PM73140/ PM73141

MECA-4A/ MECA-4I

VOICE OVER PACKET PROCESSOR

DATA SHEET

PROPRIETARY AND CONFIDENTIAL

PRELIMINARY

ISSUE 1: AUGUST 2000

REVISION HISTORY

Issue No.	Issue Date	ECN Number	Originator	Details of Change
1	March 2000		Malleable Technologies	Document Inherited
1	August 2000		PMC-Sierra, Inc.	Document converted to Word 97 from Frame

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 System-on-a-chip for Voice Processing

The MECA-4x is a high-performance chip for processing voice over ATM and IP networks. The chip's programmable DSP cores are delivered along with Firmware, an API, and device drivers. There are different Firmware versions for ATM and for IP, as well as for different processing options (such as voice coders). Different versions of the chip optimized for ATM or IP networks are designated as MECA-4A (for ATM) and MECA-4I (for IP). Part numbers used for ordering include designators for Firmware versions. This document covers both MECA-4A and MECA-4I, which are collectively referred to as MECA-4x. Detailed information about ATM and IP Firmware versions is found in the appropriate Firmware Datasheets. These give information on the capabilities and access protocols associated with that particular firmware image.

This document concentrates on:

- The I/O capabilities of the MECA-4x, including register-level descriptions of all I/O cores
- A high-level description of the way chip hardware, firmware, and software interact to create application-level solutions
- Chip-level datasheet information, such as pin-out and timing diagrams

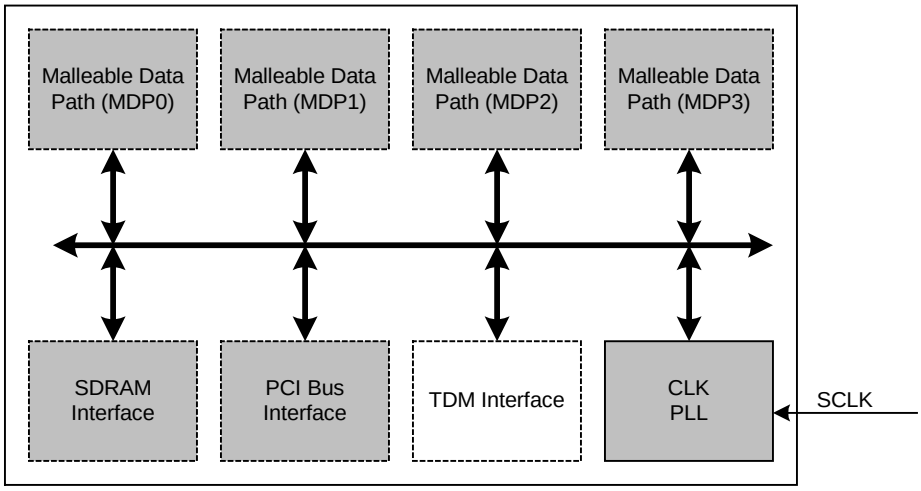
The MECA-4x chip architecture includes four instances of PMC-Sierra's proprietary processor core, which is called a Multi-Processor Data Path, or MDP. The intended method of using these parallel computing resources is to perform the same or similar processing on different data with the different MDP cores. This is a much simpler technical problem than using parallel computing resources in a tightly-coordinated fashion to solve a more computationally intensive problem on a single dataset. Thus, the design is optimized for applications which require many channels to be processed with a fixed DSP algorithm, and where a moderate number of additional DSP algorithms are required to be simultaneously resident. For example, a single MDP core running at 100MHz can process 24 channels of the G.729A speech coding algorithm (including both encode and decode) or 16 channels of the G.723.1 algorithm. Thus, a four-MDP MECA-4x chip can process 96 channels of G.729A or 64 channels of G.723.1, or it can combine them in various proportions. For applications requiring even more power or a more heterogeneous mixture of algorithms, design tradeoffs can be made to balance the workload between a CPU and a MECA-4x, or among multiple MECA-4x's.

1.1.1 Overview

The MECA-4x incorporates multiple high-performance programmable computing elements, a high-bandwidth memory interface, a standard PCI bus interface, and generic serial/parallel interfaces that allow glueless connection to a wide variety of chips and protocols. Set-up and management may be arranged through the PCI bus by an external CPU. A set of general-purpose I/O pins can be used to communicate with the MECA-4x. The MECA-4x includes a sophisticated, flexible on-chip bus which can be configured to guarantee bandwidth and latency for streaming data flows while still maintaining high throughput for random and asynchronous flows.

The four Multi-Processor Data Path (MDP) cores provide efficient, programmable, computational power within the MECA-4x for executing a wide variety of demanding voice processing and networking algorithms. The MDPs are designed to perform the computationally-intensive inner loops of these algorithms, where signal data is processed. MDPs may also perform statistics-gathering and similar functions. The MDPs have been designed to the requirements of industry-standard speech and networking algorithms including speech vocoders. MDP program memory size and instruction flexibility are designed to avoid requiring tightly-coupled interaction with a CPU to perform their tasks. When interaction with a host CPU is required, a flexible, high-bandwidth interface to the MECA-4x's register files is available via the PCI bus.

Figure 1 Block Diagram of the MECA-4x Processor



1.2 Benchmarks

A summary of the principal benchmarks of the MECA-4x are as follows.

- Supports ITU standards G.729A, G.723.1, G.711, and G.726
- G.168-compliant 8ms to 64ms adaptive echo cancellation. Channel capacity for various echo-cancellation tail lengths:

Echo Tail (in ms):	0	16	32	64
G.729A Channels:	96	72	64	48
G.711 Channels:	512	384	192	96

- In addition to the above: Silence Suppression, Voice Activity Detection (VAD) and Comfort Noise Generation (CNG)

1.3 Features

A summary of the principal features of the MECA-4x are as follows.

- Fully-programmable Voice Over Internet Protocol (VoIP) Processor
    - 100 MHz clock rate
    - Less than 2 Watts power consumption
- Speech Processing
    - Multiple ITU standards supported, including: G.711, G.726, G.729A, G723.1
    - Comfort Noise Generation
    - Voice Activity Detection & silence suppression
    - Echo Cancellation (1-65 ms tail)
    - DTMF Detection
    - Automatic Gain Control

- Fax, modem signal detection & forwarding

- Packet Processing
    - Speech-to-packets and packets-to-speech
    - IP (MECA-4I): Ethernet/UDP/IP/RTP header formatting
    - ATM (MECA-4A): AAL2 cell formatting and extraction
- TDM Interface
    - T1/E1 framer/LIUs (up to 4 groups of 1-8 framer/LIUs each)
    - MVIP (up to 2, 2.048 or 8.192 MHz)
    - H.100/H.110
    - Firmware-based time-slot selection
- Packet Interface
    - PCI for IP packets
    - 8-bit wide Utopia level 2 for ATM cells
- SDRAM Controller
    - 32 bits wide, 64-bit burst
    - Address space up to 32 MB
    - 400 MB/sec peak bandwidth
    - Automatic Refresh Generation
- PCI 2.1 Bus Interface
    - 32-bit PCI bus interface supports 33 MHz operation
    - Interface can be Target or Master

- DMA Controller
    - Eight general-purpose DMA channels
    - Any data source can be sent to any destination
    - Supports scatter/gather, incrementing or non-incrementing source and destination
    - Flexible descriptor-based operation
    - Programmable bus transaction size
- Standard Technology
    - 0.25um CMOS Process
    - 2.5V Internal power supply
    - 3.3V I/O (5V tolerant PCI)
    - 316-pin BGA Package (See Figure 2)

2 DATA SHEET

2.1 Packaging

The MECA-4x chip is packaged in a 316-PBGA. The nominal outer dimensions of this package are 27mm on a side. Figure 2 shows the pin assignments for this package.

Figure 2 316-PBGA Pin Assignments (Top View)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
A	NC	TDO	TDIN24	TDINSTRB3	TDINFRM3	TDOUT24	TDOUTFRM3	TDOUT30	NC	GPIO4	GPIO9	GPIO10	GPIO14	GPIO15	SRESET_	SDA3	SDA1	SDA0	SDA8	NC
B	TDI	G	TRST	TDIN26	TDIN28	G	TDOUTSTRB3	TDOUT31	G	GPIO3	GPIO8	G	GPIO13	NC	G	SDA5	SDA6	SDA7	G	SDA10
C	TDOUT23	TCK	TMS	TDIN25	P	TDIN31	TDOUT27	TDOUT29	P	GPIO3	GPIO7	P	GPIO12	NC	INTR_	P	SDA2	SDA11	SDBA1	SDA9
D	TDOUT22	TDOUT21	TDOUT20	C	TDIN27	TDIN30	TDOUT26	TDOUT28	NC	GPIO1	GPIO6	NC	GPIO11	NC	NC	SDA4	C	SDCS1_	SDA12	SDBA0
E	TDOUTSTRB2	TDOUTFRM2	P	TDOUT19	G	TDIN29	TDOUT25	C	G	GPIO0	GPIO5	G	C	NC	NC	G	SDCS0_	P	SDWE_	SDCAS_
F	TDOUT18	G	TDOUT17	TDOUT16	TDIN22											SDCK	SDRAS_	SDCE	G	SDDQM3
G	TDINFRM2	TDIN22	TDIN21	TDIN20	TDIN19											SDDQM1	SDDQM2	SDDQM0	SDDQ7	SDDQ8
H	TDINSTRB2	TDIN18	TDIN17	TDIN16	C											C	SDDQ11	SDDQ5	SDDQ10	SDDQ6
J	TDOUT15	G	P	NC	G				GT	GT	GT	GT				G	NC	P	G	SDDQ9
K	TDOUT14	TDOUT13	TDOUT12	TDOUTFRM1	TDOUTSTRB1				GT	GT	GT	GT				SDDQ13	SDDQ3	SDDQ12	SDDQ4	SDDQ14
L	TDOUT11	TDOUT10	TDOUT9	TDOUT8	TDIN15				GT	GT	GT	GT				SDDQ24	SDDQ0	SDDQ15	SDDQ1	SDDQ2
M	TDIN14	G	P	NC	G				GT	GT	GT	GT				G	NC	P	G	SDDQ23
N	TDIN13	TDIN12	TDINFRM1	TDINSTRB1	C											C	SDDQ25	SDDQ22	SDDQ26	SDDQ21
P	TDIN11	TDIN10	TDIN9	TDIN8	TDOUT7											SDDQ27	SDDQ20	SDDQ28	SDDQ19	SDDQ29
R	TDOUTFRM0	G	TDOUT6	TDOUT5	TDOUT4											SDDQ18	SDDQ30	SDDQ17	G	SDDQ31
T	TDOUTSTRB0	TDOUT3	P	TDOUT2	G	PAD29	PAD24	C	G	PAD16	PDEVSEL_	G	C	PAD12	PAD8	G	NC	P	SCLK	SDDQ16
U	TDOUT1	TDOUT0	TDIN7	C	PGNT_	PAD30	PAD25	PAD22	NC	PAD17	PTRDY_	NC	PPAR	PAD13	PAD9	PAD6	C	C_{VDD1}	C_{VDD2}	SCLKOUT
V	TDIN6	TDIN5	TDIN4	TDIN0	P	PAD31	PAD26	PAD23	P	PAD18	PIRDY_	P	PSERR_	PAD14	PAD10	P	PAD3	NC	PLLMODE0	C_{VD3}
W	TDINFRM0	G	TDIN2	TDIN1	PCLK	G	PAD27	PIDSEL	G	PAD19	PFRAME_	G	PPERR_	PAD15	G	PAD7	PAD4	PAD1	G	PLLMODE1
Y	NC	TDINSTRB0	TDIN3	PINT_	PRST_	PREQ_	PAD28	PCBE3_	PAD21	PAD20	PCBE2_	PSTOP_	PLOCK_	PCBE1_	PAD11	PCBE0_	PAD5	PAD2	PAD0	NC

PCI TDOUT TDIN SD Adr SD Data TEST GPIO

The balls marked with letters are interpreted as follows:

Letter	Meaning
GT	Thermal Grounds
G	Ground
P	Power
NC	Not Connected

Other groups of balls are pattern-coded according to logical groupings, which are keyed at the bottom of the Figure, and further discussed below.

2.2 System Environment

2.2.1 Overview

A high level block diagram of the MECA-4x is as shown in Figure 1.

The four MDP's plus the three I/O interfaces are connected by a single high-performance bus that has 64-bit data lines and 32-bit address lines. This on-chip bus is designed to handle high bandwidth inter-core data traffic. But more importantly, the on-chip bus can interleave many logically independent flows of data, and can do this while balancing the conflicting demands of low-latency data access (with real-time performance guarantees where appropriate) with sustained and high peak-bandwidth data access as well. See Section 3, Architectural Overview.

The MECA-4x is synchronously clocked except at the PCI and TDM interfaces. An external oscillator supplies the system clock (SCLK) which is multiplied by an internal PLL to provide SBCLK. The maximum value for SBCLK in the initial production of the MECA-4x is 100MHz.

In the typical configuration of an MECA-4x in a voice processing application, an external CPU loads firmware code and controls the MECA-4x through the PCI bus while data normally flows through the TDM interface. Variations on this model include sending packetized data through the PCI bus to another PCI card while PCM data passes through the TDM interface.

In most cases data flows can be set up to DMA directly into and out of buffers and/or hardware FIFOs within each MDP, and also to SDRAM buffers. Rate matching can be performed automatically because the MDPs can halt and resume execution based on the fullness of their FIFOs.

2.2.2 Chip interfaces

The MECA-4x includes the following interfaces.

- SDRAM Interface
- PCI Bus Interface
- TDM Interface
- General-purpose I/O Interface
- Miscellaneous signals
- JTAG signals

These interfaces are shown in Figure 4 to Figure 7.

Figure 3 TDM Interface

Figure 4 SDRAM Interface

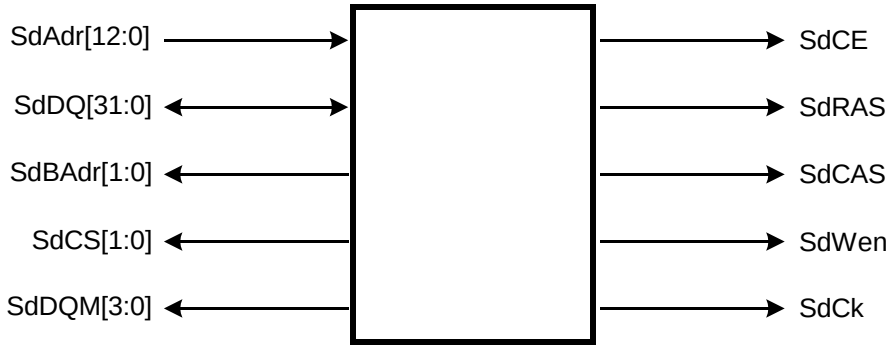


Figure 5 PCI Bus Interface

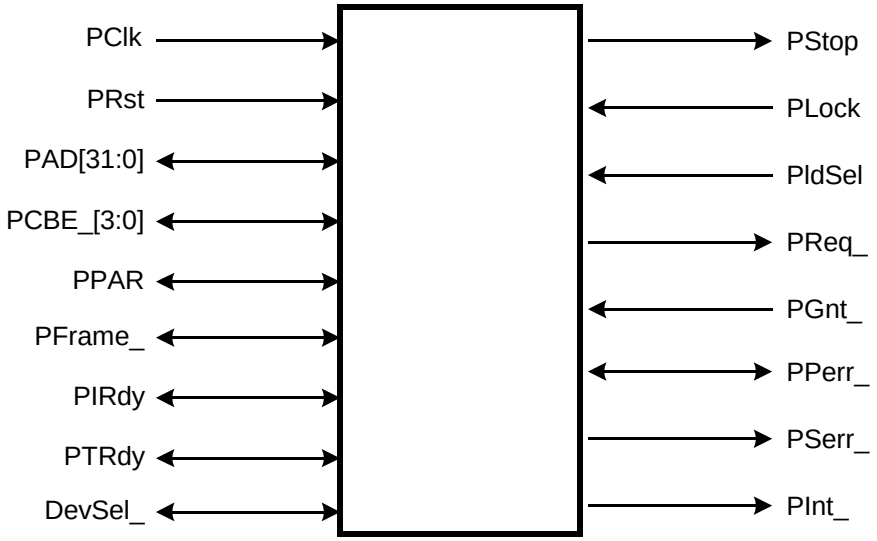


Figure 6 General-purpose I/O and Miscellaneous Signals

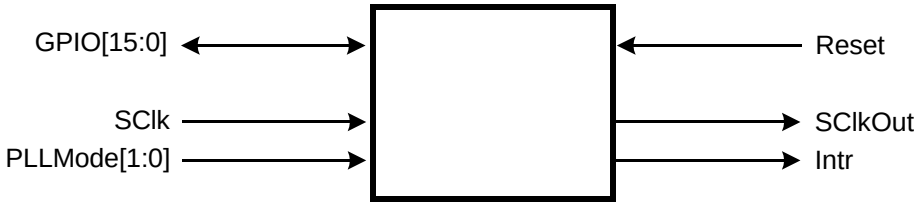
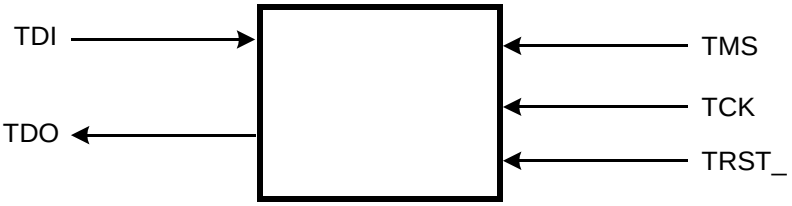


Figure 7 Test Interface Signals



2.2.3 Pin Descriptions

Table 1 shows the prefixes which are used to identify the port each signal is associated with.

Table 1 Pin Prefix Definitions

Prefix	Port
GP	General-purpose pins
T	Test interface
P	PCI
SD	SDRAM interface
TD	TDM input and output interfaces

The underscore suffix ("_") is used to indicate active low signals.

Table 2 Interface Pin Descriptions

	Signal Name	Pin Number	Type	Description
General Purpose (16)				
1	GPIO0	E10	I/O	General Purpose I/O's
2	GPIO1	D10	I/O	
3	GPIO2	C10	I/O	
4	GPIO3	B10	I/O	
5	GPIO4	A10	I/O	
6	GPIO5	E11	I/O	
7	GPIO6	D11	I/O	
8	GPIO7	C11	I/O	
9	GPIO8	B11	I/O	
10	GPIO9	A11	I/O	
11	GPIO10	A12	I/O	
12	GPIO11	D13	I/O	
13	GPIO12	C13	I/O	
14	GPIO13	B13	I/O	
15	GPIO14	A13	I/O	
16	GPIO15	A14	I/O	

	Signal Name	Pin Number	Type	Description
Miscellaneous (6)				
17	SCLK	T19	I	System Input clock
18	SRESET_	A15	I	Reset
19	INTR_	C15	O	Interrupt (Note 1)
20	SCLKOUT	U20	O	CSystem Clock out
21	PLLMODE0	V19	I	PLL Mode
22	PLLMODE1	W20	I	
JTAG Interface (5)				
23	TDI	B1	I	Test data in (Note 2)
24	TCK	C2	I	Test clock input (Note 2)
25	TDO	A2	O	Test data out
26	TMS	C3	I	Test Mode Select (Note 2)
27	TRST_	B3	I	Test reset (Note 3)
PCI Interface (51)				
28	PCLK	W5	I	PCI Clock
29	PRST_	Y5	I	PCI Reset
30	PAD0	Y19	I/O	PCI Address/Data bus
31	PAD1	W18	I/O	
32	PAD2	Y18	I/O	
33	PAD3	V17	I/O	
34	PAD4	W17	I/O	
35	PAD5	Y17	I/O	
36	PAD6	U16	I/O	
37	PAD7	W16	I/O	
38	PAD8	T15	I/O	
39	PAD9	U15	I/O	
40	PAD10	V15	I/O	
41	PAD11	Y15	I/O	
42	PAD12	T14	I/O	
43	PAD13	U14	I/O	
44	PAD14	V14	I/O	
45	PAD15	W14	I/O	

	Signal Name	Pin Number	Type	Description
46	PAD16	T10	I/O	
47	PAD17	U10	I/O	
48	PAD18	V10	I/O	
49	PAD19	W10	I/O	
50	PAD20	Y10	I/O	
51	PAD21	Y9	I/O	
52	PAD22	U8	I/O	
53	PAD23	V8	I/O	
54	PAD24	T7	I/O	
55	PAD25	U7	I/O	
56	PAD26	V7	I/O	
57	PAD27	W7	I/O	
58	PAD28	Y7	I/O	
59	PAD29	T6	I/O	
60	PAD30	U6	I/O	
61	PAD31	V6	I/O	
62	PCBE0_	Y16	I/O	Command field during address phase, Byte enables during data phase
63	PCBE1_	Y14	I/O	
64	PCBE2_	Y11	I/O	
65	PCBE3_	Y8	I/O	
66	PPAR	U13	I/O	Parity
67	PFRAME_	W11	I/O	Frames a master transaction
68	PIRDY_	V11	I/O	Initiator Ready
69	PTRDY_	U11	I/O	Target Ready
70	PSTOP_	Y12	O	Stop
71	PLOCK_	Y13	I	Lock
72	PIDSEL	W8	I	Initialization Device Select
73	PDEVSEL_	T11	I/O	Device Select
74	PREQ_	Y6	O	Bus Request
75	PGNT_	U5	I	Bus Grant

	Signal Name	Pin Number	Type	Description
76	PPERR_	W13	I/O	Parity Error
77	PSERR_	V13	O	System Error (Note 4)
78	PINT_	Y4	O	Interrupt Request (Note 4)
TDM Interface (80)				
79	TDIN0	V4	I	TDM Input Data
80	TDIN1	W4	I	
81	TDIN2	W3	I	
82	TDIN3	Y3	I	
83	TDIN4	V3	I	
84	TDIN5	V2	I	
85	TDIN6	V1	I	
86	TDIN7	U3	I	
87	TDIN8	P4	I	
88	TDIN9	P3	I	
89	TDIN10	P2	I	
90	TDIN11	P1	I	
91	TDIN12	N2	I	
92	TDIN13	N1	I	
93	TDIN14	M1	I	
94	TDIN15	L5	I	
95	TDIN16	H4	I	
96	TDIN17	H3	I	
97	TDIN18	H2	I	
98	TDIN19	G5	I	
99	TDIN20	G4	I	
100	TDIN21	G3	I	
101	TDIN22	G2	I	
102	TDIN23	F5	I	
103	TDIN24	A3	I	
104	TDIN25	C4	I	
105	TDIN26	B4	I	
106	TDIN27	D5	I	

