

inmos®

THE T9000 TRANSPUTER HARDWARE REFERENCE MANUAL

1st edition 1993

ST **SGS-THOMSON**
MICROELECTRONICS

INMOS is a member of the SGS-THOMSON Microelectronics Group

INMOS transputer databook series

Transputer Databook

Military and Space Transputer Databook

Transputer Development and *iq* Systems Databook

Transputer Applications Notebook: Architecture and Software

Transputer Applications Notebook: Systems and Performance

T9000 Transputer Hardware Reference Manual


T9000 Transputer Instruction Set Manual

T9000 Development Tools – Preliminary Datasheets

INMOS reserves the right to make changes in specifications at any time and without notice. The information furnished by INMOS in this publication is believed to be accurate; however, no responsibility is assumed for its use, nor for any infringement of patents or other rights of third parties resulting from its use. No licence is granted under any patents, trademarks or other rights of INMOS.

INMOS Limited 1993

 , IMS, occam and DS-Link are trademarks of INMOS Limited.

 is a registered trademark of the SGS-THOMSON Microelectronics Group.

INMOS Limited is a member of the SGS-THOMSON Microelectronics Group.

INMOS document number: 72 TRN 238 01

ORDER CODE: DBT9000RMST/1

Printed in Italy

Contents overview

Part 1: IMS T9000 Product Family Overview	1
1 Introducing the IMS T9000	3
2 The IMS T9000 transputer	7
3 Simplicity of system design	17
4 Protection and error handling	21
5 Support for multiprocessing	23
6 Communication links	31
7 Network communications	39
8 Other communications devices	47
9 Software and systems	49
Part 2: IMS T9000 Transputer Preliminary Data	55
1 IMS T9000 introduction	59
2 Pin designations	63
3 Central processing unit	65
4 Floating point unit	77
5 Memory management	79
6 Instruction set	85
7 Performance	101
8 Control system	105
9 Instruction and data cache	129
10 Programmable memory interface	149
11 Communications	187
12 Events	205
13 Data/Strobe links	209
14 Clocking phase locked loops	221
15 Configuration register reference guide	225
16 Package specifications	233
17 Thermal management	235
18 Electrical specifications	239
Part 3: Communications Support Devices	243
1 IMS C100 system protocol converter preliminary data	245
2 IMS C104 packet routing switch product preview	305
3 IMS C101 parallel DS-Link adaptor product preview	307
Appendices	309
A IMS T9000 special values	309
B IMS T9000 quick reference guide	311

Contents

Preface	xvii
Notation and nomenclature	xviii
Significance	xviii
Signal naming conventions	xviii
Timing diagram conventions	xviii
Font conventions	xviii
References	xix
Transputer product numbers	xx
Part 1: IMS T9000 Product Family Overview	1
1 Introducing the IMS T9000	3
1.1 Performance	3
1.2 Multiprocessing	3
1.3 Communications support devices	4
1.4 Software	4
1.5 Applications	5
2 The IMS T9000 transputer	7
2.1 Overview	7
2.1.1 Processor	7
2.1.2 Hierarchical memory system	8
2.1.3 Communications system	8
2.1.4 Multiple internal buses	8
2.1.5 Control system	9
2.1.6 Clocks	9
2.2 The transputer architecture	9
2.3 Support for concurrent processes	10
2.4 Pipelined, superscalar implementation	10
2.4.1 The pipeline	11
2.5 Hierarchical memory system	12
2.5.1 Main cache	13
Cache operation	14
Use as on-chip RAM	15
2.5.2 Workspace cache	15
3 Simplicity of system design	17
3.1 Single 5 MHz clock input	17
3.2 Programmable memory interface	17
3.3 Control links and configuration	17
3.4 Loading and bootstrapping	18
3.5 Examples	19

