product part Number	Supply	Current	Power
EBC2 (without DSP)	5 volts	0.60 A	3.0 W
	12 volts	0.25 A	3.0 W
EBC1 (with DSP @ 74 MHz)	5 volts	1.10 A	5.5 W
	12 volts	0.25 A	3.0 W

1.10 Software

Software drivers for Solaris, Windows NT and Window 2K are provided, together with a host API to allow access to and control of the EMC12 from user-developed programs. Diagnostics are provided which are sufficient to verify correct operation of all functional blocks on the EMC12.

1.11 Reference Documents

1. Crystal Semiconductor CS4231A Data Sheet.
2. Xilinx XC4000 Series FPGA Data Sheet, September 1996.
3. AT&T DSP32C Digital Signal Processor Data Sheet, February 1995.