	Signal Name	Pin Number	Type	Description
107	TDIN28	B5	I	
108	TDIN29	E6	I	
109	TDIN30	D6	I	
110	TDIN31	C6	I	
111	TDOUT0	U2	O	TDM Output Data
112	TDOUT1	U1	O	
113	TDOUT2	T4	O	
114	TDOUT3	T2	O	
115	TDOUT4	R5	O	
116	TDOUT5	R4	O	
117	TDOUT6	R3	O	
118	TDOUT7	P5	O	
119	TDOUT8	L4	O	
120	TDOUT9	L3	O	
121	TDOUT10	L2	O	
122	TDOUT11	L1	O	
123	TDOUT12	K3	O	
124	TDOUT13	K2	O	
125	TDOUT14	K1	O	
126	TDOUT15	J1	O	
127	TDOUT16	F4	O	
128	TDOUT17	F3	O	
129	TDOUT18	F1	O	
130	TDOUT19	E4	O	
131	TDOUT20	D3	O	
132	TDOUT21	D2	O	
133	TDOUT22	D1	O	
134	TDOUT23	C1	O	
135	TDOUT24	A6	O	
136	TDOUT25	E7	O	
137	TDOUT26	D7	O	
138	TDOUT27	C7	O	

	Signal Name	Pin Number	Type	Description
139	TDOUT28	D8	O	
140	TDOUT29	C8	O	
141	TDOUT30	A8	O	
142	TDOUT31	B8	O	
143	TDINSTRB0	Y2	I	TDM Input Strobes
144	TDINSTRB1	N4	I	
145	TDINSTRB2	H1	I	
146	TDINSTRB3	A4	I	
147	TDOUTSTRB0	T1	I	TDM Output Strobes
148	TDOUTSTRB1	K5	I	
149	TDOUTSTRB2	E1	I	
150	TDOUTSTRB3	B7	I	
151	TDINFRM0	W1	I	TDM Input Framing Signals
152	TDINFRM1	N3	I	
153	TDINFRM2	G1	I	
154	TDINFRM3	A5	I	
155	TDOUTFRM0	R1	O	TDM Output Framing Signals
156	TDOUTFRM1	K4	O	
157	TDOUTFRM2	E2	O	
158	TDOUTFRM3	A7	O	
SDRAM Interface (58)				
159	SDA0	A18	O	Row/Col address bus
160	SDA1	A17	O	
161	SDA2	C17	O	
162	SDA3	A16	O	
163	SDA4	D16	O	
164	SDA5	B16	O	
165	SDA6	B17	O	
166	SDA7	B18	O	
167	SDA8	A19	O	
168	SDA9	C20	O	
169	SDA10	B20	O	

	Signal Name	Pin Number	Type	Description
170	SDA11	C18	O	
171	SDA12	D19	O	
172	SDBA0	D20	O	Bank addresses
173	SDBA1	C19	O	
174	SDDQ0	L17	I/O	SDRAM data bus
175	SDDQ1	L19	I/O	
176	SDDQ2	L20	I/O	
177	SDDQ3	K17	I/O	
178	SDDQ4	K19	I/O	
179	SDDQ5	H18	I/O	
180	SDDQ6	H20	I/O	
181	SDDQ7	G19	I/O	
182	SDDQ8	G20	I/O	
183	SDDQ9	J20	I/O	
184	SDDQ10	H19	I/O	
185	SDDQ11	H17	I/O	
186	SDDQ12	K18	I/O	
187	SDDQ13	K16	I/O	
188	SDDQ14	K20	I/O	
189	SDDQ15	L18	I/O	
190	SDDQ16	T20	I/O	
191	SDDQ17	R18	I/O	
192	SDDQ18	R16	I/O	
193	SDDQ19	P19	I/O	
194	SDDQ20	P17	I/O	
195	SDDQ21	N20	I/O	
196	SDDQ22	N18	I/O	
197	SDDQ23	M20	I/O	
198	SDDQ24	L16	I/O	
199	SDDQ25	N17	I/O	
200	SDDQ26	N19	I/O	

	Signal Name	Pin Number	Type	Description
201	SDDQ27	P16	I/O	
202	SDDQ28	P18	I/O	
203	SDDQ29	P20	I/O	
204	SDDQ30	R17	I/O	
205	SDDQ31	R20	I/O	
206	SDCS0_	E17	O	Chip Selects
207	SDCS1_	D18	O	
208	SDCK	F16	O	Clock
209	SDCE	F18	O	Clock Enable
210	SDRAS_	F17	O	Row Address Select
211	SDCAS_	E20	O	Column Address Select
212	SDWE_	E19	O	Write Enable
213	SDDQM0	G18	O	DQ Masks for bytes 0-3
214	SDDQM1	G16	O	
215	SDDQM2	G17	O	
216	SDDQM3	F20	O	
Power & Ground - Package Pins (60)				
217	CVDD	H5	CS	2.5 Supply Voltage
218	CVDD	N5	CS	
219	CVDD	U4	CS	
220	CVDD	T8	CS	
221	CVDD	T13	CS	
222	CVDD	U17	CS	
223	CVDD1	U18	CS	PLL VDD - Guard Ring
224	CVDD2	U19	CS	PLL VDD - Analog
225	CVDD3	V20	CS	PLL VDD - Digital
226	CVDD	N16	CS	
227	CVDD	H16	CS	
228	CVDD	D17	CS	
229	CVDD	E13	CS	
230	CVDD	E8	CS	
231	CVDD	D4	CS	

	Signal Name	Pin Number	Type	Description
232	PVDD	E3	PS	3.3 Supply Voltage On a power ring
233	PVDD	J3	PS	
234	PVDD	M3	PS	
235	PVDD	T3	PS	
236	PVDD	V5	PS	
237	PVDD	V9	PS	
238	PVDD	V12	PS	
239	PVDD	V16	PS	
240	PVDD	T18	PS	
241	PVDD	M18	PS	
242	PVDD	J18	PS	
243	PVDD	E18	PS	
244	PVDD	C16	PS	
245	PVDD	C12	PS	
246	PVDD	C9	PS	
247	PVDD	C5	PS	
248	VSS	B2	GND	Ground Pins (Core & I/O) On a power ring
249	VSS	E5	GND	
250	VSS	F2	GND	
251	VSS	J5	GND	
252	VSS	J2	GND	
253	VSS	M5	GND	
254	VSS	M2	GND	
255	VSS	R2	GND	
256	VSS	T5	GND	
257	VSS	W2	GND	
258	VSS	W6	GND	
259	VSS	T9	GND	
260	VSS	W9	GND	
261	VSS	W12	GND	

	Signal Name	Pin Number	Type	Description
262	VSS	T12	GND	
263	VSS	W15	GND	
264	VSS	T16	GND	
265	VSS	W19	GND	
266	VSS	R19	GND	
267	VSS	M16	GND	
268	VSS	M19	GND	
269	VSS	J16	GND	
270	VSS	J19	GND	
271	VSS	F19	GND	
272	VSS	E16	GND	
273	VSS	B19	GND	
274	VSS	B15	GND	
275	VSS	E12	GND	
276	VSS	B12	GND	
277	VSS	E9	GND	
278	VSS	B9	GND	
279	VSS	B6	GND	
280	NC	A1	NC	No Connects
281	NC	A9	NC	
282	NC	A20	NC	
283	NC	B14	NC	
284	NC	C14	NC	
285	NC	D9	NC	
286	NC	D12	NC	
287	NC	D14	NC	
288	NC	D15	NC	
289	NC	E14	NC	
290	NC	E15	NC	
291	NC	J4	NC	
292	NC	J17	NC	
293	NC	M4	NC	

	Signal Name	Pin Number	Type	Description
294	NC	M17	NC	
295	NC	T17	NC	
296	NC	U9	NC	
297	NC	U12	NC	
298	NC	V18	NC	
299	NC	Y1	NC	
300	NC	Y20	NC	

Notes:

1. Open Drain Output with weak pullup
2. Input with weak pullup
3. Input with weak pulldown
4. Open Drain Output

3 ARCHITECTURAL OVERVIEW

3.1 Architectural Features

The power and efficiency of the MDP cores can be attributed to the following architectural features.

- The use of a simple but high-bandwidth memory hierarchy. Each MDP core has 16K bytes of register memory, of which 128 bits are available to the processing units on each cycle. These 128 bits are accessed from up to 4 different addresses per cycle, a flexibility that further increases the efficiency of the MDP core.

- MDP cores do not perform load/store operations. The register files include flexible, built-in support for FIFOs. This frees the MDP cores from the burden of performing load and store operations to accomplish simple data movement.

- Multiply/add function unit. A novel multiply/add function unit can launch 4 useful 16x16 multiplies or a single 32x32 multiply on every cycle. In the 16x16 multiply case, the function unit provides a set of prewired options for selecting the multiplier inputs from up to 128 different input bits, and combining the multiplier outputs. These options cover all the common DSP inner loops, including FIR, IIR, correlation, etc.

- Logic unit. A novel logic unit allows 64 bits to be permuted and copied in arbitrary ways. It permits the complex boolean logic computations found in networking and communications applications to be performed very efficiently.

3.2 Multi-Processor Datapath Cores

The MECA-4x contains 4 Multi-Processor Datapath cores, or MDP cores. The MDP cores are self-contained computing engines optimized to accelerate the most computationally-intensive parts of a variety of communications algorithms, including speech compression and related speech processing algorithms. PMC-Sierra supplies firmware for these cores.

3.2.1 FIFOs

The MDP cores each have hardware-supported FIFOs, two for input and two for output, each with variable depth. The memory for these FIFOs is in the MDP register files, so each FIFO is tightly coupled to its processing resources.

Inputs are written to the register file by *borrowing* cycles or *stealing* them. Cycles which are unused by the MDPs can be borrowed by the FIFOs without affecting program flow (since they are otherwise unused). However if cycles cannot be borrowed by a programmable deadline, the FIFOs can steal cycles by forcing a pipeline stall to the MDPs long enough to accomplish the required transaction.

Outputs are read from the register file by the output FIFOs in the same way. The MDPs have unrestricted access to the contents of all FIFOs at all times.

The input and output FIFO interfaces to the MDP can be used in a number of ways. The most common and obvious use is to move data streams among the MDPs and I/O ports in the MECA-4x using the DMA engines associated with certain of the I/O ports.

3.3 Input/Output Blocks

3.3.1 SDRAM Interface

The SDRAM interface is designed to interface to 8–32 MB of SDRAM in 1–4 packages of 64, 128, or 256 Mb using a 32-bit data bus. The SDRAM interface allows CAS latency to be configured to 1, 2, or 3 cycles. It supports two banks open simultaneously, allowing parallel transactions to manage the state associated with them. This means that transactions can go to either bank without getting serialized. The SDRAM interface uses a burst length of 2, so it is optimized for transactions of 64 bits and larger. The interface supports operations less than 32 bits wide through manipulation of the DQM signals.

3.3.2 PCI Bus Interface

The PCI bus interface meets the 2.1 PCI specification. It implements a 32-bit address/data bus, and operates up to 33 MHz in the initial release. It provides target and initiator cycles and zero wait-state burst mode. It is not normally intended to be the system master, since it does not include an arbiter. In accordance with the PCI spec, the PCI bus provides the clock, so there is a synchronizer between this interface and the rest of the MECA-4x chip, specifically at the interface to the on-chip bus module.

3.3.3 TDM Interface

The TDM interface provides multiple clock and data interfaces, each asynchronous to the internal operation of the MECA-4x, with associated DMA engines, and optional framing support. Input and output interfaces are treated as separate and independent entities. The individual interface units are each 8 bits

wide, but groups of 8-bit wide interfaces can be combined to form 16-bit and 32-bit wide interfaces. The TDM interface may be configured with 1–4 input interfaces and 1–4 output interfaces as long as the total parallel data width in each direction is 32 bits or less.

The interfaces are designed to operate asynchronously to the MECA-4x internal clock, so the TDM interfaces use external strobe inputs to receive and transmit data. These strobes may be derived from sources synchronous to MECA-4x internal clocks, so long as they operate no faster than the internal SBCLK.

The TDM is designed to interface to PHY chips for many different datacom and telecom applications. As a consequence:

- Setup and hold times are less than or equal to 2 ns on the receive side
- Output data is pipelined through an output register
- Delay time from clock to new-data may be configured individually for each output so that a range of setup and hold requirements can be met

Framing pins associated with each 8-bit TDM interface allow them to respond to external framing signals on the receive side and to generate them on the transmit side. Frame timing signals are configurable.

In addition, specialized hardware support is provided to dynamically drive output enable signals on the output data lines. This can used to create firmware-based support for TDM busses such as H.100.

Two of the 8-bit TDM interfaces can be configured to act as a Utopia Level 2 interface with polling support. Using TDM interfaces in this mode requires some of the GPIO pins to be used for this function.

3.3.3.1 TDM DMA

DMA engines are associated with each of the TDM interfaces. They allow data to be collected and sent either to memory buffers or to one of the MDP cores. The state machines associated with these DMA engines provide built-in support for routing fixed-length packets of information to different units of the chip in fixed patterns. For example, they can easily route successive data packets to the four MDP cores, drop every third packet, and so forth. These state machines can be synchronized to the framing signal associated with the corresponding TDM interface.

3.3.4 General Purpose I/O Pins

Sixteen General Purpose I/O pins (GPIO) are configurable as either input or output. Outputs can be set individually or in groups. Inputs may generate interrupts. These pins are configured through control registers associated with the TDM module; see Section 5.5, TDM and Section 5.7, GPIO Pins for details.

3.3.5 PLL

The on-chip PLL creates the internal clock SBCLK by multiplying the input clock signal SCLK by 1/2, 2, or 4. It can also be bypassed, which causes SCLK to drive SBCLK directly with a buffer delay. Except in bypass mode, the PLL cannot guarantee any particular phase relationship between SCLK and SBCLK. The PLL is controlled by two signal pins PLLMODE0 and PLLMODE1. Details are found in Section 5.10, PLL.

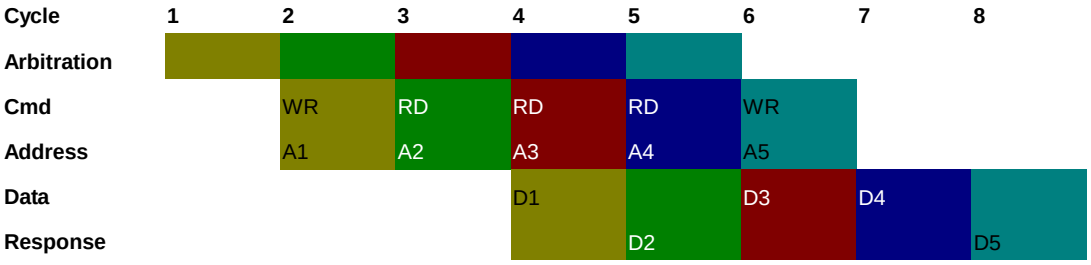
3.4 Inter-Block Communication

3.4.1 On-chip Bus

The on-chip bus provides 64 bits of data and 32 bits of address. It is a pipelined bus operating synchronously with the main chip clock, SBCLK. At the maximum clock rate of 100 MHz, this bus provides 800 MB/s of bandwidth. As with the PCI bus, units connected via the Sonics Backplane are described as *initiators* (that is, bus masters) and *targets* (bus slaves).

A key feature of the bus is that it provides guaranteed bandwidth for any desired initiators. This makes deterministic real-time behavior possible, even in the presence of multiple flows. The bus also includes a fast, fair arbitration mechanism for cycles that aren't needed by the guaranteed bandwidth protocol, thus enabling applications to approach very close to full theoretical bus bandwidth in practice.

The following diagram illustrates bus pipelining. In the diagram, there are eight cycles, containing five complete bus transactions, each shown in its own color. Each transaction takes 4 cycles: one for arbitration, then a command phase and a data/response phase. The command phase follows the arbitration phase by one cycle, then the data/response phase follows by 2 cycles. In the case of writes, the data is supplied on the data/response phase rather than simultaneously with the command and address. This means the write target must buffer the address, but it allows read and write transactions to be interleaved without the turnaround overhead found in most buses.



Arbitration is done by combining a “distributed TDMA” algorithm with a variant of round robin, as follows. Each initiator is allocated a certain set of guaranteed cycles within a bus transaction frame of configurable length (typically 256). Each initiator contains a few bits of local memory that indicate which cycles it owns—so there is no need for a central authority to communicate cycle authorizations dynamically to all possible initiators. Guaranteed cycles are subject to a “use it or lose it” policy: round robin arbitration is performed both for bus cycles that are not claimed by their owner and bus cycles that are not allocated to any initiator.

3.4.2 DMA Controllers

The MECA-4x includes two separate 4-channel DMA Controllers: one optimized for SDRAM transactions, the other for general-purpose DMA between on-chip units.

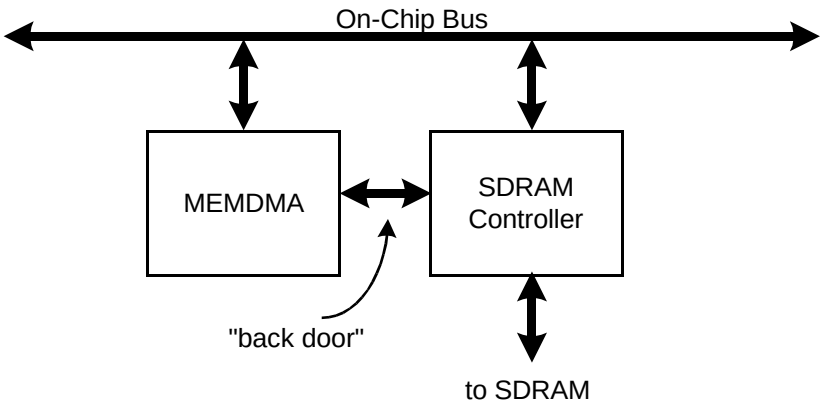
The DMA Controllers are designed for a flexible range of requirements with a minimum of overhead in memory for storing descriptors and in memory plus bus bandwidth for fetching them. The lowered overhead facilitates shorter transfers, thereby simplifying system designs.

3.4.2.1 SDRAM Memory DMA (MEMDMA)

The SDRAM Interface has an associated 4-channel DMA controller (MEMDMA). Either the source or the destination of MEMDMA transfers must be SDRAM.

The implementation of the MEMDMA uses half as many bus cycles as a naïve DMA design to transfer data because it directly accesses the SDRAM memory controller through a back door interface shown in Figure 8. Ordinarily, when transferring data into memory, a DMA controller would read some data from its source and write the data into the memory through the SDRAM Controller interface. The read would take one bus cycle, transferring data from the source to the DMA Controller. Then another bus cycle would be required to transfer data from the DMA Controller to the SDRAM Controller. The MEMDMA saves this second cycle by interfacing directly to the SDRAM Controller through a back door, effectively cutting the bus bandwidth for this type of transfer in half.

Figure 8 MEMDMA Back door to SDRAM



The MEMDMA controller arbitrates for access to the on-chip bus and SDRAM with other components attached to the on-chip bus. Arbitration is based on a round-robin scheme.

The MEMDMA controller includes all the features found in high-end DMA engines today, including:

- Source and destination addresses both optionally incrementing or not
- Performs chained commands without software intervention
- Optional software notification anywhere within a chain of commands
- Command chaining through sequential or linked list
- Overlaps processing of next command with current data
- Arbitrary settable burst lengths
- Extensive control of arbitration and priority policy

The MEMDMA controller has 5 control registers per channel, as shown in Table 3.

Table 3 MEMDMA Controller Registers

Register	Width	Description
Channel Command	32	Transfer direction, length, etc.
Channel Control	32	Burst control, priority, repeat counter, etc.
Target 1 Address	32	
Target 2 Address	32	
Pointer Address	32	Link to next channel descriptor (used only in linked mode)

3.4.2.2 Generic DMA Controller

In addition to the MEMDMA, there is another generic 4-channel DMA (GENDMA) that is used for DMA flows among PCI devices and the MDP cores, including from one MDP core to another. This DMA engine is functionally identical to the SDRAM DMA and is programmed in exactly the same way. Details are in Section 5.4, DMA Controllers.

In addition to MEMDMA and GENDMA, the TDM Interface includes as a set of DMA controllers that are more specialized than the ones described in this section. See Section 5.5, TDM.

3.5 Address Spaces

Table 4 shows the chip address spaces available in the MECA-4x.

Table 4 Address Map

Interface	Description	Start Addr	End Addr
SDRAM	1 GB, memories mapped starting at 0	00000000	3FFFFFFF
MDP0		40000000	400FFFFF
MDP1		40100000	401FFFFF
MDP2		40200000	402FFFFF
MDP3		40300000	403FFFFF
TDM	TDM configuration	41100000	411007FF
MEMDMA	DMA to/from SDRAM	41200000	412007FF
SDRAMCFG	Configuration of SDRAM Controller	412F0000	412F07FF
GENDMA	Generic DMA	41300000	413007FF

Interface	Description	Start Addr	End Addr
PCI Config	PCI Configuration Space	C0000000	C000FFFF
PCIIO	MECA-4x initiated PCI I/O Space	D0000000	DFFFFFFF
PCIMEM	MECA-4x initiated PCI Memory Space	E0000000	EFFFFFFF

Portions of the 32-bit address space that are outside any of the identified ranges cause a bus error on any access.

3.6 Combining MDPs and I/O

The combination of the MECA-4x I/O components with the powerful logic and bit-manipulation capabilities of the MDPs provides for efficient yet highly flexible data transformation. These capabilities may be combined, for example, to transform a byte-wide input consisting of 8 bit-serial streams into 8 distinct data buffers, or vice versa. For framing acquisition, an MDP can be programmed to recognize the frame timing, so that dedicated hardware is not needed for this task. Or, perhaps only a subset of the normal functionality of a Utopia interface is provided in hardware, with the rest being provided in firmware and in the cell buffering capabilities of the hardware-supported FIFOs in the MDP.