4	Protection and error handling	21
4.1	Error handling	21
4.2	Protected mode	21
4.2.1	Protected mode processes	21
4.2.2	Executing illegal instructions	21
4.2.3	Memory management	22
5	Support for multiprocessing	23
	Fast interrupt response and process switch	23
5.1	The transputer model of concurrency	23
5.1.1	Processes and channels	23
5.1.2	Program structure	23
	Example	24
5.1.3	Multiprocessor programs	25
5.2	Other models of concurrency	26
5.2.1	Shared memory	26
5.3	Hardware scheduler	27
5.4	Interrupts, events and timers	28
5.5	Shared resources	28
6	Communication links	31
6.1	Using links between transputers	31
6.2	Advantages of using links	31
6.2.1	Efficiency	31
6.2.2	Simplicity	31
6.2.3	Hardware independence	32
6.3	IMS T9000 links	32
6.3.1	Virtual channels	33
	Virtual links	33
	Sending packets	34
	Receiving packets	34
	The virtual channel processor	34
	Implementation	35
6.3.2	Levels of link protocol	35
	Packet level protocol	36
	Token level protocol	36
	Bit level protocol	37
7	Network communications	39
7.1	Message routing	39
	Advantages for the programmer	39
	Routers	39
	Separating routers and processors	40
	Parallel networks	40
7.2	The IMS C104	40
	Wormhole routing	40
	Minimizing routing delays	41
	Control links	42

7.2.1	Using IMS T9000s with IMS C104s	42
	Header deletion	42
	Routing control channels	44
7.3	Routing algorithms	44
7.3.1	Labeling networks	45
7.3.2	Avoiding deadlock	46
8	Other communications devices	47
8.1	Mixing transputer types: the IMS C100	47
8.2	Interfacing to peripherals and host systems	47
9	Software and systems	49
9.1	Development software	49
9.1.1	Configuration tools	49
	Hardware description	50
	Software description	50
	Mapping software to hardware	50
	Configuration languages	50
	Types of networks	50
9.1.2	Initializing and loading a network	51
	Levels of initialization	51
	Booting a system from link	51
	Booting a system from ROM	51
9.1.3	Host servers	51
9.1.4	Debugging	52
9.2	IMS T9000 systems products	53

Part 2: IMS T9000 Transputer Preliminary Data	55
1 IMS T9000 introduction	59
2 Pin designations	63
Supplies	63
Phase locked loops	63
Programmable memory interface	63
Control system	64
Communication links	64
Events	64
Miscellaneous	64
3 Central processing unit	65
3.1 Registers	65
3.2 Workspace cache	66
Cache operation	66
3.3 Processes and concurrency	67
3.4 Priority	68
3.5 L-processes: local error handling and debugging	69
3.6 Timers	71
3.7 Block move	72
3.8 Semaphores	72
3.9 Pipeline	72
3.9.1 Grouping of instructions	73
3.10 CPU configuration registers	75
Reason	75
EmiBadAddress	75
InitialPtr and InitialWptr	75
4 Floating point unit	77
4.1 Floating point registers	77
4.1.1 Floating-point stack	77
4.1.2 Floating-point status register	77
4.2 Floating point instructions	78
5 Memory management	79
5.1 Protection, stack extension, and logical to physical address translation	79
5.1.1 Protection	79
5.1.2 Stack extension	79
5.1.3 Logical to physical address translation	79
5.2 Regions	80
5.2.1 Region descriptors	82
5.2.2 Non-overlapping regions	82
5.3 P-process machine registers	83
5.4 Debugging	83

6	Instruction set	85
6.1	Efficiency of encoding	85
6.2	Interaction of the processor pipeline and the instruction set	85
6.3	Instruction characteristics	88
6.4	Instruction set tables	90
6.4.1	Primary instructions	90
6.4.2	Secondary instructions	91
	Sequential instructions	91
	Communication instructions	95
	Process scheduling instructions	97
	Initialization and configuration instructions	98
	Cache operation instructions	98
	Floating point instructions	99
7	Performance	101
7.1	Integer operations	101
7.2	Floating point operations	103
7.3	Predefines	104
8	Control system	105
8.1	Overview	105
8.1.1	Tiers of handshaking	106
8.1.2	Autonomous operation	107
8.2	Control system interconnections	108
8.3	Control system functional description	109
8.3.1	Control links	110
8.3.2	Packet handler	111
8.3.3	Control unit	112
	Command handler	112
	Autonomous control	112
8.3.4	System services	113
	DeviceID	113
	DeviceRevision	113
	ModeStatus	113
	ErrorCode	114
	DSLlinkPLL	114
	SysServWriteLock	114
8.4	Control commands	115
	Start	115
	Reset	115
	Identify	115
	Stop	115
	RecoverError	116
	CPeek	116
	CPoke	116
	Peek	116
	Poke	116
	Boot	116
	BootData	116