4 VOX SYSTEMS ARCHITECTURE

This chapter describes at a high level how the MECA-4x chip is used in systems to implement high channel-count Voice over Packet applications. The architectural concepts described here allow feature enhancements and variations to be implemented efficiently, and can be extended to applications involving multiple speech vocoder algorithms, different types of network interfaces, etc. This chapter does not provide enough detail for engineers and systems designers to work with the applications firmware and host driver software supplied with the MECA-4x, but does provide an overview of these matters. More complete information on applications firmware and host driver software can be found in these and related documents:

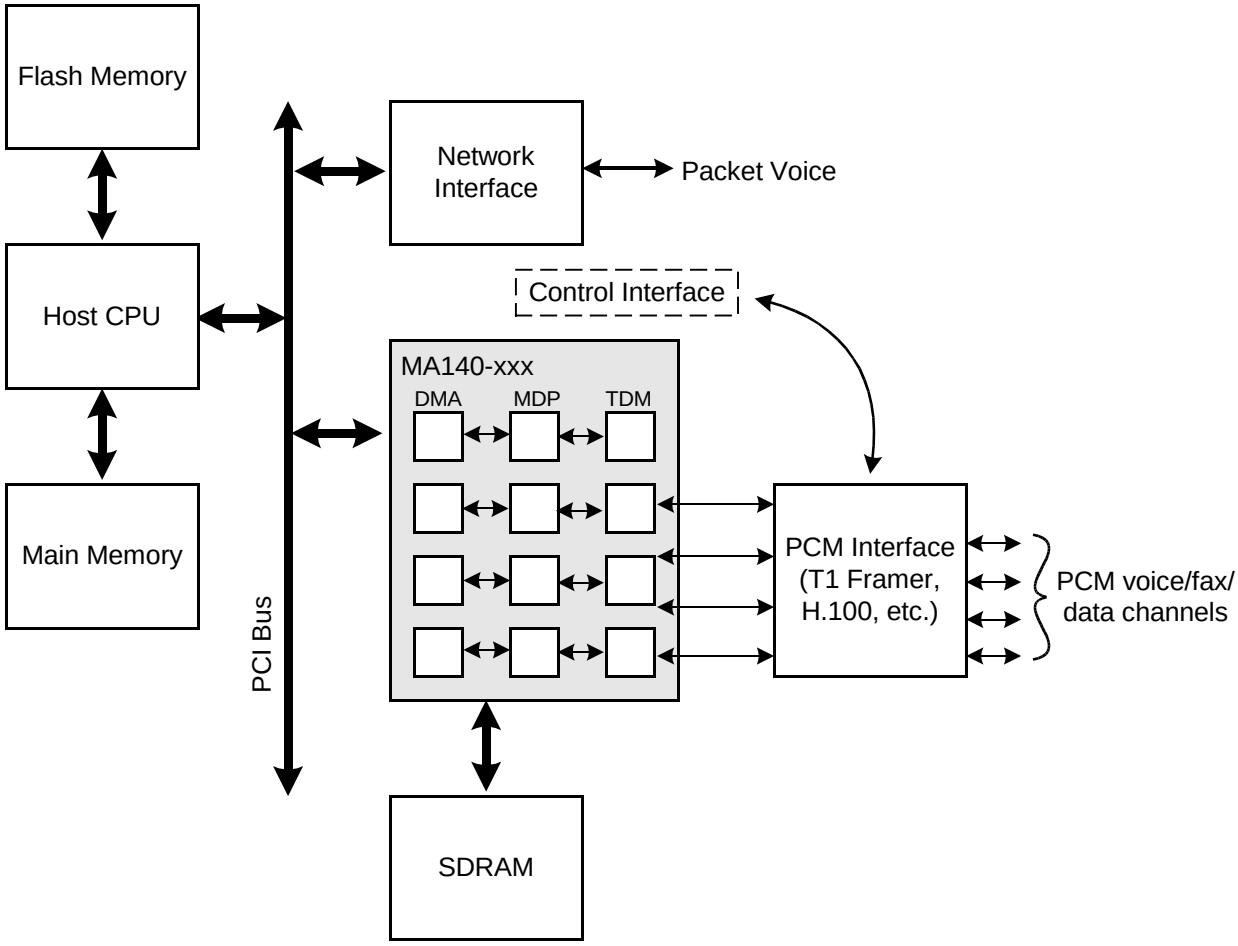
- EasyStream API User Guide
- VoX Driver Software User Guide

Please see this Related Documentation.

4.1 System Hardware Elements

Because of its high channel count capabilities for voice processing, the MECA-4x chip is ideally suited for central office and enterprise gateways and routers, IADs, and similar types of equipment. To handle the control and signalling complexity of systems with many voice channels, these products normally use 32-bit or even 64-bit CPU controllers with significant cache and memory subsystems, and large software images which require quite sophisticated operating systems. Although there is absolutely no requirement to embed the MECA-4x in products of this complexity, for the sake of discussion in this chapter it is assumed that this is what is being done. Figure 9 shows a block diagram of the components in such a system, assuming a single MECA-4x.

Figure 9 VoX System Block Diagram



This diagram shows in block form how the four MDPs in the MECA-4x are all used in the same way to process different channels. In this approach, the MDPs are being used for the simplest possible form of parallel processing. The MECA-4x is designed so that the four MDPs can be easily used in this fashion to increase the number of channels processed. However, because of the programmable nature of the MDPs, distribution of processing tasks is actually arbitrary.

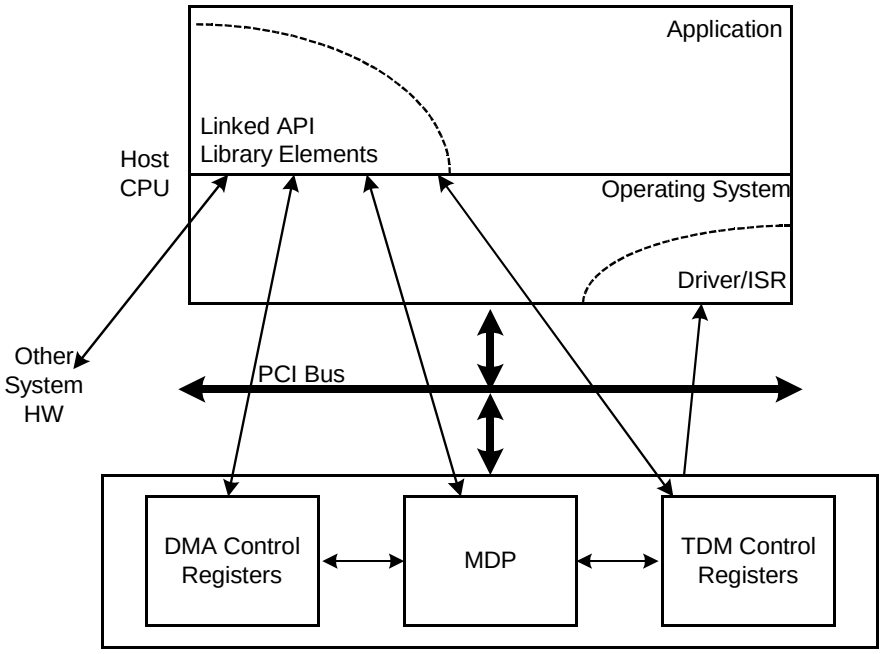
At a generic level, this diagram is quite similar to existing products. The MECA-4x increases the channel density due to its processing power and inclusion of functions (such as PCI interface and timeslot interchange capabilities) that may currently be in separate chips. The MECA-4x also increases the flexibility for the designer, since there are fewer dedicated-purpose chips on the system board.

The MECA-4x replaces multiple DSP chips, and typically will also replace a certain number of “glue logic” chips that would be required to perform time-slot interchange, packetization, PCI bridging, and similar tasks in other architectures. Even so, there are other chips required in the target application being considered here, such as the chips associated with the PCM interface, and these must be controlled by the host. This control interface is not shown in detail in Figure 9 since it can be done in several different ways depending on the details of the PCM interface chips.

4.2 System Software Elements

The generic similarity to existing products extends to the software structure. Figure 10 shows a block diagram of the software components of the system, concentrating on the parts that interact with the MECA-4x. The application software typically contains the core of the system vendor’s expertise, which has been integrated and tested with the particular set of hardware elements found in the product. The application includes elements linked from PMC-Sierra’s EasyStream API software, which provides abstract ways of controlling the elements within the MECA-4x chip.

Figure 10 VoX Software Structure Diagram



The operating system running on the host CPU includes a driver capable of handling interrupts generated by the MECA-4x and generating events from them to pass along to the application level. In addition, Figure 10 shows lines connecting the application software to components within the MECA-4x through the EasyStream API library software. These lines represent interactions with control and status registers for the DMAs and the TDM interface. Other on-chip control registers will be set up at initialization time and normally will not be changed thereafter. These are not shown in Figure 10, but they include PCI configuration registers, SDRAM setup registers, etc.

Updating and controlling the MECA-4x subsystem is efficient and simple because the registers of the MECA-4x system are memory-mapped directly into the address space of the host. Depending upon the implementation of the operating system, memory-mapped addresses may be read and written directly by the sample application. Alternately, access to registers may be provided by traps to kernel driver routines which touch hardware on behalf of the application.

Figure 10 also shows that each MDP can interact directly with DMA and TDM control registers. This is done through an external memory interface called the Master Port. The Master Port is defined in the MDP architecture and accessible through MDP firmware. The MDP firmware itself is not shown in the diagram, but it includes code for all the DSP algorithms running on the platform plus the firmware implementing other logic functions (such as serial-parallel translation), and the firmware to provide top level control of timing and coordination of the DSP code running under the ultimate control of the host software.

To summarize Figure 10, we see that the elements of the system software include:

- Application software
- API software linked into Application software
- Operating system software
- Drivers linked into the Operating system software for all specialized hardware, including MECA-4x
- Firmware binary images for each MDP

The rest of this chapter discusses EasyStream API software, drivers, and firmware, the software elements supplied by PMC-Sierra. Application software and operating systems are not discussed, since these are in the domain of the system designer. Relevant aspects of the design philosophy for the MDP

firmware and system software are described before the sections that cover PMC-Sierra-supplied software and firmware.

4.2.1 Design Philosophy

This section describes the principal design aims of the MECA-4x. A clear understanding of the design philosophy helps designers in knowing the best ways to use the chip.

4.2.1.1 DMA-Centric, Register-Resident

The MDP cores that provide all DSP functionality for the MECA-4x are designed as “inner loop accelerators”. This means they are designed to be efficient on relatively small programs that are executed at a high rate. The algorithms targetted for the MDPs include classic DSP algorithms for speech of a complexity up to and including low-rate vocoders, and other telephony-related algorithms including bit and byte interleaving and deinterleaving associated with multiplexed PCM voice streams (T1, E1, and similar). The MDPs are also efficient at a certain number of algorithms not classically associated with telephony, including packet-oriented networking algorithms, forward error correction algorithms, cryptographic scrambling protocols, etc.

The MDPs work best in situations where the ratio of computation to I/O is fairly high—that is, where there is quite a lot of computation for each unit of I/O. One of the ways the MDP gains its efficiency is by loading a significant amount of data into its large register files and then processing it without accessing external memory under program (firmware) control. When operating in this mode, it is essentially doing no loads and stores. Optimizing loads and stores in classic microprocessor and DSP architectures consumes considerable silicon area and design complexity, so the effect of essentially removing this design element can be significant.

Though the architecture de-emphasizes use of load and store operations from/to external memory in the normal operating mode of the MDP firmware, some mechanism must be provided to move data between the MDP register files and other elements of the chip, including off-chip memories and I/O’s. In the MECA-4x, this is a set of DMA engines. There are many DMA engines on the MECA-4x, with a good deal of flexibility to juggle bandwidth, control of timing, etc. Most of the DMA engines are special-purpose in the sense that they are closely associated with some component of the chip: either the TDM interface (where the DMA engines are tightly integrated into the interface itself) or the SDRAM interface.

4.2.1.2 Block-Oriented Processing

A further side effect of the DMA-centric philosophy is that the MDPs process data in a block-oriented fashion. In many applications, this is very natural because the algorithms used in the applications are inherently block-oriented. For example, all LPC-based speech vocoders are this way: G729 uses 80-sample frames, G723.1 uses 240-sample frames, and so on.

However, not all of the widely used speech processing and telephony algorithms are inherently block-oriented. G726, DTMF detection, G.711 conversion, and most line echo cancellation algorithms are examples of algorithms that are not inherently block oriented. For these, the appropriate block size is determined by trading off latency for computational efficiency. In some cases, block size is a parameter of the firmware module, in others, specific choices are represented by part number variations.

Block processing enables DMA commands to be set up in advance to automate the work of moving data in and out of MDP register files, either from/to SDRAM memory, memory in PCI space, or the TDM interface. Sometimes a chain of DMA commands will be started under control of the Host CPU; other times it will be started by the affected MDP. In most cases, things are organized so that DMA command execution is overlapped with “useful” computation.

4.2.1.3 Interrupts to the Host

We have already described how the MDP is simpler than classic microprocessors in its interaction with main memory. Another significant simplification is that MDPs cannot receive and process interrupts. This means they do not need logic for recognizing interrupts, saving and restoring context at abitrary points in their program execution, etc. The rationale for this simplification is that interrupt handling is the hallmark of a control processor, giving a high degree of flexibility to respond to external events, whereas the MDP is designed to be primarily a data processor. Another way to think of it is this: an MDP should play approximately the same role as an ASIC in a system (albeit a programmable one): an ASIC hardware block will never receive and process interrupts, but it may well generate interrupts to be handled by the controller (host CPU).

Although the MDPs cannot be programmed as full scale system controllers capable of handling interrupts and the like, there are ways to get MDPs to react to external events. Both the run/stop state of the MDP and its program counter can be manipulated directly by the Host CPU. Also, an MDP can be programmed through firmware to poll for a value (or value change, etc.) in a predefined register within its register file. Since it is designed to handle repetitive tasks of low to moderate control complexity, its outermost loop will normally have

predictable timing and occur at a fairly high rate (such as 100 Hz or more). In this case, its polled response to external events will be as good as the response time that is typically required in system designs that employ embedded controllers (although certainly not as good as the controller vendor's claims for their own interrupt response time).

4.2.1.4 Moderate Latency Requirements to the Host

Although host controller clock speeds are increasing and real-time operating system vendors claim "fast" interrupt response times, the fact is that designing a system around aggressive interrupt latency requirements and/or high interrupt rates can have the effect of sapping the performance of the host controller, resulting in a long and frustrating debug cycle. Though attainable interrupt latency requirements and interrupt rates are partly determined by global system design considerations, they are also determined by design details of the chips that the host controller controls.

The design of the MECA-4x has been influenced by the desire to keep interrupt latency requirements moderate. In other words, once the API software has set up the MECA-4x, the host should be able to interact with it at a low rate. Changes in the control and signalling environment may temporarily require a higher rate. In voice and telephony applications, a good target for the normal interaction rate might be once per block, on the order of 30 to 250 Hz.

4.2.1.5 Simplified Multiprocessor Coordination

Since there are multiple MDPs in a MECA-4x, an appropriate multiprocessor synchronization methodology is required. Inappropriate use of the Master Port feature for firmware-directed control of DMA and TDM control registers can lead to certain classic multiprocessor synchronization problems. For example, if two or more MDPs write to specific DMA control registers in an uncoordinated manner, the actual owner of the resources controlled by these registers can become unpredictable.

A common general-purpose mechanism for solving these problems is to provide for atomic transactions and use them to create semaphore data structures that employ operating system support to create critical sections. Although semaphores and critical sections are very likely to be available and appropriate as part of the host operating system and host application, they are overly general and expensive within the MDPs. Instead, a variety of more *ad hoc* mechanisms are designed in the MECA-4x. For example, there are four general-purpose DMA channels and four SDRAM-specific DMA channels. Each can be dedicated by software convention to a separate MDP. In this case there are no unresolved

coordination issues for DMA usage, since the arbitration hardware within the DMA engines takes care of it.

The advantages of this design approach include the following:

- Hardware support for atomic transactions is relatively expensive in performance and silicon area.

- Semaphores, or other mechanisms built on atomic transactions, must be implemented in software and integrated with a full operating system kernel to work well. The system level cost of the general-purpose solution is high.

- Appropriate use of MECA-4x resources allows the multiprocessor coordination problem to be *avoided* in most cases, with a considerable savings in system complexity for the system's customer, in addition to the savings in silicon area.

4.2.2 MDP Firmware

Firmware is supplied by PMC-Sierra in the form of a binary load image, a database of firmware module information, and documentation. Each firmware image is associated with a different chip part number or part number variant.

- The binary load image is a list of instructions to a firmware loader to load bit patterns into the various instruction and micro-instruction memories of an MDP. The firmware loader is included as part of the EasyStream API software.

- The module's information database is for use by the API software; see Section 4.2.4, EasyStream API Software for details.

- The documentation is a set of electronic documents and includes functional description (what the firmware does), information about initialization and configuration of the firmware, and information about the locations and function of control and status registers associated with the real-time operation of the firmware.

Information about configuration includes what configurable parameters are available, what they do, limits on their ranges, interactions among settings of configurable parameters (such as tradeoffs between echo tail length in a line echo canceller module and number of active vocoder channels), and information on how to read and write them through the EasyStream API software. Some of this information is derived directly from corresponding information about one of the modules that makes up the binary load image. Some of it concerns the

interaction of modules, so is particular to a specific binary load image. Finally, some of it is derived from the process of collecting and linking together all the modules making up a binary load image.

Internally, MDP firmware is developed by linking together a collection of *modules*. Each module provides a defined functionality, such as a specific vocoder (compression) algorithm, line echo canceller, signal monitoring or generation function, etc. Modules share common data structures and usage conventions so they can be easily connected together. Please see Section 4.4, Modules for an overview of some common MDP firmware modules.

To create a binary load image from a set of modules requires the following elements:

- Top level code to initiate DMA commands and call the modules
- Mapping information that allows the EasyStream API software to access control and status registers defined by the modules
- Initialization information that tells the API software how to set up the MDP and SDRAM environments with properly sized data buffers, state information records, etc.

The binary load image consists of the instructions and register file constants associated with each module along with the top-level glue code and auxiliary mapping and initialization information.

Module-based firmware architecture has several benefits:

- In-house development of firmware for initial customers is simplified because images for different applications can be created by combining existing modules.
- It provides organization for firmware development effort, since modules can be developed and tested separately.
- It provides a long term path to opening up the development environment in a structured way.

Since the overall firmware architecture of the MECA-4x is block-oriented (as discussed in Section 4.2.1.2, Block-Oriented Processing), modules operate on blocks of data. For some modules, the block size is fixed, while for others it is a parameter to the module.

In addition, the modules that are used in firmware load images will generally be *multi-channel*. This means they process all the blocks for a set of channels each time they are called. This approach amortizes the cost of loading any necessary tables over all the channels. In multi-channel processing, the maximum number of channels is built into the binary load image and associated API software, and a convention is established to allow channels to be made active or inactive by the host CPU through the API software. In most cases, channel activity status can be controlled on a per-module basis, so that individual processing steps can be enabled or disabled for each channel individually.

At the system level, processing efficiency requires that DMA transfers overlap with module computations. Some DMA transfers are initiated by the top-level "glue" code that calls the modules. Other DMA transfers are initiated by code within modules to read in or write back data and state information on a per-channel basis. Correct operation of overlapped DMA transfers generally requires double buffering, so that one buffer can be owned by the computational process and the other by the DMA engine. Coordination of glue-code DMA commands with intra-module DMA commands can be accomplished with features of the DMA controllers, in particular their ability to handle linked lists for command chaining. These lists are created in SDRAM memory through EasyStream API calls.

4.2.3 Device Driver

Device Driver software is supplied by PMC-Sierra in the form of source code and related software development support files for the Windows NT® operating system (with support for VxWorks® to follow). The purpose of the Device Driver is to receive interrupts from a single MECA-4x chip and translate them into an event that can be handled by the application software. Typically, handling the interrupt will involve reading one or more registers internal to the MECA-4x so that its state at the time of the interrupt is properly captured and passed to the user-level software. It may also involve clearing the interrupt. For systems using multiple MECA-4x chips, external hardware arrangements must be made to combine interrupt signals from all the possible interrupt sources, and driver source code will have to be modified accordingly by the customer.

This approach keeps the responsibilities of the Device Driver at a minimum, which reduces the design and debug time spent inside the operating system kernel.

4.2.4 EasyStream API Software

The EasyStream API software is a set of data structure definitions and functions written in the C programming language that provide access to the facilities of the

MECA-4x chip and of the firmware binary load image that is in use. The EasyStream API software assumes that the MECA-4x memory space can be direct-mapped into the application, and that interrupts generated by the MECA-4x chip are received by a simple operating system driver and passed on to the application software.

The EasyStream API software deliverables consist of the following items:

- A set of C language header files for data structure definitions and declarations
- Linkable object files for the callable functions
- C source code which demonstrates generic functionality for certain application-specific behaviors (such as response to FIFO overflows or other hardware errors)
- Documentation

Some of the EasyStream API functions are associated with the chip itself, while others are associated with, and/or parameterized by, the particular firmware binary load image. Examples of chip-level API functions include:

- Hardware initialization
- Device driver initialization
- Firmware download and initialization
- Error and event handling
- Generic functions to control TDM interface
- Generic functions to handle DMA controllers

Module-specific API functions are actually generic functions that provide module-specific behavior through the module information database provided with the firmware. This module information database consists of an object file and a C language header file. The object file defines various register addresses, array sizes, data structure initializations, and the like. It is used by the module-specific API functions. The include file is used to facilitate writing module-specific application software. Examples of module-specific API functions include:

- Channel activation and deactivation

- Parameter reading and writing, including statistics gathering and diagnostics monitoring

- Module-specific initialization calls

The application software calls both chip-level and module-specific API functions in order to create a complete application using the MECA-4x.

4.3 Example Application

In order to describe some of the concepts introduced in this chapter in a more concrete fashion, this section covers a simplified application showing how the hardware, firmware, and software elements work together.

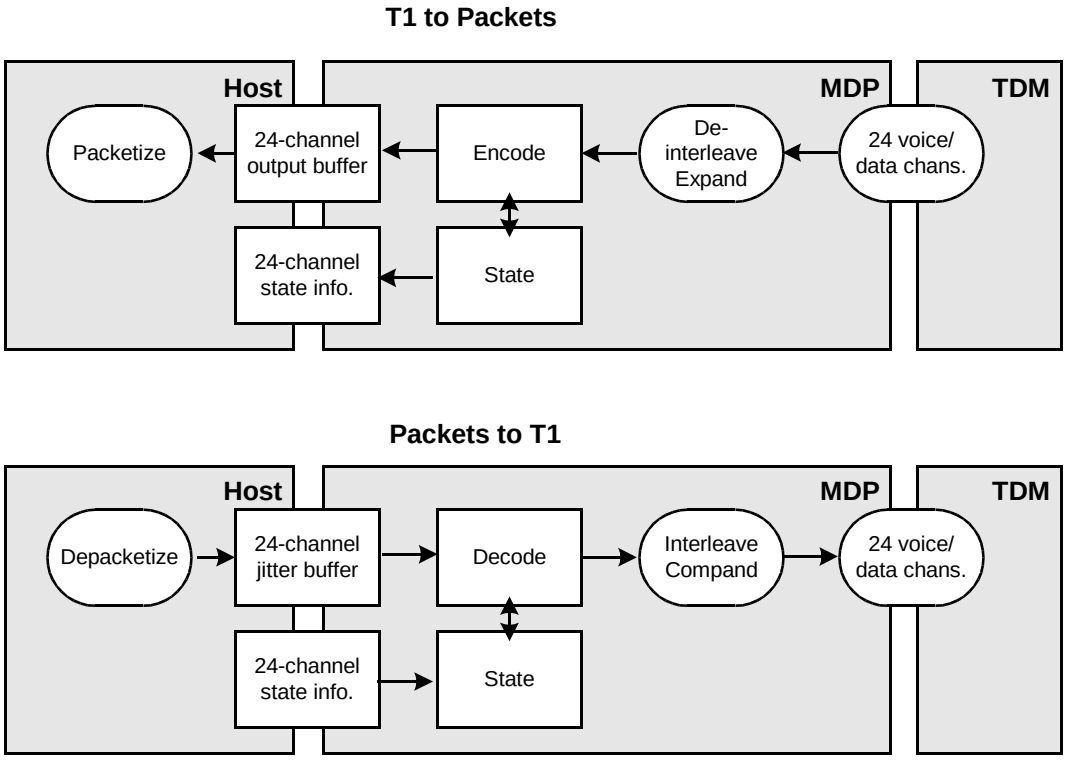
4.3.1 Example Features

In this application, the system hardware is like that shown in Figure 9: a single MECA-4x connected to a single CPU through the PCI bus. In this example, the TDM interfaces are connected to four T1 framer/LIU chips, and each MDP works completely independently to process the data for a single T1. The processing done by the MDP includes packing and unpacking of T1 data, and compression and decompression by the G.729A algorithm. The host application does call processing, packetization, network interface, and jitter buffer management. For simplicity, we assume signalling information is completely outside of the MECA-4x; for example, it may be extracted from the T1 framers and passed directly to the Host CPU through some unspecified mechanism.