	Run	116
	Reboot	116
	Error message	119
8.4.1	IMS T9000 gross state and validity of commands	119
8.5	Errors	121
8.5.1	Recording of Errors	121
8.5.2	Stand alone mode errors	122
8.5.3	Errors on control links	122
8.5.4	Post-mortem debugging of IMS T9000 systems	123
	State delivered to the boot program	123
8.6	Configuration	124
8.6.1	Booting from link	124
8.6.2	Boot from ROM then link	125
8.7	Reset levels	126
8.7.1	Level 0 – hardware reset	126
8.7.2	Level 1 – labelled control network	126
8.7.3	Level 2 – configured network	127
8.7.4	Level 3 – booted network	127
8.7.5	Loading code	127
8.7.6	Levels of reset effect	127
9	Instruction and data cache	129
9.1	Cache overview	129
9.1.1	Cache organization	130
9.2	Cache functional description	133
9.2.1	Port crossbar switch and arbiter	134
9.2.2	Refill engine	135
9.2.3	Replace pointer	135
9.3	Cache operation	136
9.3.1	Cache request	136
9.3.2	Arbitration	138
	Queueing to ensure fairness	138
9.3.3	Cacheable and non-cacheable accesses	140
	Non-cacheable accesses	140
	Cacheable accesses	140
9.3.4	Cache refill cycles	141
	Cache refills from 8/16 bit ports	142
	Write-back cycles	143
	DMA and cache-refill cycles	143
9.4	Cache instructions	144
9.4.1	Flushing data from the cache	145
	Flush dirty cache address (fdca)	145
	Flush dirty cache line (fdcl)	145
9.4.2	Invalidate cache block	145
	Invalidate cache address (ica)	145
	Invalidate cache line (icl)	145
9.4.3	Cache instruction performance	145
9.5	Cache configuration registers	146
9.5.1	RamSize and DoRamSize registers	146
9.5.2	RamLineNumber, RamAddress and DoAllocate registers	147
9.6	Initialization of the cache	147

9.6.1	Reset state	147
9.6.2	Starting the cache	148
10	Programmable memory interface	149
10.1	Pin functions	150
10.1.1	MemData0-63	150
10.1.2	MemAddr2-31	150
10.1.3	notMemWrB0-3	150
10.1.4	notMemRAS0-3	150
10.1.5	notMemCAS0-3	151
10.1.6	notMemPS0-3	151
10.1.7	MemWait	151
10.1.8	MemReqIn, MemGranted	151
10.1.9	MemReqOut	151
10.1.10	notMemBootCE	152
10.1.11	notMemRf	152
10.1.12	notMemStrobe	152
10.2	External bus cycles	152
10.2.1	External DRAM cycles	155
10.2.2	External non-DRAM cycles	158
10.2.3	Bank switching	159
10.2.4	Cache refill cycles	162
	Cache refills from 8/16 bit ports	162
	Write-back cycles	162
	Wait states and cache-refill cycles	162
10.2.5	External DMA	163
10.3	PMI configuration registers	164
10.3.1	Bank address registers	164
	Address registers	164
	Mask registers	165
	RAS bits registers	165
	Format control registers	165
	DoPMIConfigured register	168
	Error address register	168
10.3.2	Strobe timing registers	169
	Strobe registers	169
	Timing control registers	171
	Refresh control register	174
	Remap boot bank register	175
10.3.3	PMI write lock register	175
10.4	PMI errors	176
10.4.1	Errors detected by the PMI	176
	PMI errors signalled by the CPU	176
	PMI errors signalled by the PMI	176
10.5	Initialization of the PMI	177
10.5.1	Bootspace allocation	177
10.5.2	The boot sequence	177
10.5.3	Bootspace timing	178
10.6	Booting from ROM	178
10.6.1	Booting from EPROM	178
10.6.2	Booting from Flash EPROM	179

10.6.3	Re-mapping the boot bank	180
10.7	PMI AC timing characteristics	181
	Read cycle	183
	Write cycle	184
	Consecutive cycles	185
	Memory wait	186
11	Communications	187
11.1	Overview	187
11.1.1	Channels	187
	Internal channels	187
	External channels	187
11.1.2	Channel addresses	187
11.1.3	Communication instructions	187
11.1.4	Efficient variable-length communications	188
11.2	Virtual channel processor	189
11.2.1	VCP protocol	189
11.2.2	Virtual links	189
11.2.3	VCP link queues	190
11.2.4	Virtual link control blocks	191
	vl.HeaderCtrl word	191
	vl.DataQueueLink and vl.AckQueueLink words	192
11.3	Operation of the VCP	193
11.3.1	Channel states	193
	Resetting channels	193
	Stopping channels	194
11.4	Resources	194
11.5	Byte-stream mode	195
11.6	Memory and channel address spaces	196
11.6.1	Channel address space	196
11.6.2	Memory allocation for virtual links	197
	Memory start value register	198
	Minimum invalid virtual channel register	198
	External resource channel base register	198
11.7	VCP configuration registers	199
11.7.1	VCP command register	199
11.7.2	VCP status register	199
11.7.3	Header area base register	200
11.7.4	Header offset registers	200
11.7.5	Packet header limit registers	202
11.7.6	VCP link mode registers	202
11.7.7	ChanWriteLock	202
11.8	Initialization of the VCP	203
11.8.1	VCP state on start up	203
11.8.2	VCP state following reset	203
11.9	Errors	204
	Null buffer pointers	204

12 Events	205
Input event channel	205
Output event channel	205
Use of event channels with interrupts	205
12.1 Event channel addresses	206
12.2 Event channel state	207
13 Data/Strobe links	209
13.1 Link format and protocol	209
13.2 Link functional description	211
13.3 Low-level flow control	212
13.4 Link speed select	212
13.5 Errors on DS-Links	213
13.5.1 Reliable links	214
Handling of errors on reliable links	214
13.5.2 Unreliable links	214
13.6 Link configuration registers	215
13.7 Initialization	217
13.7.1 Link state on start up	217
13.7.2 Link state following reset	217
13.8 Link connections	218
13.9 DS-Link timings	219
14 Clocking phase locked loops	221
14.1 Clock input	221
14.2 PLL decoupling	221
14.3 Processor speed selection	222
14.4 Processor clock output	222
14.5 ClockIn timings	223
14.6 ProcClockOut timings	224
15 Configuration register reference guide	225
15.1 Configuration bus	225
15.2 Subsystem addresses	225
15.2.1 Shared registers	225
15.3 CPU write locking	226
15.4 Subsystem registers	226
15.4.1 CPU configuration registers	227
15.4.2 PMI configuration registers	227
PMI bank address configuration registers	227
PMI strobe timing configuration registers	228
15.4.3 VCP configuration registers	229
15.4.4 System services configuration registers	229
15.4.5 Cache configuration registers	230
15.4.6 Scheduler configuration registers	230
15.4.7 Link configuration registers	230
15.4.8 Control link configuration registers	231

16 Package specifications	233
16.1 208 pin CLCC package pinout	233
16.2 208 pin CLCC package dimensions	234
16.3 208 pin CLCC package thermal characteristics	234
17 Thermal management	235
17.1 Forced air flow cooling	236
17.2 Heat sinks	236
17.3 Other thermal management techniques	237
18 Electrical specifications	239
18.1 Absolute maximum ratings	239
18.2 Operating conditions	239
18.3 Power rating	240