Features missing from this example that might be required in a real application include echo cancellation, detection and re-routing of fax and data calls, VAD/CNG per G.729B, DTMF detection, and DTMF generation. These features add to the complexity of the example but would not change the organization of either the firmware, API, or driver; thus, they are not covered here. In addition, some system designs may partition the processing differently, for example, by moving some packetization and/or jitter buffer management into the MDPs.

A block diagram of the processing involved in this example is shown in Figure 11.

Figure 11 Per-MDP Processing for the Example Application



The implementation of this example is described in sub-sections covering each of the different parts of the hardware, firmware, and host software.

4.3.2 Implementation Overview

The MECA-4x and associated firmware and software deliverables are designed to simplify implementation of this and other applications. The system designer is, of course, responsible for system architecture and board design. The following paragraphs summarize how the application works and how PMC-Sierra's deliverables participate in each step:

Hook it up

The MECA-4x hooks gluelessly to many industry-standard components used in telephony applications. For host CPU support, the PCI bus provides an interface that is widely available and has reasonable bandwidth and latency. In the example application given here, the system designer must specify the network interface in addition to the components shown in Figure 9.

Power it up

When power is applied, all hardware components are initialized. The MECA-4x comes out of reset in idle mode. The host CPU boots, and its operating system launches the application. One of the application's initialization steps is to call EasyStream API functions from PMC-Sierra that initialize MECA-4x hardware.

Host processor downloads firmware

Under application control, the host processor selects the appropriate MECA-4x firmware and downloads it to the MDPs. The MDPs are left in an idle state.

Host processor runs initialization code

The host initializes parameters of the firmware application, performs any needed synchronization, and starts the MDPs. The MDPs are now active, but unless some channels have been activated as part of the initialization sequence, nothing very interesting is happening yet.

Host processor goes into steady state loop

At this point the host processor awaits events that cause it to activate or deactivate channels, events that signal arrival of network data, events that signal block processing completion on the MDPs, etc. The host processor's response to these events will cause channels to become active, cause the next data block(s) to be processed, and so on. In this mode, the host processor will also be maintaining statistics registers and the like, and responding to requests from external management entities as they occur.

Updating and controlling the MECA-4x subsystem is efficient and simple because the registers of the MECA-4x system are memory-mapped directly into the address space of the host. Depending upon the implementation of the operating system, memory-mapped addresses may be read and written directly by the sample application. Alternately, access to registers may be provided by traps to kernel driver routines which touch hardware on behalf of the application.

4.3.3 Data Flow

Voice processing on the MECA-4x is block-oriented, as was discussed in Section 4.2.1.2, Block-Oriented Processing. In this example, the block length is 80 samples, corresponding to 10 ms, which is the block length of the G.729A algorithm.

For the T1-Packet flow shown in the top part of Figure 11, TDM data is sent to SDRAM memory, DMA'ed into the MDP as a serial bitstream, deinterleaved, expanded per G.711 from 8 to 16 bits per sample, and then written back into 24 channel buffers in 80-sample blocks. These blocks are concatenated into 240-sample blocks which are processed per channel by the G.729A encoder. The G.729A encoder module is called once per frame (10 ms frames). The encode process involves a single DMA command that loads tables and other constants, then 3 DMA commands per active channel, 2 to load state and current data for that channel, and 1 to write back updated state. The per-channel compressed data is written back to SDRAM by the firmware directly, instead of through the DMA, since there is so little data. The compressed data is written to buffers that are directly available to the host software, under the assumption that the host will copy each 10-byte compressed packet into its local cache, add network headers to it there, and copy it to the network interface, which is assumed to be on another PCI card in this example.

For the Packet-T1 flow shown in the bottom part of Figure 11, compressed data arrives from the network interface and is placed by the host in the incoming compressed data buffer in the MECA-4x SDRAM space for the appropriate channel, based on information in the packet header. The G.729A decode process is similar to the encode process. Specifically, the G.729A decoder module is called once per frame (10 ms frames), and involves a single DMA command that loads tables and other constants, then 3 DMA commands per active channel. In the decode case, 1 DMA command is to load state, and 2 are write backs, one for updated state and one for decompressed data. The compressed data is read directly by the MDP firmware. After all the channels have been decoded, the decompressed data for all channels (including dummy data for inactive channels) is DMA'ed into the MDP register file space and the T1 interleave firmware module is called. It interleaves the 24 channels, bit-serializes them, and initiates a DMA command that writes the results back to SDRAM again.

Both input and output TASGs of the TDM interface are running constantly to receive and transmit bit-serial T1 data. Accordingly, double buffering is used. The SDRAM buffers are each large enough to hold one block (10 ms) worth of data, which is 24 channels by 8 bits/sample by 80 samples. Since the T1 data is packed one bit per byte, there are a total of 15360 bytes per buffer. Since these buffers are rather large, interleaving and deinterleaving are done in several stages, with DMA commands to read and write data being run from within the interleave and deinterleave modules.

4.3.4 TDM Hardware Setup

The TDM interface hardware is configured as 4 independent byte-wide interfaces. The received data output of the T1 LIU/framer chips is connected to

pins TDIN0, TDIN8, TDIN16, and TDIN24. All other TDIN pins should be tied low. The received clock output of the T1 LIU/framer chips is connected to pins TDINSTRB0, TDINSTRB1, TDINSTRB2, and TDINSTRB3. If the T1 LIU/framer chips are configured for gapped clock, the MECA-4x input framing pins should be tied high: TDINFRM0, TDINFRM1, TDINFRM2, and TDINFRM3; otherwise the framing signal outputs of the corresponding chips should be tied to these pins.

On the transmit side, the transmit data input of the T1 LIU/framer chips is connected to pins TDOUT0, TDOUT8, TDOUT16, and TDOUT24. All other TDOUT pins should be left floating. The received data clock should be used to clock transmit data out of the MECA-4x by connecting received clock output of the T1 LIU/framer chips to pins TDOUTSTRB0, TDOUTSTRB1, TDOUTSTRB2, and TDOUTSTRB3. If a framing signal is used, TDOUTFRM0, TDOUTFRM1, TDOUTFRM2, and TDOUTFRM3 should be connected to the transmit framing signal pin of the appropriate LIU/framer chip; otherwise, the configuration of these pins doesn't matter.

4.3.5 TDM Interface Configuration

The TDM Interface is configured through EasyStream API calls that set the hardware to be 4 independent byte-wide interfaces and then initialize the hardware. An API call is made to allocate the double buffers for the TDM input and output, and to associate these with the firmware and the DMA address ranges in the TDM interface hardware. The TASG sequencing logic is initialized through another API call. Then an API call is made to cause each byte-wide interface to synchronize to the T1 frames, so that the first bit of each frame is aligned with the start of the buffer. The host code makes use of special firmware in the MDP as well as the synchronized start hardware in the TDM interface, as described in 5.5.3.3, TASG Synchronized Start Logic.

Once frame-synchronized, the TDM interfaces are in Active state, so data is streaming into the double buffers. These have been arranged so that an interrupt is caused at each buffer boundary (i.e. at a 100 Hz rate), but these interrupts are not enabled until the voice processing is actually running.

4.3.6 MDP Initialization

The firmware binary load image associated with this application is loaded through an API call. The MDP is left in an idle state, with all queue managers cleared and disabled. This particular application uses DMA extensively, both the MEMDMA engine and the TDM DMA engines, but it does not use the MDP's queue manager hardware.

4.3.7 Firmware Module Initialization

After the firmware has been loaded, API calls are made to allocate buffers and arrays of state records associated with all the firmware modules, to associate specific buffer and state record locations with the firmware, and to initialize state records in module-specific ways. At this point, all the channels for all the modules would typically be inactive.

4.3.8 Running the Application

The timing for this application is driven by the T1 clock. This clock is visible through the timing of TDM buffer interrupts that occur once every 80 frames (or 10 ms). Each time this interrupt occurs, the Host CPU driver generates an event to the application software, which should then start the MDP. (If all 4 T1 clocks are synchronized, the interrupt response task can start all 4 MDPs at this time, otherwise, it must respond to their interrupts individually.) To start the application, the host enables the TDM interrupts; the MDP firmware starts running after the next frame arrives. In this application, there would normally be no active channels at this time, so the incoming frames would be deinterleaved and then ignored, while outgoing T1 channels would be generated from constant data set at initialization time. Application software uses EasyStream API calls to change the activity status of individual channels as calls are set up and completed.

When there are active channels, meaningful data appears as compressed output on the appropriate channel buffers destined for the network, and compressed input arrives from the network and is copied to the appropriate channel buffers for decompression. The host software manages jitter buffering, i.e., decisions about how many frames of data is held back, when frames of data are forwarded, what data is forwarded in case of dropped packets, at what point to generate alarms, etc.

The MDP firmware allows the application software to monitor the available slack time for the firmware, and provides other diagnostic capabilities as well through register locations.

4.4 Modules

The following paragraphs contain a brief description of some of the modules that can be put together to create different firmware binary images.

TE1Interleave

TE1Interleave receives an array of blocks of 16-bit linear PCM samples in a format determined by the settings of control parameters, and produces an output block of 8-bit companded samples interleaved for output through the TDM interface. Parameters determine whether it processes 24 or 32 channels, the companding law to be used, the block length, etc.

TE1Deinterleave

TE1Deinterleave receives a block of bit-interleaved, 8-bit companded samples, and produces an array of blocks of expanded 16-bit linear PCM samples. Parameters determine whether it processes 24 or 32 channels, the companding law to be used, the block length, etc.

TE1InterleaveAlign

TE1InterleaveAlign is used at initialization time and following a loss of synch error to detect frame alignment of a TDM input coming from a T1 or E1 framer.

DTMFGenerate

DTMFGenerate receives channel buffers and a per-channel array of state information. For those channels whose state indicates active generation of DTMF tones, it replaces the contents of the corresponding buffer with the appropriate DTMF signal.

DTMFDetect

DTMFDetect receives channel buffers and a per-channel array of state information. For those channels whose state indicates detection of DTMF tones, it processes the signal in the corresponding channel buffer using and updating state information . If a tone is detected, it sets the Tone and TonePresent slots of the state information, and optionally arranges for an interrupt to be generated. DTMFDetect never changes the sample values in any channel buffer.

FaxModemDetect

FaxDetect receives channel buffers and a per-channel array of state information. For those channels whose state information indicates that fax or

modem tone detection should be performed, it processes the signal in the corresponding channel buffer using and updating the state information; if a tone is detected, it sets the TonePresent slot of the state record, and optionally arranges for an interrupt to be generated. FaxDetect never changes the sample values in any channel buffer.

LECancel

LECancel receives a block of transmit data, a block of (time-aligned) receive data, a block of coefficients, and a control block. It performs echo cancellation on the transmit block, using and continuously updating the coefficients. It performs non-linear processing; it detects double-talk situations and selectively shuts down the echo cancellation process when they occur.

G729AEncode

G729AEncode receives input channel buffers (240 samples per channel), output channel buffers (10 bytes per channel), per-channel output bit array for silence indication, and a per-channel array of state information. It processes enabled channels to produce a 10-byte output or silence indicator into the corresponding output channel buffer, and updates the state information.

G729ADecode

G729ADecode receives an array of input channel buffers (10 bytes per channel), an array of output channel buffers (240 samples per channel), and a per-channel array of state information. For enabled channels, it produces decompressed output from the corresponding input channel combined with information in the state record. It writes the 80-sample output into the corresponding output channel buffer, and updates the state information.

G726Encode

G726Encode receives channel buffer inputs, compressed buffer outputs, and a per-channel array of state information. For those channels whose state records indicate compression, it produces ADPCM compressed data into the corresponding output buffer and updates the state information.

G726Decode

G726Decode receives compressed channel buffer inputs, uncompressed buffer outputs, and a per-channel array of G726Dec state records; for those

channels whose state records indicate decompression, it produces decompressed data according to the ADPCM algorithm into the corresponding output buffer and updates the corresponding state record.

AAL2Receive

AAL2Receive is called once per block of time, for example, at 5 ms intervals. It receives an array of ATM cells, and data structures that describe the VCs and CIDs for active sessions, including buffers to put received data, and information about the state of each such buffer (to support jitter buffering implementations). It unpacks the subframes of each cell that are described by its data structures, ignoring (and optionally incrementing error counters) for subframes that are not so described. It places the resulting data in the output buffers at the appropriate locations.

AAL2Transmit

AAL2Transmit is called once per block of time, for example, at 5 ms intervals. It receives data structures that describe the VCs and CIDs for active sessions, including data produced in the current block of time for each session by whatever computing process is associated with that session. (Different sessions may use different compression schemes, including no compression; the length of data produced is associated with the data for each session.) It produces an array of ATM cells and returns the number of cells produced. An integral number of cells is produced for each VC, with padding only on the last cell associated with each VC.

5 FUNCTIONAL UNITS

5.1 MDP

There are four MDP cores in the MECA-4x, each with its own memory-mapped instruction and data memories. Each core occupies 1048576 bytes (0x100000) of address space altogether. Base addresses for the MDPs are shown in Table 4. The address segment offsets within each MDP address space are shown in Table 5:

Table 5 MDP Address Segments Map

Offset (hex)	Segment	Description
00000	ICVMEM	Main instruction memory
10000	ADP	ADP microcode memory
20000	(unused)	
30000	RTREES	RTREES microcode memory
40000	SDP	SDP microcode memory
50000	SUBSETS	SUBSETS microcode memory
60000	LDPMUX	LDPMUX microcode memory
70000	RFA_PRI	Priority access: A Regfile
80000	RFB_PRI	Priority access: B Regfile
90000	CONFIG	Configuration space (see Table 7)
A0000	CNTRL	CNTRL microcode memory
E0000	RFA	Regular access: A Regfile
E8000	RFB	Regular access: B Regfile
F0000	RFAB	Regular access: A+B Regfiles interleaved
F8000	STRMPORT 0	Stream port 0
FC000	STRMPORT 1	Stream port 1

Register file access segments and stream ports are of interest to applications development. Most of the MDP address segments are relevant only to firmware downloading, and are of interest only for MDP firmware development and debugging; for details see publications describing the MDPs in the MDP User Guide.

5.1.1 MDP Register File Access

As shown in Table 5, there are two access segments for each register file: Priority and Regular. The Priority access segments are normally intended for use by the firmware loader and by debugger software. Access through these segments always takes priority over the MDP firmware, if it is enabled to run, by inserting a pipeline stall. Thus, external access through these addresses slows down MDP firmware and makes its timing less predictable, which is a bad thing for real-time code.

The Regular access segments are intended for data transfers including DMA that are part of the normal flow of data during a computational cycle. Access through these segments normally waits until a cycle in which the MDP firmware is not using the register file port to which the Regular access segments are wired (specifically, this is port 2), then uses (“borrows”) that cycle for the requested read or write access. A FIFO is provided for read and write commands so that in most cases, no time is wasted retrying commands. In addition to the Regular access segments for A and B register files, there is a third segment for A+B, which has the effect of interleaving 64-bit words across both register files. More specifically, a byte address in the three Regular access segments is interpreted according to Table 6.

Table 6 Register File Access Segment Addressing Rules

RFA	Bits 0 and 1 must be zero, bits 2-12 select word in register file A
RFB	Bits 0 and 1 must be zero, bits 2-12 select word in register file B
RFAB	Bits 0 and 1 must be zero, bit 2 selects the register file (0 for A, 1 for B), and bits 3-13 select word in register file

5.1.2 MDP Stream Port Access

In addition, the Stream Ports are single-word addresses that provide access to input and output Queue Managers for the MDPs. These Queue Managers create up to four FIFOs within either or both MDP register files, 2 input FIFOs and 2 output FIFOs. An input FIFO is written into by an external entity (such as a DMA controller, TDM TASG, or across the PCI bus) and the data is placed at the head of the FIFO. An address generator within the MDP reads the data from the tail of the FIFO. Writing data to an input FIFO increments the FIFO head and makes the FIFO fuller. Incrementing the MDP address generator associated with the FIFO increments the FIFO tail and makes the FIFO less full. Output FIFOs work in the converse fashion: and output FIFO is read from by an external entity, with the data coming from the FIFO tail, and the FIFO tail is incremented after each

read. The MDP address generator associated with the output FIFO writes data to the FIFO head and is incremented to signify that the data is on the FIFO, thereby making the FIFO fuller.

FIFO fullness is tracked for all FIFOs and continuously compared to a threshold value that is configurable by the Host CPU through a control register. The result of the comparison is available to the MDP firmware as a branch condition and also as a Yield condition.

FIFOs must be enabled through the appropriate Queue Manager control register before data is written to or read from them. This is required because configurable state within the MDP address generator associated with each FIFO must be in place before FIFO behavior is meaningful. FIFO underflow and overflow conditions are checked and can be made to cause interrupts to the Host CPU. Another potential interrupt cause is attempting to access a disabled FIFO.

Queue Managers accept 32-bit and 64-bit aligned accesses only.

To access input FIFO 0 (IQM0), an external component will normally write to BASE+0xF8000. However, a properly-aligned access anywhere in the range BASE+0xF8000 to BASE+0xF8FFC will address the FIFO, so it is not strictly necessary to use non-incrementing destination addresses in FIFO accesses.

5.1.3 MDP Control Registers

There are 8 MDP control registers at the offsets shown in Table 7:

Table 7 MDP Control Register Addresses

Address (hex)	Description
90800	Thread
90C00	Input Queue Manager 0 Config
90C04	Input Queue Manager 1 Config
90C08	Output Queue Manager 0 Config
90C0C	Output Queue Manager 1 Config
90C10	Master Port Config
90C20	MDP Interrupt CSR
90C24	MDP Interrupt Enables

Details of these control registers are found in the following sub-sections.

5.1.3.1 Queue Manager Configuration Registers

Each of the four Queue Managers has its own configuration register; each is interpreted in the same way. The bits are shown in Table 8:

Table 8 Queue Manager Configuration Bits

Bit	Description
0	Enable (read-write)
1	Empty (read-only)
5:2	A/B Control
8:6	QM Interrupt Status
11:9	QM Interrupt Mask
23:12	Read: fullness; write: threshold
24	Invert comparison
25	Branch condition (read-only)
31:26	Unused

The A/B Control field controls 32-bit mode versus 64-bit mode and, in the 32-bit case, whether the FIFO managed by the Queue Manager is in the A or B register file. There are 3 interrupt sources whose status is in the QM Interrupt Status field: overflow (bit 6), underflow (bit 7), and access-while-disabled (bit 8). Each source is individually maskable with the corresponding bit (9 through 11) in the QM Interrupt Mask field.

5.1.3.2 Thread Configuration Register

The thread configuration register bits are shown in Table 9:

Table 9 Thread Configuration Bits

Bit	Description
11:0	PC (read-write)
12	PC Enabled Flag (read-only)
19:16	Yield Condition (write-only)
20	Enable PC (write-only)
21	Load PC (write-only)
22	Init Thread (write-only)

5.1.3.3 Master Port Configuration Register

The master port configuration register bits are shown in Table 10:

Table 10 Master Port Configuration Bits

Bit	Description
7:0	Timeout
9:8	Width
10	Dynamic Width Flag
12:11	Offset

The Timeout value is an initial value for an LFSR-based counter, this value is preloaded into the LFSR every time a master port transaction starts. The master port abort error occurs when this LFSR reaches 0xFF. Set this field to 0x00 for no timeout. An API function call is available to set this value given a maximum delay in cycles for master port event completion.

The Width field encodes the transaction width. The encoding is the same as for the DMA controllers and is shown in Table 11:

Table 11 Master Port Width Encoding

Value (binary)	Encoded Width (bits)
00	64
01	32
10	16
11	8

A 64-bit transaction is implemented as two 32-bit transactions. In the case of a write, this means that the first transaction of each pair causes no on-chip bus activity, while the second transaction of each pair causes a single 64-bit bus transaction to occur, using the address of the first transaction. In the case of a read, a 64-bit transaction means that two 32-bit values will be returned; the MDP firmware must poll for each of them and retire them in turn to the register file.

If the Dynamic Width Flag is set to 1, the Width field is ignored and the upper two bits of the master port transaction address are used instead, with the same interpretation as for the Width field. In this case, the Offset field is substituted as the upper two address bits for the transaction, otherwise, the Offset field is ignored.

5.1.3.4 MDP Interrupt CSR and Enable Registers

The bits of both the MDP Interrupt CSR and Interrupt Enable registers are interpreted in the same way, as shown in Table 12:

Table 12 MDP Interrupt CSR and Interrupt Enable Bits

Bit	Name	Description
0	IQOv0	IQM0 error
1	IQOv1	IQM1 error
2	OQUn0	OQM0 error
3	OQUn1	OQM1 error
4	ThdYielded	Thread is not enabled
5	MPAbortRd	Master Port read has aborted
6	MPAbortWrt	Master Port write has aborted
7	MPAddrRd	Master Port read address alignment error
8	MPAddrWrt	Master Port write address alignment error

Interrupts are generated when any bit is set to 1 in both the CSR and Enable registers. In other words, the Enable register masks the CSR bits from generating interrupts. Bits 4-8 are cleared by writing the value 1 to the corresponding bit in the CSR. Writing the value 0 to any bit of the CSR register has no effect. Bits 0-3 of the CSR (QM error bits) are cleared by resetting the appropriate Queue Manager, which is done by writing to its control register.

5.2 PCI

The PCI interface provides two address windows for an external initiator to access on-chip resources. These two windows are both 32 MB in size. Their address offsets, as seen by the external initiator, are determined by the BAR registers in PCI configuration space that are set up at boot time by the CPU. Call these *inward-looking address windows*. The PCI interface also provides three address ranges by which an internal initiator (including the MDP, DMA, TDM, etc.) can access off-chip resources through the PCI bus. Call these *outward-looking address windows*. These provide access to PCI memory space, I/O space, and configuration space.

The two inward-looking address windows are intended to be mapped so that one gives access to on-chip control registers and the other gives access to the SDRAM. The SDRAM portion provides a 32 MB space that is shared between the MECA-4x and any PCI initiator.

The offsets of the outward-looking address windows are fixed, and are shown in Table 4.

5.3 SDRAM Interface

The SDRAM interface provides a 32-bit wide data bus. Its addressing resources consist of two chip selects, 2 bank address bits, and 13 column address bits. This allows many combinations of SDRAM chips to get different memory sizes. Assuming 4-bank memories,

Table 13 shows how different combinations of x32, x16, x8, and x4 parts yield different total memory sizes. The column labelled “Memory Size” has 3 numbers corresponding to 64M, 128M, and 256M parts.