Part 3: Communications support devices 243

1	IMS C100 system protocol converter preliminary data	245
1.1	IMS C100 introduction	246
1.2	IMS C100 modes of operation	248
1.2.1	Mode pins	248
1.2.2	Mode 0: Enables a single T9-series transputer to be used in a T2/T4/T8-series network	249
1.2.3	Mode 1: Enables a T2/T4/T8-series system to use a T9-series subsystem	251
1.2.4	Mode 2: Enables a T9-series system to use an existing T2/T4/T8-series subsystem	252
1.2.5	Mode 3: Enables a T9-series system to use a T2/T4/T8-series subsystem	254
1.3	Link protocols	256
1.3.1	T2/T4/T8-series oversampled links	256
1.3.2	T9-series data/strobe links	257
	Byte-stream mode	259
1.4	Link protocol conversion	260
1.4.1	Byte-stream conversion – modes 0 and 2	260
	Messages from the T9000 to the T2/T4/T8	261
	Messages from the T2/T4/T8 to the T9000	262
1.4.2	Packetized conversion – modes 1 and 3	262
	Messages from the T2/T4/T8 to the T9000	263
	Messages from the T9000 to the T2/T4/T8	263
1.5	Control protocols	265
1.5.1	T2/T4/T8-type control	265
1.5.2	T9-type control	265
	Control link protocols	266
1.6	Control protocol conversion	267
1.6.1	RAE master control (mode 0)	268
	Control commands sent by the IMS C100 in RAE master control mode	269
	Handshake and Error messages received by the IMS C100 from the IMS T9000	270
	Behavior of the control system in RAE master mode	270
	Errors	271
1.6.2	CLink0 master control (modes 1, 2 and 3)	272
	Control commands sent by the controlling processor (IMS T9000) to the IMS C100	273
	Errors	278
	OS-Link 0 special function – modes 2 and 3	279
	Commands which correspond to the protocol of an unbooted T2/T4/T8 transputer	279
	Resetting and Analyzing	280
1.7	Links	281
1.7.1	Data links	281
	Data link speed pins	281
	DS-Link speeds in mode 0	281
	DS-Link speeds in modes 1, 2 and 3	281
	Errors on DS-Links	282
	Link connections	284
1.7.2	Control links	286
	Control link speeds	286
1.7.3	Starting and resetting links	286

1.8	Levels of reset	287
1.8.1	Level 0 – hardware reset	287
1.8.2	Level 1 – labelled control network	287
1.8.3	Level 2 – configured network	287
1.8.4	Level 3	287
1.8.5	Effects of different levels of reset	287
1.9	Configuration	288
1.9.1	Configuration space	288
1.9.2	Configuration register addresses	288
1.9.3	Configuration registers	290
	System services configuration registers	290
	Data DS-Link configuration registers	291
	All data links	292
	Control link configuration registers	292
	Write lock registers	292
1.10	Electrical specifications	293
1.10.1	Absolute maximum ratings	293
1.10.2	Operating conditions	293
1.11	Recommended decoupling	294
1.11.1	Power decoupling	294
1.11.2	Phase locked loop decoupling	294
1.12	Clocks	294
1.12.1	Clock input	294
1.13	Timing specifications	295
1.13.1	Reset and Analyse timings	295
	ResetOut and AnalyseOut timings	295
	TReset and AnalyseIn timings	295
1.13.2	ClockIn timings	296
1.13.3	DS-Link timings	297
1.13.4	OS-Link timings	298
1.14	Pin designations	299
	Supplies	299
	Clocks	299
	Links	299
	Control unit	300
	JTAG support	300
	Miscellaneous	300
1.15	Package specifications	301
1.15.1	IMS C100 100 pin cavity-up ceramic quad flatpack package pinout	301
1.15.2	100 pin ceramic quad flatpack package dimensions	302
1.15.3	IMS C100 100 pin cavity-up plastic quad flatpack package pinout	303
1.15.4	100 pin plastic quad flatpack package dimensions	304
2	IMS C104 packet routing switch product preview	305
2.1	IMS C104 introduction	306
3	IMS C101 parallel DS-Link adaptor product preview	307