Table 13 SDRAM Package Options

Chip Data Width	Number of Packages	Chip Selects Used	Memory Size (64, 128, 256 Mb parts)
32	1	0	8, 16, 32 MB
16	2	0	16, 32, 64 MB
16	4	0,1	32, 64, 128 MB
8	4	0	32, 64, 128 MB
8	8	0,1	64, 128, 256 MB
4	8	0	64, 128, 256 MB
4	16	0,1	128, 256, 512 MB

In the initial release of the MECA-4x chip, a subset of the options shown in this table will be tested, specifically, those combinations using 2 SDRAM packages of x16 parts.

The SDRAM controller within the MECA-4x is configured through the control registers shown in Table 14 through Table 17. These registers must be properly set for the SDRAM package(s) that are used in the system.

Table 14 SDRAM Controller Config Reg 0, address 0x412F0000

Bits	Field name	Description
31:28	PGOPENMIN	Min cycles that a page can be open (i.e. from activate to precharge); defaults to 4
27:24	DPL	Min cycles between write recovery or data-in and precharge command; defaults to 2
23:20	RCD	Min cycles between activate and either read or write command; defaults to 2
19:16	RC	Min cycles between activate and either activate or refresh in the same bank; defaults to 6
15:12	RP	Min cycles between precharge and next command in the same bank; defaults to 2
11:8	MRD	Min cycles between mode register set and next command in the same bank; defaults to 2
7:4	RRD	Min cycles between between activate commands in different banks of the same device; defaults to 2
3:0	CASLAT	CAS Latency. Legal values are 1, 2, and 3; defaults to 2

Table 15 SDRAM Controller Config Reg 1, address 0x412F0004

Bits	Field name	Description
28	CSDUAL	Set to ‘1’ if 2 chip selects in use (else 1 chip select, SDCS0_)
27:26	PGPERBANK	Pages per bank: 0-2 for 2048, 4096, 8192
25:24	PGSIZE	Page size: 0-3 for 1K-8K
17:16	PARMODE	0-2 for none, odd, even parity, respectively
15:0	PGOPENMAX	Max cycles that a page can be open (i.e. from activate to precharge); defaults to 9600

Table 16 SDRAM Controller Config Reg 2, address 0x412F0008

Bits	Field name	Description
0	SDCREADY	‘1’ indicates controller is ready; field is read-only

Table 17 SDRAM Controller Config Reg 3, address 0x412F000C

Bits	Field name	Description
15:0	REFCNT	Max cycles between refresh commands; defaults to 1250

5.4 DMA Controllers

There are two DMA Controllers in the MECA-4x, each providing four DMA channels. One controller is the MEMDMA, which performs optimized DMA operations between SDRAM and on-chip units. The other Controller is the GENDMA, which can perform DMA operations from any source to any destination on the chip.

5.4.1 DMA Controller registers

The control space for each DMA Controller is identical, consisting of 3 global registers plus 6 channel registers for each channel. The address space for the DMA Controller is 2048 bytes (0x800) altogether. Channel registers are at the bottom of this address space; global registers are near the top. Base addresses for the DMA Controllers are shown in Table 3. Each register is 32-bits or less. Channel registers are spaced at 64-bit intervals. Table 18 shows the channel register address offsets from the base address of each DMA Controller.

Table 18 DMA Controller Address Map

Offset (hex)	Register Name	Description
Channel 0		
00	DMACMD	Control bits and transfer length for a single DMA command.
08	DMACNTL	Control bits that change less frequently than CMD register.
10	T1ADDR	Address for Target 1 (which may be source or destination).
18	T2ADDR	Address for Target 2 (which may be source or destination).
28	DMANEXT	Address of next DMA command for linked list command chaining.
30	TDMA	TDMA allocation tables for this channel
Channel 1		
40	DMACMD	Control bits and transfer length for a single DMA command.
48	DMACNTL	Control bits that change less frequently than CMD register.
50	T1ADDR	Address for Target 1 (which may be source or destination).
58	T2ADDR	Address for Target 2 (which may be source or destination).
68	DMANEXT	Address of next DMA command for linked list command chaining.
70	TDMA	TDMA allocation tables for this channel
Channel 2		
80	DMACMD	Control bits and transfer length for a single DMA command.
88	DMACNTL	Control bits that change less frequently than CMD register.
90	T1ADDR	Address for Target 1 (which may be source or destination).
98	T2ADDR	Address for Target 2 (which may be source or destination).
A8	DMANEXT	Address of next DMA command for linked list command chaining.
B0	TDMA	TDMA allocation tables for this channel
Channel 3		
C0	DMACMD	Control bits and transfer length for a single DMA command.
C8	DMACNTL	Control bits that change less frequently than CMD register.
D0	T1ADDR	Address for Target 1 (which may be source or destination).
D8	T2ADDR	Address for Target 2 (which may be source or destination).
E8	DMANEXT	Address of next DMA command for linked list command chaining.
F8	TDMA	TDMA allocation tables for this channel

Fields of the command, control, and TDMA registers are shown in Table 19, Table 20, and Table 24, respectively. The registers T1ADDR, T2ADDR, and DMANEXT are 32-bit registers containing byte addresses.

Table 19 DMACMD Register

Name	Field	Description
VALID	31	Entry is valid if set to 1; else channel stops
GINT	30	Generate Interrupt when transfer completes
Reserved	29:27	Set to binary 000
CHEAT	26	Bypass arbitration, remain active if set
T1_SRC or TO_MEM	25	Target 1 is Source if set, Destination if cleared (for MEMDMA, set means memory is Destination)
Reserved	24:21	Set to binary 0001
LD_DMACNTL	20	Load DMACNTL register for next command if set
LD_T1ADDR	19	Load T1ADDR register for next command if set
LD_T2ADDR	18	Load T2ADDR register for next command if set
LD_DMANEXT	17	Load DMANEXT reg for next command if set
Reserved	16	Set to 0
LENGTH	15:0	Transfer length in bytes

The VALID bit is interpreted differently depending on whether the channel is becoming enabled or is chaining; see Section 5.4.2.3, Starting and Stopping. The CHEAT bit is relevant to arbitration, see Section 5.4.2.2, DMA Channel Arbitration. For a caveat regarding the LENGTH field, see Section 5.4.3, DMA Transfer Length Restrictions.

The DMA Controller identifies source and destination in a somewhat unusual way. Rather than having source and destination address registers, there are two address registers (called T1 and T2) plus a bit to say which is source and which is destination. In the MEMDMA, the T2ADDR is required to be the memory address (to SDRAM). Consequently, in the MEMDMA, a transfer from memory to another chip component (such as an MDP register file) *must* place the memory base address in T2 and identify T2 as the source (by clearing the TO_MEM bit in the DMACMD register). In case of a memory to memory copy using the MEMDMA, both T1 and T2 will be memory addresses, so this constraint is relaxed. Also when using the GENDMA engine for transfers involving SDRAM memory, there is no constraint on the order, however, this makes less efficient use of on-chip bus bandwidth.

The DMACNTL register is optional. Its fields are shown in Table 20.

Table 20 DMACNTL Register

Name	Field	Description
	31	Reserved, set to 0
T1_BURST	30:28	Target 1 Addressing Mode (see Section 5.4.2.1, Burst Processing)
T1_WIDTH	27:26	Target 1 Bus Transfer Width (See Table 21)
Reserved	25:23	Set to binary 010
T2_BURST	22:20	Target 2 Addressing Mode (see Section 5.4.2.1, Burst Processing)
T2_WIDTH	19:18	Target 2 Bus Transfer Width (See Table 21)
Reserved	17:12	Set to binary 010000
CHAIN	11	Enable DMA command chaining
Reserved	10	Set to 0
USE_TDMA	9	Use TDMA for Arbitration (see Section 5.4.2.2, DMA Channel Arbitration)
PRI_HI	8	High order bit of Channel Priority
Reserved	7:4	Set to binary 0000
PRI	3:0	Lower 4 bits of Channel Priority

If the CHAIN bit is cleared, the DMA engine stops (becomes disabled) for this channel after every command. If it is set, the next command is determined by the DMANEXT register. For contiguous arrays of DMA commands, this register is updated automatically, while in linked-list mode, the LD_NEXT bit in the DMA command is set to fetch the pointer to the next command. Arrays and links can also be interleaved to create linked pages of DMA commands.

The T1_WIDTH and T2_WIDTH fields are encoded as shown in Table 21.

Table 21 DMA Width Encoding

Value (binary)	Encoded Width (bits)
00	64
01	32
10	16
11	8

This determines the size of bus transactions that are done to accomplish the DMA transfer. For a caveat, see Section 5.4.3, DMA Transfer Length Restrictions.

The USE_TDMA, PRI_HI, and PRI fields control arbitration; see Section 5.4.2.2, DMA Channel Arbitration.

5.4.1.1 DMA Controller Global Registers

The DMA Controller global registers are shown in Table 22, with their offsets from the base address of the DMA Controller and a brief description.

Table 22 DMA Controller Global Registers

Offset (hex)	Name	Description
798	Enable	Set bits 0-3 to enable channels 0-3; 0 bits have no effect
79C	Status	Bits 16-19 shows status of channels 0-3 respectively, 0 means enabled
100	Control	Two 4-bit fields to set limit of major and minor TDMA counters for all channels

The Enable register is used to start each channel between each individual command or chain of commands that it executes. Setting bits in the register starts the corresponding channel; clearing bits has no effect. Writing a 1 to a currently-active channel has no effect.

The Status register uses bits 16-19 to give the status of the DMA channels; other bits in this register should be ignored. This is a read-only register.

The Control register contains 2 fields in bits 0-3 and 4-7 which set the TDMA counter limits for the minor and major counters respectively. See Section 5.4.2.2, DMA Channel Arbitration for further information.

5.4.2 DMA Controller Operation

The per-channel DMA control registers are organized so that between 1 and 5 registers is fetched for each DMA command. When multiple registers are fetched from memory, they are packed on 32-bit boundaries, not 64-bit boundaries as mapped into the address space. However, for a caveat see Section 5.4.4, DMA Command Alignment Restrictions.

The one register that is always fetched is the DMACMD register. It contains the transfer length and a few other important parameters plus four bits to control access of the other four registers. The DMACNTL register contains less commonly-changed parameters. The T1ADDR register contains the start address of one side of the transfer, whether source or destination is set by a bit in the DMACMD register. The T2ADDR register contains the start address of the other side of the transfer. (“T1” stands for “Target 1”; “T2” stands for “Target 2”. Both the source and destination of DMA commands stand in a target relationship

to the DMA controller, which acts as the initiator.) Finally, if linked-mode is used (as controlled by a bit in the DMACNTL register), the DMANEXT register contains the address from which to fetch the 1 to 5 registers required for the next command (assuming CHAIN mode is enabled).

Each of the registers containing addresses (T1ADDR, T2ADDR, DMANEXT) is updated by every DMA command. For example, if T1ADDR starts at address 0x400 in a command that is 0x100 long, it will normally be 0x500 when the command finishes (unless the command is non-incrementing, see Section 5.4.2.1, Burst Processing). Similarly for DMANEXT: if it is 0x40400 at the start of a DMA command and there are 4 registers to fetch for this command, it will be 0x40410 after the command. There is a 64-bit alignment restriction with DMANEXT; see Section 5.4.4, DMA Command Alignment Restrictions.

Because address registers are updated during the execution of DMA commands and fetches of these and other registers can be avoided, it is possible to optimize the use of memory for DMA commands, and to optimize memory bandwidth used for fetching them. For example, it is possible to set up arrays of DMA commands. The last element in such an array can be linked to another array that is discontiguous, so that paged arrays of DMA commands can be implemented without CPU intervention. It is possible to scatter a large array of source data to a set of discontiguous destinations by setting up DMA commands which repeat the source address between commands (leaving out the fetch of T1ADDR) but set up a new destination address (explicitly specifying T2ADDR). Numerous other examples are possible as well.

The DMA Controller has two sections that work in pipeline fashion: a DMA Command Processing section fetches commands and the DMA Transfer Engine carries them out. These are pipelined in the sense that Command Processing for the next active channel is normally overlapped with Transfer Engine processing for the currently active channel. However, if the next active channel is the same as the currently active one, command processing is not overlapped with transfers. Arbitration logic determines the order in which commands are processed for the different channels implemented by a single DMA Controller.

5.4.2.1 Burst Processing

Commands for each channel are normally processed one burst at a time, with arbitration taking place between each such burst. The T1_BURST and T2_BURST fields in the channel control register determine what a burst is. The burst field is also used to encode non-incrementing addresses, such as to MDP stream ports. The encoding is shown in

Table 23:

Table 23 Burst Encoding

Binary Value	Mnemonic	Max Transfers per Burst
010	TWO	2
100	FOUR	4
101	STRM	Transfer length/width
110	EIGHT	8
111	CONT	Transfer length/width

The STRM burst value means that the address is non-incrementing; all other values represent incrementing addresses. A burst value of FOUR, for example, means that 4 transfers (normally 8 bytes each) should take place as a burst and then arbitration should (normally) take place again. The re-arbitration can be circumvented by setting the CHEAT bit in the channel command register. When this bit is set, the entire command will complete before arbitration is entered again.

5.4.2.2 DMA Channel Arbitration

Between each command, the DMA transfer engine arbitrates for the next active channel. Every channel that is ready participates in the arbitration. There are two distinct arbitration policies: TDMA for real-time channels, and priority-based for everything else. Channels using TDMA are higher priority than any others. TDMA means the channel's access to the on-chip bus is synchronized to a configurable, repetitive pattern of bus cycles. Such a channel doesn't become ready (and hence, participate in arbitration) until a bus cycle specifically allocated to the DMA has come by. Once it is ready, it wins arbitration before any priority-based channels. If several TDMA channels are ready at a time, they win arbitration in FIFO order (the order they became ready).

The second arbitration policy is priority-based. There are 5 priority bits for a total of 32 levels. These are interpreted as an unsigned number; higher numbers represent greater priority. The priority bits are split across 2 fields in the DMACNTL register, named PRI_HI and PRI.

The TDMA arbitration policy is selected when the DMACNTL register's USE_TDMA bit is set. Thus, it can be turned on and off in different commands, although this is not likely to be done. The TDMA arbitration hardware can be used to implement variants of fair scheduling policies, including round robin. TDMA arbitration is enabled by the USE_TDMA bit, but is also partly controlled through the per-channel TDMA registers, whose fields are shown in Table 24.

Table 24 Channel TDMA Register

Name	Field	Description
TDMA_MAJOR	31:16	16-bit lookup table for major cycles
TDMA_MINOR	15:0	16-bit lookup table for minor cycles

These TDMA registers control how TDMA events are generated for each channel; when channels use TDMA, these events drive the readiness of the channel to participate in arbitration. TDMA event generation is driven by a clock source coming from the on-chip bus at a configurable rate. At each pulse of this clock, a 4-bit minor cycle counter increments. When it overflows, a 4-bit major cycle counter increments. These counters index the 16-bit major and minor bit tables defined in the TDMA registers for each channel. There is one pair of counters indexing 4 different sets of bit tables (one set for each DMA channel). This means that the bit tables in all the TDMA registers are synchronized. A TDMA event occurs for a channel when the bit corresponding to the counter indices of *both* its major and minor bit tables are set.

For example, suppose the major cycle bit table for a given channel is 0x0044 and the minor cycle bit table is 0x1111. On cycle 0 (both counters set at 0), the indexed bits are 0 and 1 for major and minor, respectively, so no TDMA event occurs. The index bits are both 0 until cycle 4, when they are again 0 and 1, but still, no TDMA event occurs. On cycle 16, the minor counter wraps back to 0 and the major counter increments to 1, but the indexed bit in the major cycle table is still 0, so cycles 16-32 cause no TDMA events. On cycle 32, the major counter increments to 2, which causes the indexed bit in the major cycle table to become 1. At the same time, the minor counter has wrapped to 0 and its indexed bit is also 1, so the first TDMA event occurs on cycle 32. Further TDMA events occur on cycles 36, 40, and 44. By similar logic, TDMA events also occur on cycles 96, 100, 104, and 108. No other TDMA events occur during the 256-cycle total.

It is required that TDMA registers for all the channels be configured so that at most one TDMA event is caused per cycle. This means that each channel's settings must correspond to a disjoint set of cycles.

To make things slightly more complicated (and flexible) the global Control register consists of two 4-bit fields that allow the major and minor counters (for all channels) to wrap at values other than 16. Specifically, the counters wrap at 1+N where N is the value in the corresponding field of the global Control register. For example, by setting these fields to 14 for minor and 11 for major, we get a total cycle length of 180 (instead of 256) and the uppermost bit of all the minor bit tables as well as the uppermost 4 bits of all the major bit tables are ignored.

This hardware allows bandwidth to be allocated to each channel, either equally or unequally. Within limits, it can be spread out evenly or concentrated, and so on.

5.4.2.3 Starting and Stopping

Each DMA channel must be started by setting its bit in the global Enable register. When it is enabled, it examines the VALID bit in its command register. If the VALID bit is set, it processes the command represented by the values currently in its registers, otherwise it fetches a command from the address given by its DMANEXT register. Thus to get started, the external program must either set up a complete command or the VALID bit of the command register and the address in the DMANEXT register.

If the channel is not in CHAIN mode (as determined by the channel control register), it stops upon completion of the current command. Otherwise, it fetches another command from the DMANEXT register. If the VALID bit of the command register of that command is not set, the channel becomes disabled. Thus, to stop a CHAIN of commands, create a final command register value for which the VALID bit is cleared. (Normally, that value would have none of the LD_* bits set, so that only the one 32-bit word is required to end the command sequence.)

5.4.3 DMA Transfer Length Restrictions

The transfer length in bits 15:0 of the DMACMD register is a byte count, however it must always be set to a multiple of both the T1_WIDTH and T2_WIDTH. Since optimal bus utilization will occur when both widths are set to 64 bits, it is best to arrange things so that DMA transfer lengths are a multiple of 8 bytes when possible.

5.4.4 DMA Command Alignment Restrictions

DMA Commands must be aligned on 8-byte boundaries. They are fetched in 64-bit units, so commands having an odd number of 32-bit words must be padded out to a length divisible by 64 bits.

5.5 TDM

The TDM interface consists of I/O sections divisible into groups of data, strobe, and framing signals; these are known as TDM I/O groups. Each group has an associated state machine for data packing or unpacking, framing, and DMA to/from other cores in the chip. These aspects of the TDM interface are covered in the following sections.

In addition, the TDM interface has the following features:

- GPIO pins, with associated control registers
- Utopia Level 2 interface option (which reuses some I/O groups and GPIO pins)
- Global chip-wide interrupt control registers

Each of these features is covered in a separate section, below.

Each of the TDM I/O groups forms a distinct clock domain, driven by its associated strobe signal. The clock domain of each TDM I/O group is distinct from the internal clock domain which is driven by SBCLK. Synchronization hardware is present in the TDM interface to convert data and framing signals between the external and internal clock domains.

5.5.1 TDM Address Map

The TDM register space consists of 9 parts: a set of global registers, 4 sets of registers for the 4 input groups, and 4 sets of registers for the 4 output groups. Their base addresses are shown in Table 25.

Table 25 TDM Address Space Groups

Address (hex)	Description
41100000	Global Registers
411000A0	Input Group 0 Registers
41100160	Input Group 1 Registers
41100220	Input Group 2 Registers
411002E0	Input Group 3 Registers
41100400	Output Group 0 Registers
411004C0	Output Group 1 Registers
41100580	Output Group 2 Registers
41100640	Output Group 3 Registers

The Global Registers are described in Section 5.5.1.1, TDM Global Registers, while the other register groups are described in Section 5.5.3.1, TASG Registers.

5.5.1.1 TDM Global Registers

The TDM global registers are all 32-bit registers, and are packed at 64-bit intervals. Their addresses and descriptions are shown in Table 26 TDM Global Registers:

Table 26 TDM Global Registers

Address	Register	Description
41100000	TDM_GBL_CSR	Control and configuration
41100008	TDM_GBL_INTR_EN	Interrupt Enable
41100010	TDM_GBL_INTR_CSR	Interrupt Status
41100018	TDM_GPIO_INTR_EN	GPIO Interrupt Enable; see Section 5.7, GPIO Pins
41100020	TDM_GPIO_INTR_CSR	GPIO Interrupt Status; see Section 5.7, GPIO Pins
41100028	TDM_GPIO_DATA	GPIO Data; see Section 5.7, GPIO Pins
41100030	TDM_GBL_RESET	Reset bits for different parts of TDM interface
41100038	TDM_GBL_PRIORITY	Priority bits control arbitration
41100040	UTOPIA_TX_CLAV	Utopia register; see Section 5.6, Utopia Mode
41100048	UTOPIA_RX_CLAV	Utopia register; see Section 5.6, Utopia Mode
41100050	UTOPIA_TX_ADDR	Utopia register; see Section 5.6, Utopia Mode
41100058	UTOPIA_RX_ADDR	Utopia register; see Section 5.6, Utopia Mode
41100060	UTOPIA_ATM_CLAV	Utopia register; see Section 5.6, Utopia Mode
41100068	GBL_INTR_EN	Interrupt enable register; see Section 5.8, Interrupt Controller
41100070	GBL_INTR_CSR	Interrupt status register; see Section 5.8, Interrupt Controller

All global registers are cleared on hard reset and when bit 0 of the TDM_GBL_RESET register is set to ‘1’. This means that no I/O groups are enabled, no interrupts are enabled, and all GPIO pins are inputs. The registers are described in the following sections.

TDM_GBL_CSR register

The TDM_GBL_CSR register is used to individually enable input and output groups or combinations of them, and to set the direction of the GPIO pins. Its bit assignments are shown in Table 27.

Table 27 TDM_GBL_CSR Register

Bit	Function
0	Enable Input group 0 if set (ignored if bits 4 or 6 are set)
1	Enable Input group 1 if set (ignored if bits 4 or 6 are set)
2	Enable Input group 2 if set (ignored if bits 5 or 6 are set)
3	Enable Input group 3 if set (ignored if bits 5 or 6 are set)
4	Enable 16-bit Input Aggregate 0-1 if set (ignored if bit 6 is set)
5	Enable 16-bit Input Aggregate 2-3 if set (ignored if bit 6 is set)
6	Enable 32-bit Input Aggregate 0-1-2-3 if set
7	Enable Output group 0 if set (ignored if bits 11 or 13 are set)
8	Enable Output group 1 if set (ignored if bits 11 or 13 are set)
9	Enable Output group 2 if set (ignored if bits 12 or 13 are set)
10	Enable Output group 3 if set (ignored if bits 12 or 13 are set)
11	Enable 16-bit Output Aggregate 0-1 if set (ignored if bit 13 is set)
12	Enable 16-bit Output Aggregate 2-3 if set (ignored if bit 13 is set)
13	Enable 32-bit Output Aggregate 0-1-2-3 if set
15:14	Enable Utopia mode
31:16	If set, the corresponding GPIO pin is an output; if clear, the corresponding GPIO pin is an input.