Appendices

A IMS T9000 special values 309

A1 IMS T9000 special values 310

B IMS T9000 quick reference guide 311

B1 IMS T9000 quick reference guide 312

 B1.1 Electrical specifications 313

 B1.1.1 Absolute maximum ratings 313

 B1.1.2 Operating conditions 313

 B1.1.3 Power rating 314

 B1.2 Timing specifications 315

 B1.2.1 ClockIn timings 315

 B1.2.2 ProcClockOut timings 316

 B1.2.3 Programmable memory interface timings 317

 Read cycle 318

 Write cycle 319

 Consecutive cycles 320

 Memory wait 321

 B1.2.4 Link timings 322

 B1.3 Processor speed select 323

 B1.4 Link speed select 323

 B1.5 Package details 324

 B1.5.1 208 pin CLCC package pinout 324

 B1.5.2 208 pin CLCC package dimensions 325

 B1.5.3 208 pin CLCC package thermal characteristics 325

Index 327

Preface

The *T9000 Transputer Hardware Reference Manual* provides information on the latest member of the transputer range of microprocessors, the IMS T9000. Transputers are designed to provide extremely high performance in single processor applications and are also designed with hardware and software features for the construction of multiprocessing systems.

Other transputer products include the IMS T225, a 16 bit microprocessor, the 32 bit IMS T4xx series and the IMS T8xx series, which are 32 bit microprocessors with an on-chip 64 bit floating point processor. Details of these and their support devices can be found in *The Transputer Databook*, which is available as a separate publication. Other transputer related documents, including various application and technical notes, are also available from INMOS.

This manual consists of three parts. Part 1, an overview section, introduces the transputer architecture and then the features and benefits of the IMS T9000 family. Part 2, the IMS T9000 transputer preliminary datasheet, contains more detailed information on the IMS T9000 transputer. Part 3 contains information on the communications support devices and includes the IMS C100 system protocol converter preliminary datasheet, IMS C104 packet routing switch product preview and the IMS C101 parallel DS-link adaptor product preview.

More detailed information on the IMS T9000 family communications devices is in preparation, and will be issued as a separate document, titled the *Transputer Networks Manual*. It will contain datasheets on the IMS C104 packet routing switch, the IMS C100 system protocol converter and the IMS C101 parallel DS-Link adaptor.

For full details of the IMS T9000 instructions refer to the *T9000 Transputer Instruction Set Manual*.

Software and hardware examples given in this databook are outline design studies and are included to illustrate various ways in which transputers can be used. The examples are not intended to provide accurate application designs.

In addition to transputer products the INMOS product range also includes development systems and systems products. For further information regarding INMOS products please contact your local SGS-THOMSON sales outlet.

Notation and nomenclature

The nomenclature and notation in general use throughout this databook is described below.

Significance

The bits in a byte are numbered 0 to 7, with bit 0 least significant. The bytes in words are numbered from 0, with byte 0 least significant. In general, wherever a value is treated as a number of component values, the components are numbered in order of increasing numerical significance, with the least significant component numbered 0. Where values are stored in memory, the least significant component value is stored at the lowest (most negative) address.

Similarly, components of arrays are numbered starting from 0 and stored in memory with component 0 at the lowest address.

Transputer memory is byte addressed, with words aligned on eight-byte boundaries on 64 bit devices, four-byte boundaries for 32 bit devices and on two-byte boundaries for 16 bit devices.

Hexadecimal values are prefixed with #, as in #1DF.

Where a byte is transmitted serially, it is always transmitted least significant bit (0) first. In general, wherever a value is transmitted as a number of component values, the least significant component is transmitted first. Where an array is transmitted serially, component 0 is transmitted first. Consequently, block transfers to and from memory are performed starting with the lowest (most negative) address and ending with the highest (most positive) one.

In diagrams, the least significant component of a value is to the right hand side of the diagram. Component 0 of an array is at the bottom of a diagram, as are the most negative memory locations.

Signal naming conventions

Signal names identifying individual pins of a transputer chip have been chosen to avoid being cryptic, giving as much information as possible. The majority of transputer signals are active high. Those which are active low have names commencing with **not**. Capitals are used to introduce new components of a name, as in **ProcClockOut**.

Composite signal names are given for names which refer to a number of pins, with the number of the pins given at the end of the signal name. For example, **MemData0-63** refers to the 64 data pins. Note that **MemData0** represents the least significant bit.

Timing diagram conventions

Cross hatch shading on a waveform diagram indicates that the signal can be high or low.

A minimum timing value with a negative value indicates that the transition of the second signal can occur before the first signal. For example the minimum timing specification for address setup to strobe valid is -6 ns, i.e. the strobe can be valid 6 ns before address setup.

Font conventions

The following font conventions are used throughout this databook.

Parameter	Font	Example
Signal / pin names	bold	ProcClockOut
Register names	bold	Areg
Bit names	bold	SpeedMultiply
Flag names	bold	FPErrorFlag
Errors	italic	<i>FPError</i>
Workspace special values	italic	<i>Notprocess.p</i>
Workspace offsets	bold	Link.s
Queue pointers	bold	eh.wptr
Memory locations	bold	MemStart
Instruction codes	italic	<i>stopch</i>
Command messages	italic	<i>CPoke</i>
occam program notation	typewriter	ChanOut

Table 1.1 Font conventions

References

This book is divided into three parts with each part having a number of chapters, sections and subsections. Figures and tables have reference numbers tied to relevant chapters of a particular part of the manual. Unless otherwise stated, all references refer to those within the current part of the manual.

Transputer product numbers

Product numbers take the following form:

IMS abc-xyyz

IMS = INMOS company identifier

a = Product group

T = Transputer
C = Communications peripheral
S = general software
D = Development software
F = Application software
B = Motherboards and TRAMs

b = Unique 3 or 4 digit product identifier

e.g. 9000 = 32 bit transputer with on-chip floating point unit.

c = Revision code

This is not present on all products.
Product traceability is guaranteed by a separate lot number found elsewhere on the package.

x = Package type

G = PGA
P = Plastic DIL
S = Ceramic DIL
J = PLCC
F = Ceramic QFP
X = Plastic QFP
N = Ceramic LCC
E = Plastic SOJ

yy = Speed variant

z = Specification

S = Commercial 0°C to +70°C
E = Extended -55°C to +125°C
I = Industrial -40°C to +85°C
M = Mil Std 883C -55°C to +125°C

(Note: **x**, **yy** and **z** apply to product groups T and C only.)



Part 1

IMS T9000 Product Family Overview

1 Introducing the IMS T9000

The IMS T9000 is the latest member of the transputer family and is designed for embedded systems and high performance computing applications.

INMOS has used advanced CMOS technology to integrate a 32-bit integer processor, a 64-bit floating point processor, 16 Kbytes of cache memory, a communications processor and four high bandwidth serial communications links on a single IMS T9000 chip.

The IMS T9000 transputer excels in real-time embedded applications, delivering exceptional single processor performance and scaleable multiprocessor capability.

There is extensive, industry standard software support for all members of the transputer family; this includes high-level language compilers, systems software (such as real-time operating systems) as well as an extensive range of development tools.

1.1 Performance

It is essential that any microprocessor family designed for the embedded system market provides the required performance at low cost.

The transputer family includes a 16 bit processor, a range of 32 bit fast integer and IEEE floating point processors and now, the highest performance member of the family, the IMS T9000. These are all designed to make it easy to design low cost, high performance systems.

- **Single processor performance:** the IMS T9000 transputer boasts exceptional single processor performance; the new superscalar pipelined CPU is capable of a peak performance of 200 MIPS and 25 MFLOPS.
- **Real-time performance:** the IMS T9000 offers sub-microsecond interrupt response and context switch times, making it ideal for high performance real-time systems.
- **Communications performance:** the four IMS T9000 communication links provide a total of 80 Mbytes/second bidirectional bandwidth.
- **Multiprocessor performance:** the interprocessor communications architecture and multiprocessing architecture gives scaleable performance – the ability to increase the performance of a system by adding more processors.
- **Usable performance:** the IMS T9000 implementation makes it easy for compilers to fully exploit the superscalar performance using a range of industry standard programming languages.
- **Price/performance:** the IMS T9000 offers supercomputer performance at an embedded systems price.

1.2 Multiprocessing

For applications that demand performance that single processors cannot provide, the IMS T9000 has complete on-chip support for multiprocessing:

- **Hardware scheduler:** the transputer architecture includes instruction level support for the creation and scheduling of any number of concurrent processes
- **Inter-process communication:** the transputer instruction set includes instructions for communicating between concurrent processes. The same instructions are used to communicate between processes running on a single transputer and between processes running on separate transputers.