When aggregates are enabled (bits 4–6 and 11–13) the controlling group is the lowest-numbered one. For example, when 32-bit input mode is enabled (bit 6 = 1), Input group 0 controls all 32 TDM input pins. When aggregates are enabled, the enable bits for the less aggregated options are ignored. For example, if bit 11 is set, bits 7 and 8 are ignored; if bit 13 is set, bits 7-12 are ignored.

Bits 14 and 15 (Utopia mode) disable groups 2 and 3, and causes the TDM data and clock pins to have special meanings; details are found in Section 5.6, Utopia Mode.

TDM_GBL_INTR_EN register

Interrupts are enabled in each group to indicate DMA buffer wrap-around, for framing errors on input groups in certain modes, and for other errors. I/O group state may also be polled. The status of these interrupts is collected into the TDM_GBL_INTR_EN register. Each group gets 4 bits, as shown in Table 28.

Table 28 TDM_GBL_INTR_EN Register

Bit	Condition
0	Input group 0 DMA 0 wrap-around
1	Input group 0 DMA 1 wrap-around
3:2	Input group 1 DMA 0 & 1 wrap-around
5:4	Input group 2 DMA 0 & 1 wrap-around
7:6	Input group 3 DMA 0 & 1 wrap-around
11:8	Input group 0-3 Error
15:12	Unused, reserved.
16	Output group 0 DMA 0 wrap-around
17	Output group 0 DMA 1 wrap-around
18	Output group 0 Framing Error
19	Output group 0 Other Error
23:20	Output group 1 interrupt bits (similar to bits 19:16)
27:24	Output group 2 interrupt bits (similar to bits 19:16)
31:28	Output group 3 interrupt bits (similar to bits 19:16)

TDM_GBL_INTR_CSR register

The 32-bit TDM_GBL_INTR_CSR register records status of TDM conditions indicated by Table 28. When read, each set bit indicates that a condition corresponding to those shown in Table 28 has occurred. For example, if bit 18 is set, a framing error has occurred on Output group 0.

The hardware will set bits in the TDM_GBL_INTR_CSR register even if the corresponding interrupt enable bit is not set in TDM_GBL_INTR_EN, thus allowing polling for event conditions. The event conditions are cleared by writing a 1 to the corresponding bit in TDM_GBL_INTR_CSR. Writing a 0 has no effect, while writing a 1 clears the corresponding condition.

When aggregated inputs or outputs have been selected with the TDM_GBL_CSR, the disabled groups cannot generate errors or other interrupt events. For example, if Input Groups 0 and 1 are aggregated, errors from this 16-bit interface will appear on group 0; group 1 will never get errors.

TDM_GBL_RESET register

The TDM_GBL_RESET register allows selective reset of each part of the TDM interface. The bits of this register are listed in Table 28.

Table 29 TDM_GBL_RESET Register

Bit	Function
0	Reset the TDM global registers
1	Reset Input group 0
2	Reset Input group 1
3	Reset Input group 2
4	Reset Input group 3
5	Reset Output group 0
6	Reset Output group 1
7	Reset Output group 2
8	Reset Output group 3
31:9	unused

Clearing bits in the TDM_GBL_RESET register has no effect, while setting bits resets all control registers associated with the indicated group. For example, writing 0x1FF resets everything. This is a write-only register. Data returned from reading this register is meaningless.

TDM_GBL_PRIORITY register

The TDM_GBL_PRIORITY register contains bits to set the priority for arbitration among InGroups and OutGroups. The interpretation of bits of this register is shown in Table 30:

Table 30 TDM_GBL_PRIORITY Register

Bit	Function
1:0	Priority for InGroup0
5:4	Priority for InGroup1
9:8	Priority for InGroup2
13:12	Priority for InGroup3
17:16	Priority for OutGroup0
21:20	Priority for OutGroup1
25:24	Priority for OutGroup2
29:28	Priority for OutGroup3

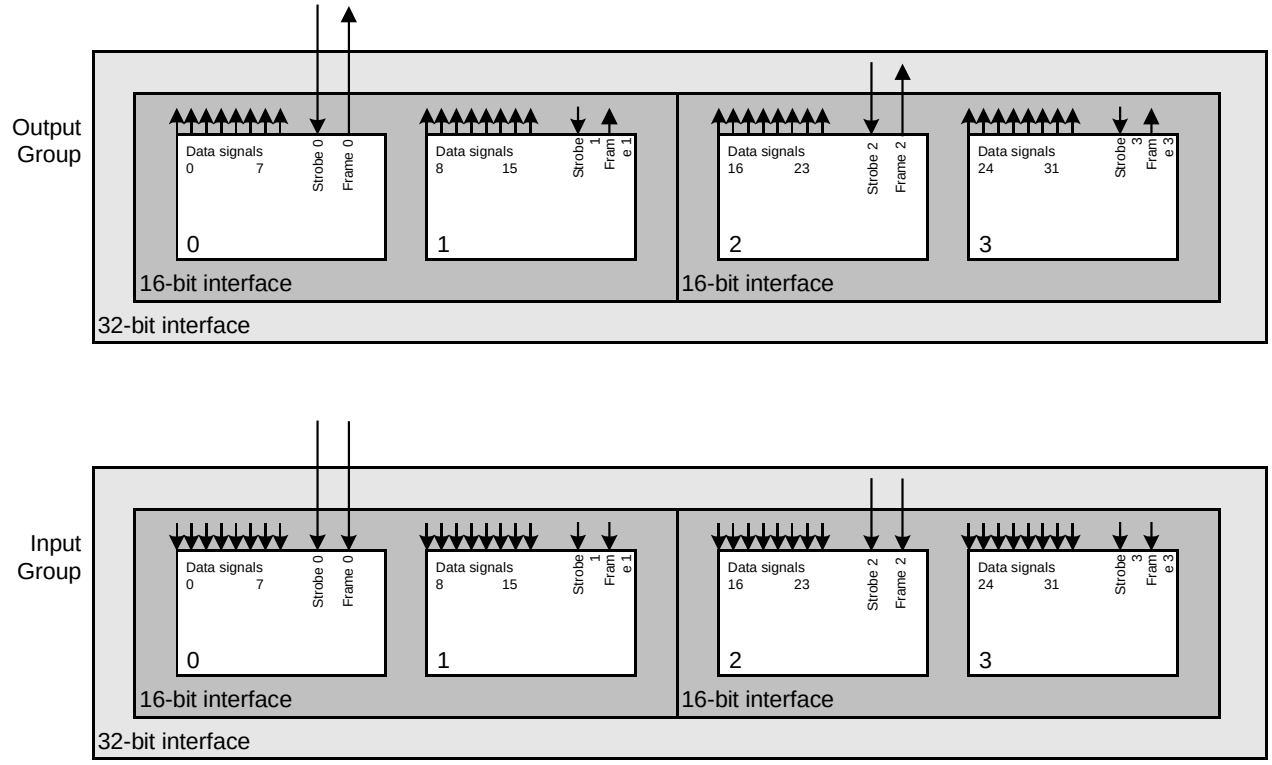
The register consists of eight 2-bit fields, each of which can encode a value between 0 and 3, with 0 being the lowest priority and 3 being the highest. Other bits are unused and should be set to 0. The four InGroups share a common interface to the on-chip bus, and the 4 OutGroups share a common interface to the on-chip bus (but a different interface than for the InGroups). In certain cases where one InGroup (or OutGroup) has much higher strobe frequency than another and the on-chip bus bandwidth allocated for all the InGroups (or OutGroups) is limited, it can become important to set the priority so as to avoid the possibility of overflow or underflow on the pin side of the interface. In this case, the InGroup (or OutGroup) with the highest strobe frequency should be given the highest priority. If two or more groups have the same strobe frequency, their priorities can be made equal.

This register is cleared at reset.

5.5.2 TDM I/O Groups (TDMIOG)

As shown in Figure 12, the TDM implements four input groups and 4 output groups (TDMIOG), numbered 0–3 each. Each group has 8 data signals, a strobe signal, and a framing signal. Thus, there are a total of 32 input data and 32 output data signals, 4 input strobes and 4 output strobes, 4 input framing signals and 4 output framing signals altogether. Each of the strobe signals is associated with the corresponding byte of the data signals: input strobe 0 goes with input data pins 0–7, input strobe 1 goes with input data pins 8–15, and so forth. The framing signals are associated with data signals in the same manner. All of the strobe signals are inputs to the MECA-4x chip.

Figure 12 TDM I/O Groups



These byte-wide signal groups can be aggregated into larger groups. I/O groups 0 and 1 can be aggregated into a 16-bit interface; I/O groups 2 and 3 can be independently aggregated into a 16-bit interface, or all 4 I/O groups can be aggregated into a 32-bit interface. When I/O groups are aggregated, the least-numbered strobe and framing signals in the group are used while the other strobe and framing signals are ignored. Table 31 summarizes the possibilities.

Table 31 TDM I/O Width Mappings

Group	Width	Strobe Pin	Framing Pin	Data Pins
In0	8	TDINSTRB0	TDINFRM0	TDIN0..TDIN7
In1	8	TDINSTRB1	TDINFRM1	TDIN8..TDIN15
In2	8	TDINSTRB2	TDINFRM2	TDIN16..TDIN23
In3	8	TDINSTRB3	TDINFRM3	TDIN24..TDIN31
In01	16	TDINSTRB0	TDINFRM0	TDIN0..TDIN15
In23	16	TDINSTRB2	TDINFRM2	TDIN16..TDIN31
In0123	32	TDINSTRB0	TDINFRM0	TDIN0..TDIN31
Out0	8	TDOUTSTRB0	TDOUTFRM0	TDOUT0..TDOUT7
Out1	8	TDOUTSTRB1	TDOUTFRM1	TDOUT8..TDOUT15
Out2	8	TDOUTSTRB2	TDOUTFRM2	TDOUT16..TDOUT23
Out3	8	TDOUTSTRB3	TDOUTFRM3	TDOUT24..TDOUT31
Out01	16	TDOUTSTRB0	TDOUTFRM0	TDOUT0..TDOUT15
Out23	16	TDOUTSTRB2	TDOUTFRM2	TDOUT16..TDOUT31
Out0123	32	TDOUTSTRB0	TDOUTFRM0	TDOUT0..TDOUT31

5.5.3 TDM Address Sequence Generators (TASG)

Each I/O group has an associated state machine, known as a TASG. For input groups, this state machine either discards received data or assembles it into bus-sized units and transfers it to another core in the chip. For output groups, the state machine either sends a fixed bit pattern to the pins upon request by an output strobe, or requests data from another core and unpacks it to the width required by the group. In addition to these basic functions, Input and Output TASGs have options to sense framing signals on the input side and generate them on the output side, and to be started under program control. These options are designed to facilitate MDP firmware that can acquire framing information from the external interface and synchronize internal buffers to information frames in both directions. Details are found in Section 5.5.3.3, TASG Synchronized Start Logic.

The TASGs have fairly sophisticated facilities for routing fixed amounts of data to and from different places in fixed sequences; certain TDM schemes can be routed and filtered entirely within the TDM interface, with no intervention by CPU or MDP. However, for processing frames whose length is data-dependent, the

data must be sent to an MDP. Details of TASG sequencing facilities are found in Section 5.5.3.2, TASG Sequencing Logic.

5.5.3.1 TASG Registers

Each individual TASG has its own set of control registers, which are the same for each TASG. The base addresses for these register sets are shown in Table 25. The TASG registers are 32-bit registers, and are packed at 64-bit intervals. Their addresses and descriptions are shown in Table 32.

Table 32 TASG Registers

Offset (hex)	Register	Description
00	TASG_BASE_CSR	Overall TASG operation
08	TASG_FRAME_CSR	TASG framing
10	TASG_OUT_CSR	Output characteristics (Output TASGs only)
18	TASG_INTR_CSR	TASG condition status
20	TASG_BASE0	Address values for DMA channel 0
28	TASG_LIMIT0	
30	TASG_CUR0	
38	TASG_BASE1	Address values for DMA channel 1
40	TASG_LIMIT1	
48	TASG_CUR1	
50	FIXED_DATA0	Data to be written when an output TASG uses address code 0
58	FIXED_DATA0_OE	Output enable pattern for FIXED_DATA0
60	FIXED_DATA1	Data to be written when an output TASG uses address code 1
68	FIXED_DATA1_OE	Output enable pattern for FIXED_DATA1
70	PORT_ADDR0	Fixed address associated with address code 4
78	PORT_ADDR1	Fixed address associated with address code 5
80	PORT_ADDR2	Fixed address associated with address code 6
88	PORT_ADDR4	Fixed address associated with address code 7
90, 98, A0, A8	TASG_SEQ_MEM[3:0]	Four-word array, defines in little-endian order what sequence of address codes is used by this TASG.

The two sets of 32-bit registers named TASG_BASE{0,1}, TASG_LIMIT{0,1}, and TASG_CUR{0,1} define address ranges for the two DMA channels associated with each TASG. The FIXED_DATA{0,1} registers define what data are written when an output TASG uses address codes 0 and 1 respectively. The

FIXED_DATA{0,1}_OE registers define output enables for the corresponding data; these registers are meaningful only for Output TASGs. The PORT_ADDR{0,3} registers define the fixed addresses associated with address codes 4–7. The TASG_SEQ_MEM register defines what sequence of address codes is used by this TASG. It is a 4-word array where each word is interpreted as eight 3-bit fields packed on 4-bit boundaries, encoded in little-endian order. Thus, the TASG_SEQ_MEM register contains a maximum of 32 entries.

All TASG control registers are cleared on reset.

TASG_BASE_CSR register

The TASG_BASE_CSR fields are defined in Table 33.

Table 33 TASG_BASE_CSR Register

Field Name	Bits	Description
State	1:0	Values 03 correspond to {Inactive=0, Waiting=1, Counting=2, Active=3} respectively
SeqCurrent	6:2	Current value of the sequence counter
SeqLimit	11:7	Maximum value of the sequence counter
BusGranularity	13:12	Values 03 cause bus transactions when 64, 32, 16, or 8 bits are received, respectively
SeqRptCount	25:14	“Cell length,” that is, the repeat count for each sequence index. Properly, it is the total number of input bytes: 1, 2 or 4 corresponding to 8-, 16-, or 32-bit I/O respectively, and then rounded up.
OEMode	27:26	Output Enable mode (Output TASGs only)
NegEdgeStrb	28	Negate sense of Strobe signal (positive edge if 0, else negative edge)
	31:29	Unused, reserved

The State field is explained in Section 5.5.3.3, TASG Synchronized Start Logic. The SeqCurrent, SeqLimit, and SeqRptCount fields are explained in Section 5.5.3.2, TASG Sequencing Logic.

The OEMode field for output TASGs determines how output enables are set for the data pins associated with the TASG. This field takes on 3 useful values, as shown in Table 34:

Table 34 TASG_BASE_CSR.OEMode Values

Value	Mode	Description
0	OEFix	Always use contents of FIXED_DATA0_OE register
1	OEActive	Use contents of FIXED_DATA0_OE *except* when SeqVal is 0 or 1
2	OEData	Output enables are interleaved with data (see below)

The OEData mode means that the TASG reads twice as much data and uses half of it to set the output enables on a per-transaction basis, and uses the other half as data. In this mode the values in the FIXED_DATA{0,1}_OE registers provide the output enables when the SeqValue register is 0 or 1. In OEData mode, the BusGranularity must be at least twice the TASG width, so that an output transaction can still be serviced by a single bus cycle.

The OEMode field should only be changed when the TASG is in InActive state.

TASG_FRAME_CSR register

The TASG_FRAME_CSR fields are given in Table 35.

Table 35 TASG_FRAME_CSR Register

Field Name	Bits	Description
FrmOpcode	2:0	Code defining how frame signal is used
NegEdgeFrm	3	Inverts sense of framing signal
FrmAddr	6:4	On output, defines pattern while waiting
FrmOffsetCount	22:7	Number of transactions to remain in Counting state
FrmIgnoreMode	23	If set, input TASG ignores data acquired while frame signal asserted, output TASG sends data in FIXED_DATA0 and FIXED_DATA0_OE registers while frame signal asserted.
	31:24	Unused reserved

TASG_OUT_CSR register

The TASG_OUT_CSR fields are given in

Table 36. This register is ignored by input TASGs.

Table 36 TASG_OUT_CSR Register

Field Name	Bits	Description
OutHoldCount	8:0	Drive next data on output lines *N* SBCLK cycles after last output strobe, where *N* is the value of this field. Allows configuration of setup and hold times relative to TxOutStrb.
FrmHoldCount	19:10	Assert frame signal *N* SBCLK cycles after the output strobe that causes it to become enabled, where *N* is the value of this field.
FrmLength	31:20	Hold asserted frame signal for *N* strobe cycles, where *N* is the value of this field.

The OutHoldCount field allows configuration of setup and hold times relative to the TDOUTSTRB*n* signals. (This assumes the strobe signal is periodic.) Specifically, the OutHoldCount determines the number of SBCLK periods that the output data sampled by TDOUTSTRB*n* is held. After this timer expires, the next pre-fetched data is driven onto the pins.

TASG_INTR_CSR register

The TASG_INTR_CSR fields are given in Table 37.

Table 37 TASG_INTR_CSR Register

Field Name	Bits	Description
FrmError	0	Framing error
Overflow	1	Failed to obtain bus cycle; data lost
	31:2	Unused, reserved

These bits provide details about cause of the Input TASG0-3 Error conditions given by bits 8–11 of TDM_GBL_INTR_{EN,CSR} registers.

5.5.3.2 TASG Sequencing Logic

Each input or output group, and therefore each TASG, has a *pin side* and a *bus side*. The width of the pin side is either 8, 16, or 32 bits, and is defined by setting bits in TDM_GBL_CSR. The width on the bus side is either 8, 16, 32, or 64 bits, and is defined by setting the BusGranularity field in the TASG_BASE_CSR register for the corresponding TASG. The TASG sequencing logic determines how sequences of data elements, as defined on the pin side, are handled. With the sequencing logic it is possible for each TASG to define a cell length and then to define a sequence of up to 32 addresses as the source or destination for successive cells; this sequence is repeated indefinitely. For example, one could define a 53-byte cell length on a byte-wide input TASG and then send successive cells first to MDP0, then MDP1, then MDP2, then MDP3. The same sorts of things can be done with respect to source addresses for output data. On the input side, it is possible to check that framing signals are asserted in certain patterns relative to the sequence of destination addresses; on the output side is it possible to generate framing signals in similar ways.

The cell length is defined by the SeqRptCount field in the TASG_BASE_CSR register. This field is in units of the width of the I/O group (i.e. 8, 16, or 32 bits). Properly, SeqRptCount is the total number of input bytes: 1, 2 or 4 corresponding to 8-, 16-, or 32-bit I/O respectively, and then rounded up.

The length of the sequence is defined by the SeqLimit field in the TASG_BASE_CSR register, which should be set to the highest attained sequence memory index, which is one less than the sequence length. The address sequence is defined in the 4 TASG_SEQ_MEM registers. Each of these registers contains eight 3-bit fields at 4-bit offsets (i.e., at bit offset 0, 4, 8, and so on). The least significant 3-bit field defines the first address in the sequence, the next 3-bit field defines the next address in the sequence, and so on. The 3-bit fields encode address value as defined in Table 38.

Table 38 Seq Mem Address Encoding

Value	Description[1]
0	Read: Ignore, Write: fixed data pattern 0
1	Read: Ignore, Write: fixed data pattern 1
2	DMA0
3	DMA1
4	Port 0
5	Port 1
6	Port 2
7	Port 3

Thus, to define a 4-element sequence in which successive cells are sent first to MDP0, then MDP1, then MDP2, then MDP3, one could set the PORT_ADDR0 to a Stream Port address for MDP0, PORT_ADDR1 to a Stream Port address for MDP1, PORT_ADDR2 to a Stream Port address for MDP2, PORT_ADDR3 to a Stream Port address for MDP3, and set TASG_SEQ_MEM[0] to 0x07060504. To complete this example, set the SeqLimit field in the TASG_BASE_CSR register to 3 (not 4).

Another common usage pattern will be to use DMA0 and DMA1 to define two buffers both equal to the cell (or frame) length, then it is possible to arrange to get an interrupt at each frame boundary.

5.5.3.3 TASG Synchronized Start Logic

It is possible to start the TASGs so they are synchronized with external framing. Usually, this is accomplished partly through firmware, by setting the sequence length to the proper frame length, giving the MDP access to the data and/or framing signals and then letting it analyze where in the sequence the framing signal is found. (Note that the framing may be in-band.) Assuming the sequence length is set correctly, the MDP will see the frame start at the same offset in its buffer each time. Then it needs to communicate that offset to the TDM synchronized start logic in such a way that it can skip the right number of samples so that frame start coincides with the start of the internal buffer. Similar considerations apply to output frames.

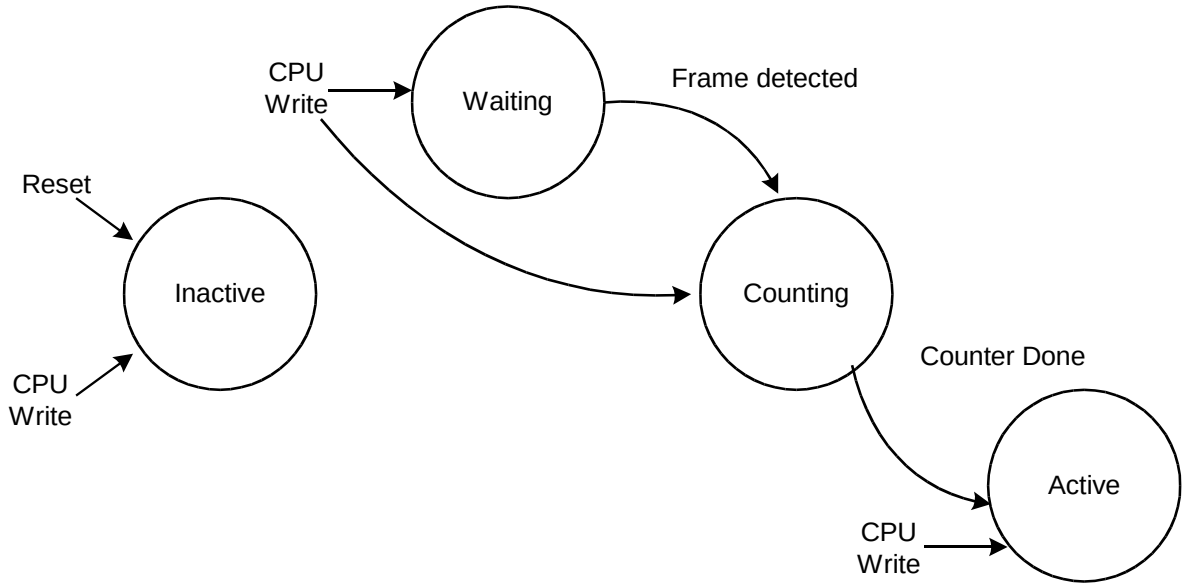
To this end, TASGs have 4 states, defined as follows:

[1] *Read* means sample an input on TDIn pins; *write* means drive an output on TDOut pins.

- Inactive = 0
- Waiting = 1
- Counting = 2
- Active = 3.

See Table 33. These can be set manually by the CPU. The state transition diagram is shown in Figure 13.

Figure 13 TASG State Transition Diagram



After the firmware-based frame acquisition code determines the correct sequence offset, either the MDP firmware itself or the Host CPU writes that number into the FrmOffsetCount field of the TASG_FRAME_CSR register, and then sets the TASG into Waiting state by writing into the State field of the TASG_BASE_CSR register. Then the input buffer will be aligned within less than 2 frame times.

5.5.3.4 TASG Framing Options

The FrmOpcode and NegEdgeFrm fields of the TASG_FRAME_CSR determine how the framing input signal is used by each TASG. Normally, the framing input signal is asserted high; this is inverted when the NegEdge field is set to 1. The FrmOpcode values have the interpretation given in Table 39.

Table 39 Input TASG FrmOpcode Values

0	No use of framing signal
1	Framing signal qualifies strobe
2	Expect frame signal asserted whenever SeqCurrent changes
3	Expect frame signal asserted at start of each sequence (SeqCurrent set to 0)
4	Replace bit 0 of data with framing signal value

When opcode values 2 or 3 are used and the framing signal is not asserted at the required moment, the error is signalled through the TASG's Error interrupt.

The FrmOpcode field is interpreted differently for output TASGs than input TASGs. The specific values are shown in Table 40.

Table 40 Output TASG FrmOpcode Values

0	No use of framing signal, pin is not driven
2	Assert frame signal at each sequence memory increment
3	Assert frame signal at start of each sequence (SeqCurrent set to 0)

The length of the framing signal (when used) is defined by the FrmLength field in the TASG_OUT_CSR register.

5.5.3.5 TASG Hardware Synchronizer Startup Behavior

As implemented on the MECA-4x, the synchronization hardware between the TDM data pins and the internal logic is not guaranteed to correctly propagate data during the first few transitions of the strobe signal following enablement of the interface. This will cause no problems when the interface is used with continuously clocked external agents, as it is normally intended to be.

5.6 Utopia Mode

Hardware support is provided for the UTOPIA Level 2 specification including MPHY and 8-bit data, with the MECA-4x in the role of master (ATM Layer), or slave (PHY Layer). The required MPHY operations are provided; Direct Status Indication and Multiplexed Status Polling are not. Bits 14 and 15 of the TDM_GBL_CSR enable the UTOPIA Level 2 mode:

Table 41 Utopia Mode Enable Bits

TDM_GBL_CSR[15:14] (binary)	Description
00, 11	Utopia not enabled
01	Utopia enabled; MECA-4x is PHY layer (slave)
10	Utopia enabled; MECA-4x is ATM layer (master)

If the Utopia Level 2 interface is enabled in either the ATM or PHY mode, TDM I/O groups 0 and 1 operate normally while the pins associated with TDM I/O groups 2 and 3 are used for the Utopia interface. Therefore, half the TDM pins are used for Utopia, and the other half can still be used for other streaming I/O requirements.

Support for the Utopia mode includes 5 special TDM global registers, which are listed in Table 26; these are UTOPIA_TX_CLAV, UTOPIA_RX_CLAV, UTOPIA_TX_ADDR, UTOPIA_RX_ADDR, and UTOPIA_ATM_CLAV. These can be reset (along with all the other TDM global registers) by setting bit 0 of the TDM_GBL_RESET register.

5.6.1 Utopia PHY mode

When the Utopia enable bits are 01, the MECA-4x plays the role of the physical layer in the ATM system. In this case, the pins of the TDM interface are interpreted as shown in Table 42:

Table 42 Utopia Pin Assignments in PHY Mode

Old Name	New Name	Type	Description
TDINSTRB2	TxClk	I	Transmit Clock
TDIN16..TDIN23	TxData	I	Transmit Data
TDINFRM2	TxSOC	I	Transmit Start of Cell
TDINFRM3	TxEnb_	I	Transmit Enable
TDOUT24	TxClav	O	Transmit Cell Available
TDIN24	TxPrty	I	Transmit Parity
GPIO0..GPIO4	TxAddr	I	Transmit Address
TDOUTSTRB2	RxClk	I	Receive Clock
TDOUT16..TDOUT23	RxData	O	Receive Data
TDOUT25	RxSOC	O	Receive Start of Cell
TDIN25	RxEnb_	I	Receive Enable
TDOUT26	RxClav	O	Receive Cell Available

Old Name	New Name	Type	Description
TDOUT27	RxPrty	O	Receive Parity
GPIO5..GPIO9	RxAddr	I	Receive Address

In the PHY role, the Utopia level 2 operation is very similar to that of normal TDM operation, the major exception being that the TDM needs to indicate when it has cells available to transmit or receive. The PHY drives the Clav signal when it is polled, and this occurs when the PHY's address is present on the Addr inputs. The MECA-4x's TxAddr is defined by bits [20:16] of the TDM_GPIO_DATA register, while the RxAddr is defined by bits [25:21]. The polling address is applied by the ATM layer to the GPIO[9:0] pins, with the lower 5 bits being the TxAddr and the upper 5 bits being the RxAddr. So, when GPIO[4:0] = TDM_GPIO_DATA[20:16], the MECA-4x is being polled and on the next strobe cycle it must drive the TxClav. To implement this functionality, the UTOPIA_TX_CLAV and UTOPIA_RX_CLAV registers are used. These registers indicate how many cells are available for transmit (in the case of UTOPIA_TX_CLAV), or how many cell buffers are unoccupied (in the case of UTOPIA_RX_CLAV).

Both of these registers are laid out as follows:

Bit	Function
0-7	Write: amount to increment current count, Read: low 8 bits of current count
8-9	Write: ignored, Read bits 8-9 of current count
10	If set, Firmware guarantees that cells are always available

In short, the lower 10 bits of these registers read back as the contents of a 10-bit counter that decrements when the Master side of the Utopia bus drives a cell in or out of this interface. Writes to these registers add between 0 and 255 to the counter. Bit 10 is specially interpreted to mean that the counter can be ignored. Thus, when the Master side of the Utopia bus polls the MECA-4x, the TxClav or RxClav signals are driven high when polled (thereby accepting the poll) if either

1. bit 10 of the appropriate register is set, or

2. the value of the appropriate counter is greater than zero

To ensure proper functioning when Utopia PHY mode is enabled, TASG 2 must be set up in the following way:

Register	Field Name	Value
TASG_BASE_CSR	SeqRptCnt	53
TASG_BASE_CSR	SeqLimit	0
TASG_FRAME_CSR	FrmOpCode	2
TASG_FRAME_CSR	FrmOffsetCnt	0
TDM_GBL_CSR	Bits 16..25	0 (GPIO pins are inputs)

5.6.2 Utopia Master Mode

When the Utopia enable bits are binary 10, the MECA-4x plays the role of the ATM layer. In this case, the pins of the TDM interface are interpreted as shown in Table 43:

Table 43 Utopia Pin Assignments in Master Mode

Old Name	New Name	Type	Description
TDOUTSTRB2	TxClk	I	Transmit Clock
TDOUT16..TDOUT23	TxData	O	Transmit Data
TDOUTFRM 2	TxSOC	O	Transmit Start of Cell
TDOUTFRM3	TxEnb_	O	Transmit Enable
TDIN25	TxClav	I	Transmit Cell Available
TDOUT27	TxPrty	O	Transmit Parity
GPIO0..GPIO4	TxAddr	O	Transmit Address
TDINSTRB2	RxClk	I	Receive Clock
TDIN16..TDIN23	RxData	I	Receive Data
TDINFRM2	RxSOC	I	Receive Start of Cell
TDOUT25	RxEnb_	O	Receive Enable
TDINFRM3	RxClav	I	Receive Cell Available
TDIN24	RxPrty	I	Receive Parity
GPIO5..GPIO9	RxAddr	O	Receive Address

In ATM mode (master), the MECA-4x must poll and select the devices that it wishes to transmit to or receive from. The following UTOPIA_TX_ADDR and UTOPIA_RX_ADDR registers implement the polling functionality. These are similar to the UTOPIA_TX_CLAV and UTOPIA_RX_CLAV registers that indicate

cell availability in PHY mode, but the UTOPIA_TX_ADDR and UTOPIA_RX_ADDR registers also have a field defining the address to poll, as shown in Table 44:

Table 44 UTOPIA_TX_ADDR, UTOPIA_RX_ADDR

Bit	Function
0-4	Address to poll (for TX or RX)
5-12	Write: amount to increment current count, Read: low 8 bits of current count
13-14	Write: ignored, Read bits 8-9 of current count
15	If set, Firmware guarantees that cells are always available

To poll devices, bits [4:0] of the above registers are set with the desired address. Bits [14:5] of these registers each read back as the value of a 10-bit counter indicating how many cells the MECA-4x chip has to transmit (for the UTOPIA_TX_ADDR register), or how much cell buffer space the MECA-4x chip has for receiving cells from the PHY (for the UTOPIA_RX_ADDR register). Bit 15 of either register indicates that the counter should always be treated as non-zero, which is equivalent to asserting that the chip always has a cell ready to transmit (in the case of the UTOPIA_TX_ADDR register) or an available buffer to receive (for the UTOPIA_RX_ADDR register).

If the appropriate counter is non-zero, a poll is initiated to the device address in bits [4:0] of the ADDR register, and when the addressed PHY responds, a cell is transferred automatically. This makes it quite convenient to use the MECA-4x chip in ATM mode interacting with a single ATM PHY: just set the device address once and then manipulate the counter as needed to indicate availability of cells to transmit and/or buffer space to receive cells.

To use the MECA-4x chip in ATM mode with multiple PHYs is more complex. In this case, to transfer a cell (in either direction), set the device address and set the counter to 1 in a single write to the appropriate register. Then wait for the count to become 0 and repeat the process. The UTOPIA_ATM_CLAV register provides additional status information that may be helpful in this case, as well as in monitoring and/or debugging problems with the single-PHY case:

Table 45 UTOPIA_ATM_CLAV

Bit	Function
0	TxClav
1	TxTransmitting
3	RxClav
4	RxReceiving

The Clav fields in the UTOPIA_ATM_CLAV register record the state of the TxClav/RxClav inputs one cycle after the device is polled. The TxTransmitting and RxReceiving bits in the UTOPIA_ATM_CLAV register indicate that the MECA-4x is presently transmitting/receiving.

To ensure proper functioning when Utopia Master mode is enabled, TASG 2 must be set up in the way shown in Table 46.

Table 46 Utopia Master Mode Settings

Register	Field Name	Value
TASG_BASE_CSR	SeqRptCnt	53
TASG_BASE_CSR	SeqLimit	0
TASG_FRAME_CSR	FrmOpCode	2
TASG_FRAME_CSR	FrmOffsetCnt	0
TDM_GBL_CSR	Bits 16..25	all 1 (GPIO pins are outputs)

5.7 GPIO Pins

There are 16 GPIO (General Purpose I/O) pins configurable as either inputs or outputs. When configured as outputs, these pins are settable individually and in groups. When configured as inputs, these pins are are registered as interrupt sources and selectable transitions will generate interrupts. These pins are configured through control registers associated with the TDM module. See Section 5.5, TDM.

The upper 16 bits of the TDM_GBL_CSR register are used to set the direction of the GPIO pins. See Table 27. As documented in Section 5.6, Utopia Mode, certain of the GPIO bits are reserved when Utopia mode is in effect.

The TDM_GPIO_INTR_EN register selects interrupt causes for the GPIO pins, independently for positive-going and negative-going transitions. See Table 47.

The TDM_GPIO_INTR_CSR register works with the TDM_GPIO_INTR_EN register just like the TDM_GBL_INTR_CSR register works with the TDM_GBL_INTR_EN register: bits set on read indicate corresponding interrupt events have happened, writing 0's clears corresponding interrupts.

The TDM_GPIO_DATA register records the state of the 16 GPIO pins. See Table 48.

For pins that have been set to be outputs, the corresponding TDM_GPIO_DATA bit is the value last written to that bit of the register; for pins that have been set to be inputs, it is the current value driven on the pin by whatever source is driving it. Writes to the TDM_GPIO_DATA register only affect bits corresponding to output pins (as determined by the appropriate field in the TDM_GBL_CSR register).

TDM_GPIO_INTR_EN register

The TDM_GPIO_INTR_EN register selects interrupt causes for the GPIO pins, independently for positive-going and negative-going transitions.

Table 47 TDM_GPIO_INTR_EN register

Bit	Function
0-15	Enable interrupt on positive edge
1631	Enable interrupt on negative edge

TDM_GPIO__CSR register records INTR_CSR register

The TDM_GPIO_INTR status of TDM GPIO conditions indicated by Table 47. When read, each set bit indicates that a condition corresponding to those shown in Table 47 has occurred. For example, if bit 0 is set, a positive-edge transition has occurred on GPIO signal 0.

The hardware will set bits in the TDM_GPIO_INTR_CSR register even if the corresponding interrupt enable bit is not set in TDM_GPIO_INTR_EN, thus allowing polling for event conditions. The event conditions are cleared by writing a 1 to the corresponding bit in TDM_GPIO_INTR_CSR. Writing a 0 has no effect, while writing a 1 clears the corresponding condition.

TDM_GPIO_DATA register

The TDM_GPIO_DATA register records the state of the 16 GPIO pins as shown in Table 48.

Table 48 TDM_GPIO_DATA register

Bit	Function
0	GPIO pin 0
…	…
15	GPIO pin 15
16-25	RX and TX device number in Utopia PHY mode, see Section 5.6, Utopia Mode

For pins that have been set to be outputs, the corresponding TDM_GPIO_DATA bit is the value last written to that bit of the register; for pins that have been set to be inputs, it is the current value driven on the pin by whatever source is driving it. Writes to the TDM_GPIO_DATA register only affect bits corresponding to output pins as determined by the appropriate field in the TDM_GBL_CSR register.

5.8 Interrupt Controller

The interrupt controller registers for the whole chip are located in the TDM register space. The interrupt controller receives interrupts from each MDP, each DMA controller, the TDM interface, and the aggregated GPIO pins. The Interrupt Controller itself contains two registers: GBL_INTR_EN and GBL_INTR_CSR.

The GBL_INTR_CSR records the status of each of the interrupt sources, while the GBL_INTR_EN register has enable bits corresponding to each source. Each global interrupt bit is masked with its enable, and the result is OR-reduced to one overall chip interrupt signal. This signal is then routed to the PCI Bus for use as a PCI interrupt source, and is also made available on the INTR pin. The interrupt status bits are registered into the GBL_INTR_CSR before masking with the GBL_INTR_EN register so that software or Firmware can poll for event conditions.

The Interrupt Controller also contains hardware to allow up to 4 distinct interrupts to be generated under software (or Firmware) control. These Firmware-initiated interrupts are intended to allow the Firmware running on one or more of the MDP cores to generate interrupts to the Host to signal Firmware-detected errors, alarm conditions, or the like.

Except for the bits that cause Firmware-initiated interrupts, the bits in the GBL_INTR_CSR and GBL_INTR_EN registers are interpreted in the same way, which is shown in Table 49:

Table 49 GBL_INTR_EN and GBL_INTR_CSR registers

Bit	Function
0	MDP0 interrupt
1	MDP1 interrupt
2	MDP2 interrupt
3	MDP3 interrupt
4	General DMA interrupt
5	SDRAM DMA interrupt
6	TDM Global interrupt
7	GPIO interrupt
8	Firmware-initiated interrupt 0
9	Firmware-initiated interrupt 1
10	Firmware-initiated interrupt 2
11	Firmware-initiated interrupt 3
12	Initiate Firmware interrupt 0 (CSR only)
13	Initiate Firmware interrupt 1 (CSR only)
14	Initiate Firmware interrupt 2 (CSR only)
15	Initiate Firmware interrupt 3 (CSR only)
31:16	Unused

When writing to the GBL_INTR_EN register, writing a 1 to a bit enables the interrupt while writing a 0 disables the interrupt. Only bits 0 to 11 of the GBL_INTR_EN register are meaningful.

When writing to the GBL_INTR_CSR register bits 0 to 11, writing a 1 to a bit clears the bit while writing a 0 to a bit has no effect. When writing to the GBL_INTR_CSR register bits 12 to 15, writing a 1 to a bit causes the corresponding interrupt to occur. For example, writing 0x1000 to the GBL_INTR_CSR register causes Firmware-initiated interrupt 0 (assuming bit 12 of the GBL_INTR_EN register is set), and the GBL_INTR_CSR will read back as 0x100 indicating the interrupt source. When reading the GBL_INTR_CSR register, a 1 bit means that an interrupt has been caused by the corresponding event. Bits 12 to 15 of the GBL_INTR_CSR register always read back as 0.

5.9 JTAG

The MECA-4x includes a JTAG port that supplies the four required signals TDI, TDO, TMS, and TCK, plus the optional TRST. These are described in IEEE std. 1149.1 as follows:

Table 50 JTAG Signals

Pin	Description	Function
TDI	Test data input	Serial input pin for instructions as well as test and programming data. Data is shifted in on the rising edge of TCK.
TDO	Test data output	Serial data output pin for instructions as well as test and programming data. Data is shifted out on the falling edge of TCK. The pin is tri-stated if data isn't being shifted out of the device.
TMS	Test mode select	Input pin that provides the control signal to determine the transitions of the TAP controller state machine. Transitions within the state machine occur at the rising edge of TCK. Therefore, TMS must be set up before the rising edge of TCK. TMS is evaluated on the rising edge of TCK.
TCK	Test clock input	The clock input to the BST circuitry. Some operations occur at the rising edge, while others occur at the falling edge.
TRST_	Test reset input	Active-low input to asynchronously reset the boundary-scan circuit.

The IEEE1149.1 BST circuitry requires the following registers:

- Instruction register: Used to determine the action to be performed and the data register to be accessed. In MECA-4x, this register is 4 bits.
- Bypass register: A 1-bit-long data register used to provide a minimum-length serial path between TDI and TDO.
- Boundary scan register: A shift register composed of all the boundary-scan cells of the device.

The JTAG port on the MECA-4x includes the required BST Instructions plus one additional instruction named SMI_ACTIVE. Coding of all the BST Instructions is shown in Table 51.

Table 51 BST Instruction Codes

Instruction	Code
BYPASS	1111
EXTEST	0000
IDCODE	0001
SAMPLE PRELOAD	0011
CORE_SCAN	1010
BIST_MODE	0110
BIST_RESULT	0101
CLAMP	0010
SMI_ACTIVE	1100

The use of the SMI_ACTIVE instruction is required only in low level software supplied by PMC-Sierra.

5.10 PLL

The on-chip PLL creates the internal clock SBCLK by multiplying the input clock signal SCLK by 1/2, 2, or 4. It can also be bypassed, which causes SCLK to drive SBCLK directly with a buffer delay. Except in bypass mode, the PLL cannot guarantee any particular phase relationship between SCLK and SBCLK. The PLL is controlled by two signal pins PLLMODE0 and PLLMODE1as shown in Table 52.

Table 52 Interpretation of PLLMODE Pins

PLLMODE[1:0] (binary)	SBCLK/SCLK	Notes
00	1/1	Bypass mode
01	1/2	
10	2/1	
11	4/1	

The minimum frequency for SCLK is 20 MHz except when PLLMODE is binary 00 (bypass mode). The maximum frequency for SBCLK is 100 MHz. The SCLK input frequency and PLLMODE pins should be jointly set to respect these constraints.

6 AC/DC CHARACTERISTICS

6.1 DC Characteristics

Table 53 Absolute Maximum Ratings Over Operating Case Temperature Range (Unless Otherwise Noted)

Supply voltage range, CV DD (see Note 1)[1]	–0.3 V to 3 V
Supply voltage range, DV DD1	–0.3 V to 4 V
Input voltage range	–0.3 V to 4 V
Output voltage range	–0.3 V to 4 V
Operating case temperature range, C	0 C to 90 C
Storage temperature range	T stg –55C to 150 C

Stresses beyond those listed in Table 53 may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated on Table 54 is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

Table 54 Recommended Operating Conditions

Description	Min	Nom	Max	Unit
I_{OL} Low-level output current			8	ma
T_C Operating case temperature	0	25	70	C
CV_{DD} Supply voltage	2.375	2.5	2.625	Volt
DV_{DD} Supply Voltage	3	3.3	3.6	Volt
V_{SS} Supply Ground	0	0	0	Volt
V_{IH} High-level input voltage	2			Volt
V_{IL} Low-level input voltage			0.8	Volt
I_{OH} High-level output current			8	ma

1 All voltage values are with respect to V SS.

Table 55 Electrical Characteristics Over Recommended Ranges Of Supply Voltage And Operating Case Temperature (Unless Otherwise Noted)

Parameter Test Conditions	Min	Typ	Max	Unit
V_{OH} High-level output voltage	2.4			volt
V_{OL} Low-level output voltage			0.6	volt
Input current ‡ VI = V_{SS} to DV_{DD}			10	uA
I_{OZ} Off-state output current V_O = DV_{DD} or 0 V			10	uA
$I_{DD}2V$ Supply current, CV_{DD} = NOM, CPU clock = 100 MNz		TBD		mA
$I_{DD}3V$ Supply current, I/O pins # DV_{DD} = NOM, CPU clock = 100 MHz		TBD		mA
Ci Input capacitance			10	pF
Co Output capacitance			10	pF

6.2 AC Characteristics

6.2.1 TDM

Timing diagrams for TDM input and output are shown in Figure 14 and Figure 15:

Figure 14 TDM Input Timing

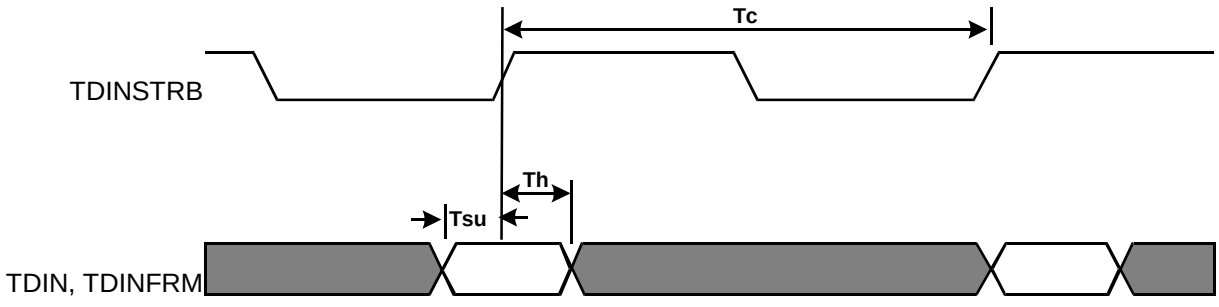


Figure 15 TDM Output Timing

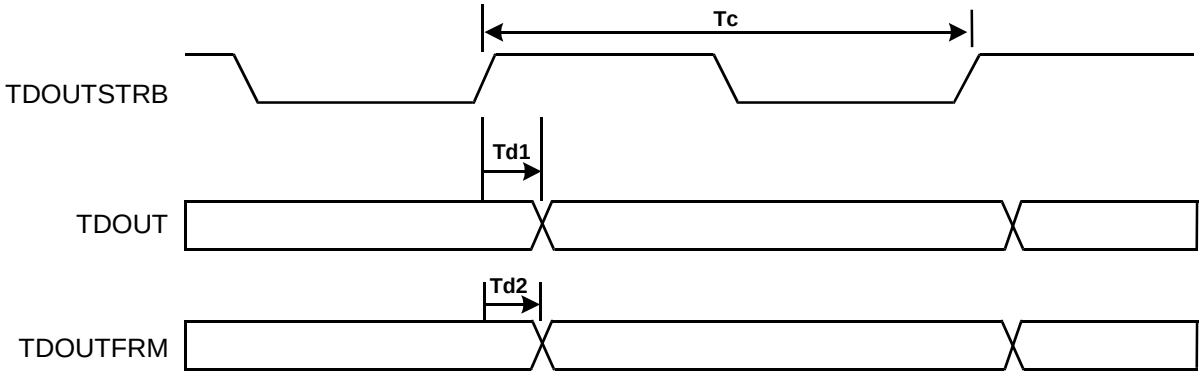
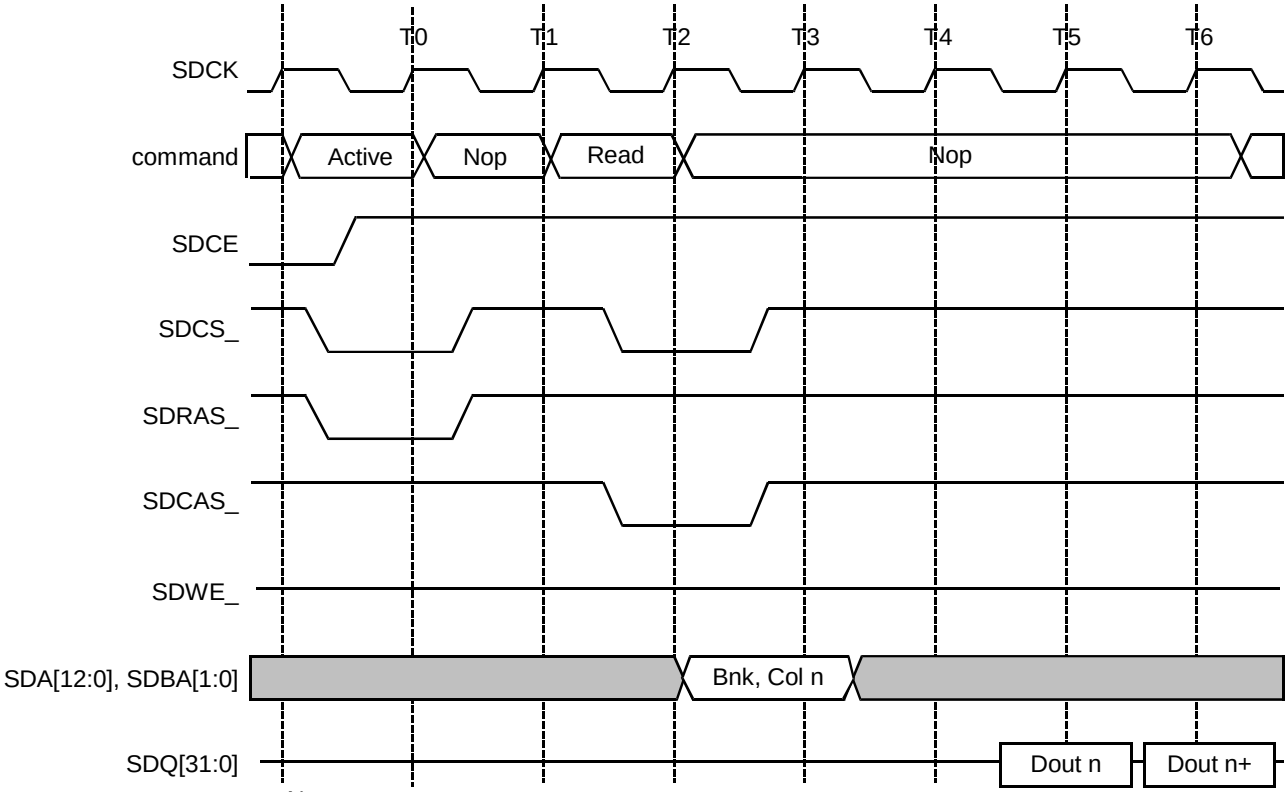


Table 56 TDM Timing Definitions

Signal	Min	Max	Notes
Tc	10		
Tsu	2		
Th	0		
Td1	10		Settable in SBCLK units
Td2	10		Settable in SBCLK units

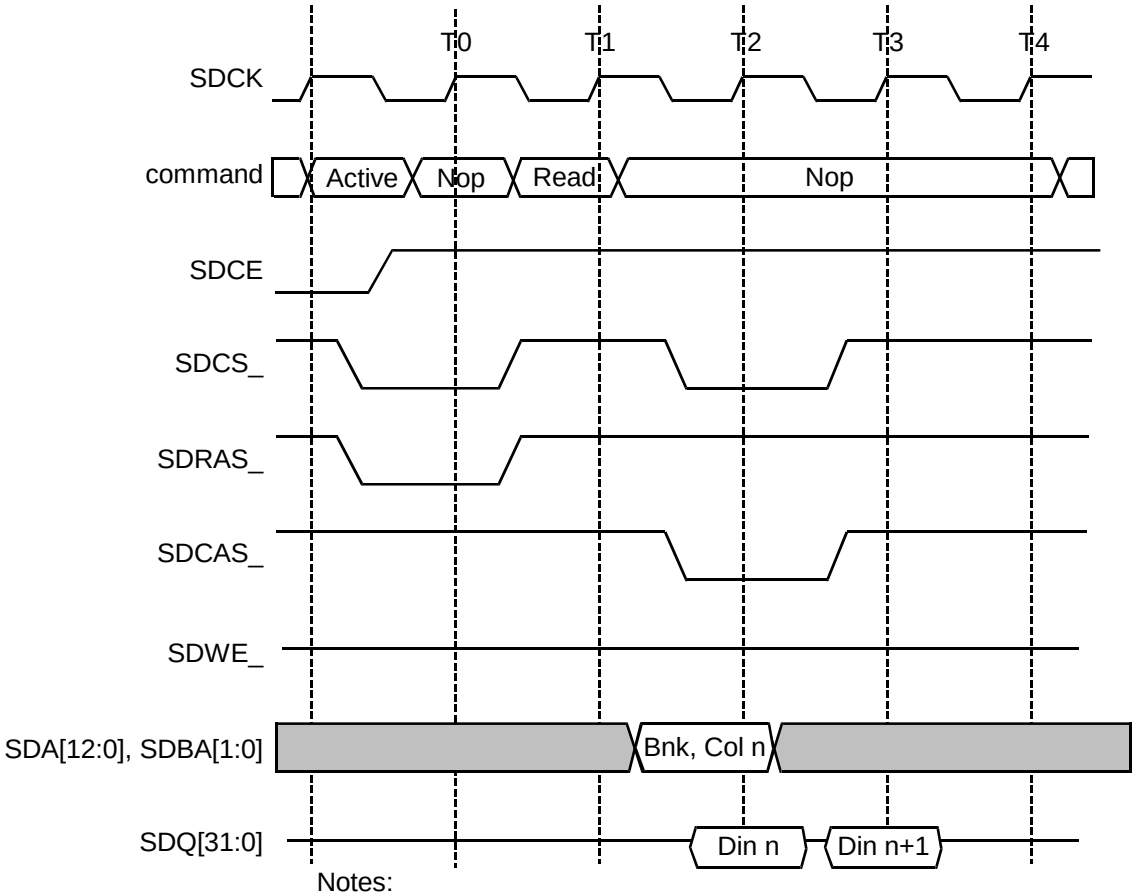
6.2.2 SDRAM

Figure 16 SDRAM Read Timing



Notes:
1. Burst Length = 2
2. CAS Latency = 3

Figure 17 SDRAM Write Timing



Notes:
1. Burst Length = 2

6.2.3 PCI

A few basic timing diagrams for PCI reads and writes are shown in Figure 18 and Figure 19. The PCI interface in the MECA-4x is compliant with version 2.1 of the PCI standard; readers should refer to that document for more complete information.

Figure 18 PCI Read Transaction Timing

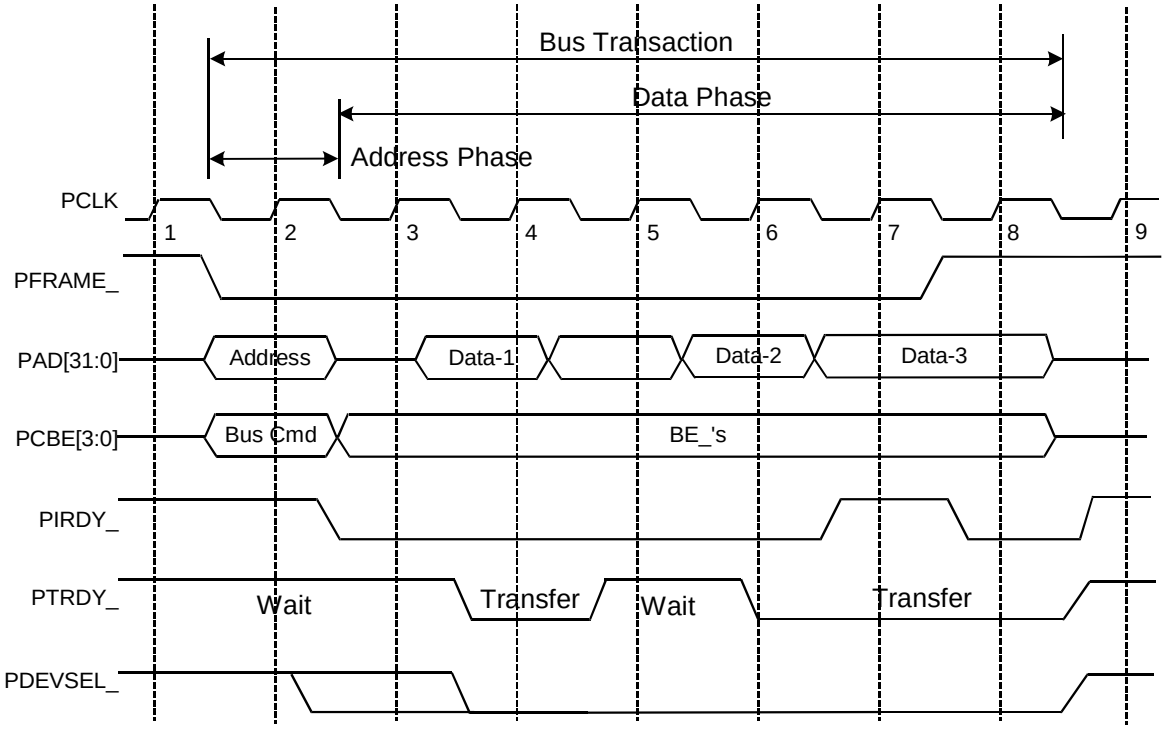
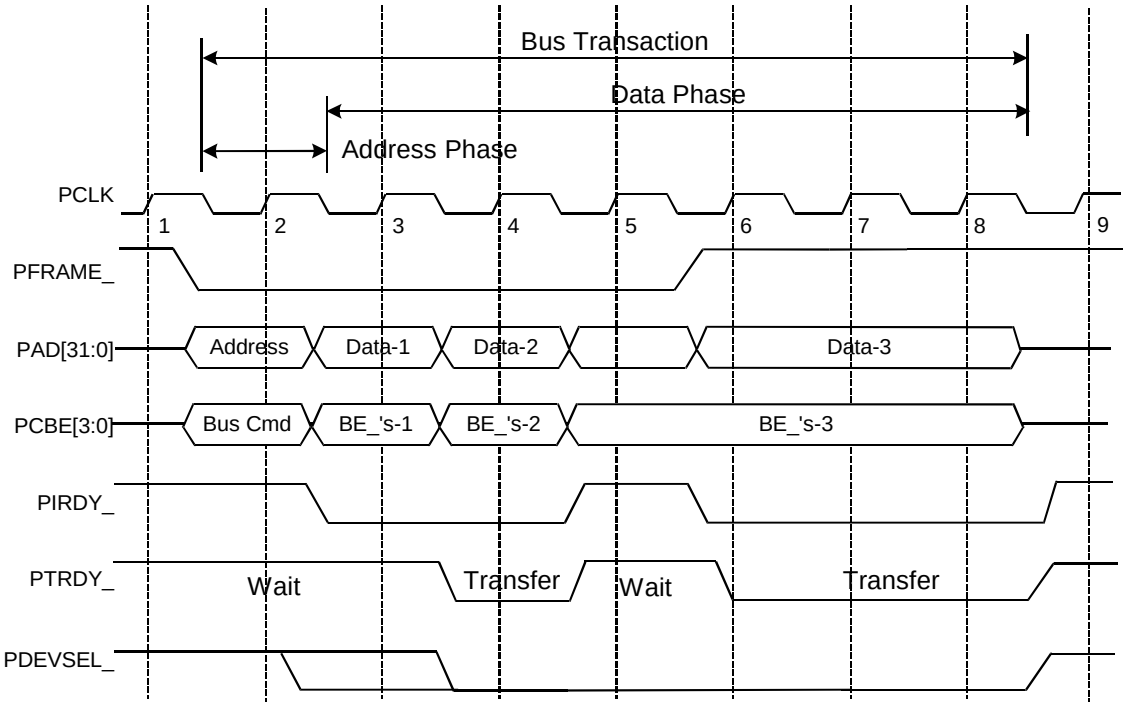


Figure 19 PCI Write Transaction Timing



6.2.4 Utopia

Figure 20 Utopia Transmit 0 Timing

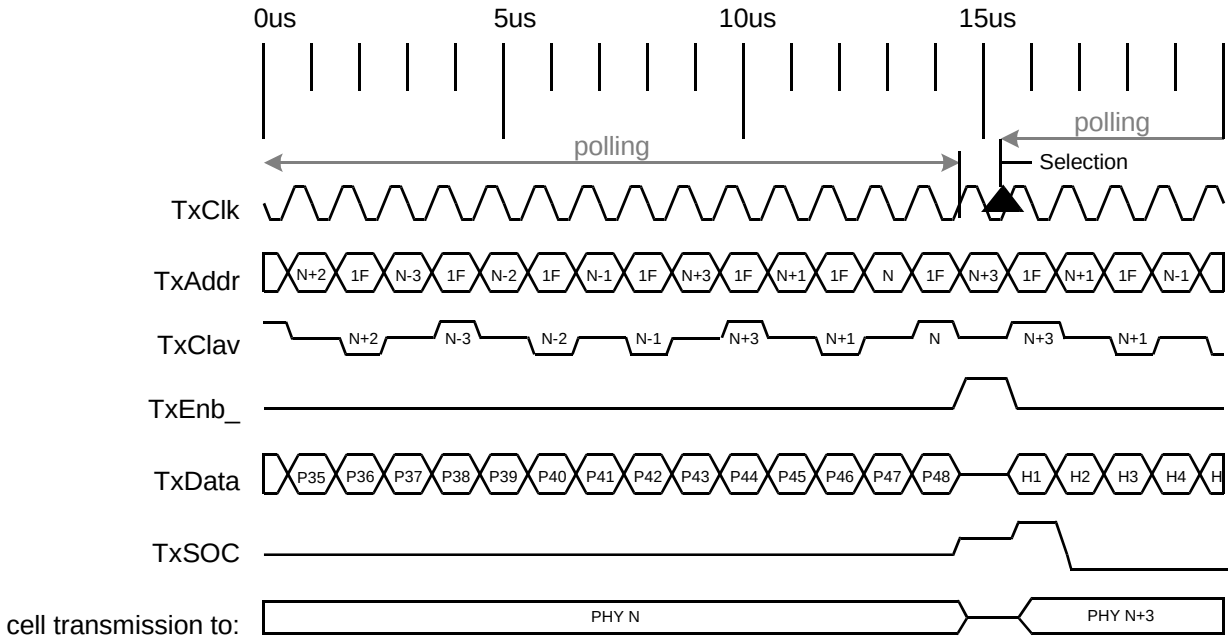


Figure 21 Utopia Transmit 1 Timing

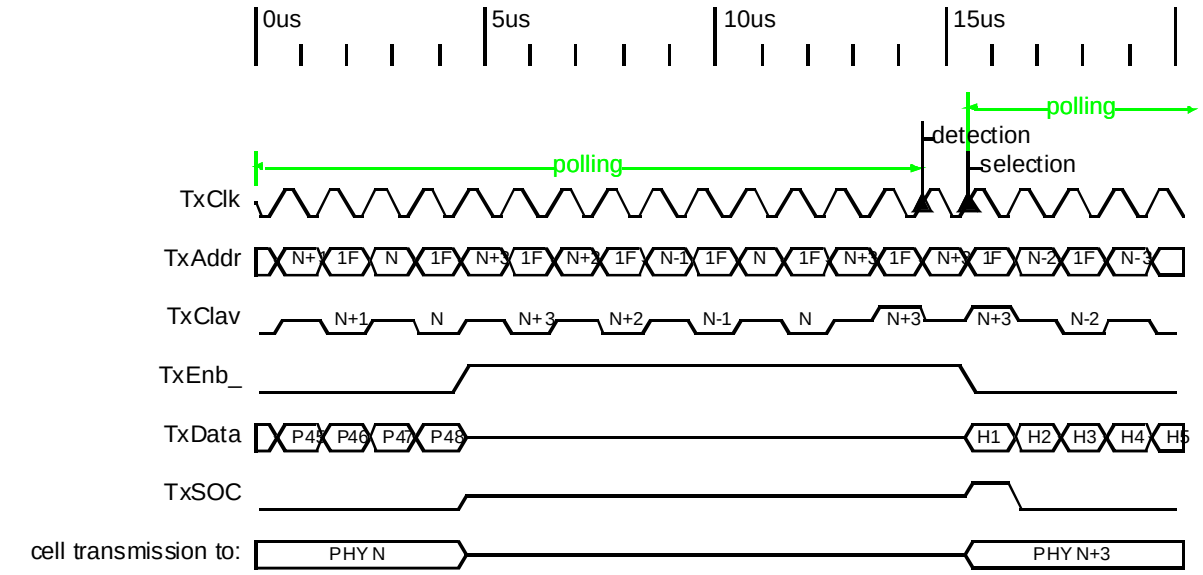


Figure 22 Utopia Receive 0 Timing

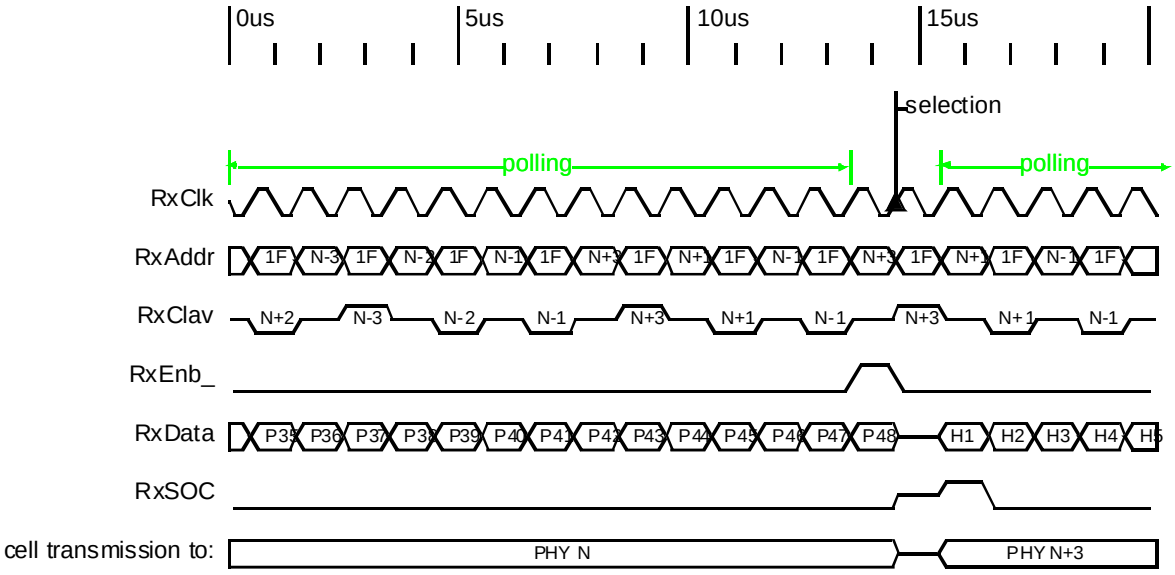
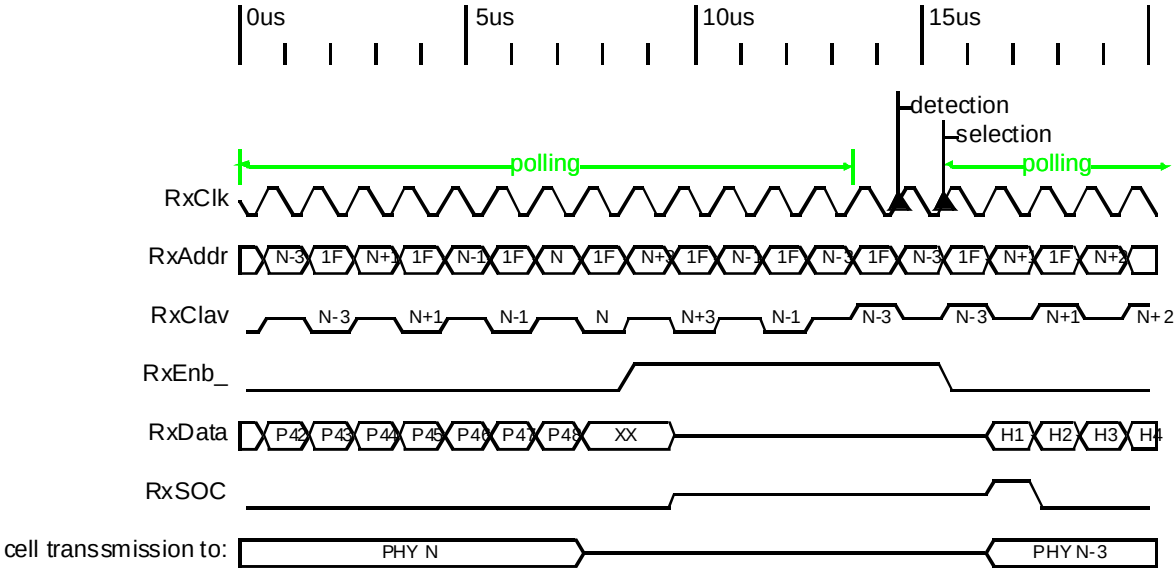


Figure 23 Utopia Receive 1 Timing



INDEX

NOTES:

CONTACTING PMC-SIERRA, INC.

PMC-Sierra, Inc.
105-8555 Baxter Place Burnaby, BC
Canada V5A 4V7

Tel: (604) 415-6000

Fax: (604) 415-6200

Document Information: document@pmc-sierra.com
Corporate Information: info@pmc-sierra.com
Application Information: apps@pmc-sierra.com
(604) 415-4533
Web Site: http://www.pmc-sierra.com

PMC-2001253 (P1) Issue date: August 2000