- **Inter-processor communication subsystem:** the presence of a dedicated communications processor which operates concurrently with the main processor, makes interprocessor communications flexible and efficient. The integration of the communications system on-chip makes it easy to write programs for multiprocessor systems and eliminates the need for communications hardware such as shared memory or bus arbitration.
- **System control and monitoring:** all the IMS T9000 transputers in a system can be initialized, loaded with code and monitored for errors through a completely independent control communications system.

1.3 Communications support devices

The IMS T9000 transputer is complemented by a range of communications peripherals that extend the communications capabilities of the IMS T9000. The IMS C1XX family ensures that any size of IMS T9000 system can be constructed, connecting first generation and second generation transputers and providing an interface to the outside world.

- **IMS C104 packet routing switch:** the IMS C104 is a complete routing switch on a single chip. The IMS C104 connects 32 links to each other via a 32 by 32 way, non-blocking crossbar switch with sub-microsecond latency. This allows simple, fast communication between IMS T9000 transputers that are not directly connected. Multiple IMS C104s can be connected together to make larger and more complex networks, linking any number of IMS T9000 transputers, or any other devices that use the link protocol.
- **IMS C100 system protocol convertor:** the IMS C100 system protocol convertor converts between the first generation transputer links and control signals and the new IMS T9000 protocol. The IMS C100 provides an inter-networking solution for transputer systems, allowing networks to be constructed using the optimum mix of transputers to satisfy processing power, communication bandwidth and system cost.
- **IMS C101 link adaptor:** the IMS C101 link adaptor interfaces between IMS T9000 links and external systems such as buses, peripheral devices and even other microprocessors.

1.4 Software

The success of any microprocessor is determined as much by the quality of its software development tools as by any other feature.

INMOS has over a decade of experience in developing software tools for transputers and for multiprocessor systems. The range of compilers and powerful development tools support all the requirements of software developers.

- **Compiler toolsets:** for fast time to market, and to satisfy the diverse programming requirements of different application areas, the transputer can be programmed in one, or a mixture of the following languages:
 - ANSI C, an IMS T9000 optimized version of the INMOS externally validated ANSI C toolset.
 - C++
 - OCCam 2, a language designed by INMOS for efficient implementation of highly parallel systems.

Each compiler is packaged as a separate Toolset product containing a complete set of development tools for programming transputer systems. Designed to exploit the architectural features/optimizations of the transputer, the toolsets offer a leading development environment for em-

bedded systems developers whether working on single or multiprocessor target systems. Toolsets are available for industry standard Sun 4 and IBM PC compatibles (DOS 5, 386 minimum specification).

- **INQUEST:** targeting customer requirements for debugging and performance maximization, INQUEST provides an advanced software development environment supplementing language toolsets. INQUEST contains debugging and profiling tools using windowing graphical user interfaces.
- **System software:** the transputer incorporates in hardware many of the features normally offered by real-time executives. The efficient hardware operation offers significant performance gains over equivalent microprocessor plus real-time executive combinations. Real-time executives are available for the transputer, such as products from CHORUS Systems including distributed Unix SVR3.2.

This impressive array of development tools, industry standard compilers and system software satisfies the demands of the embedded systems market. It also ensures that the user can benefit from a significant improvement in the critical time to market.

1.5 Applications

The transputer family provides unprecedented price/performance solutions for a wide range of embedded systems applications.

The IMS T9000 transputer has been specifically developed to satisfy the requirements of three segments of the embedded systems market:

- **Imaging:** the imaging market comprises applications that involve the generation, manipulation and transmission of image data. Such applications include:
 - Laser printers
 - Graphics systems
 - Image processing systems
 - Industrial inspection systems
 - Robotics
 - Scanners
- **Embedded computing:** the embedded computing market comprises applications that are run within a computer environment and add overall performance and functionality to the computer system. Such applications include:
 - Application accelerators: (graphics, numerical, scientific, DTP)
 - Disk arrays and high performance file servers
 - Databases
 - X terminals
 - Factory automation
- **Supercomputers:** the supercomputers market targets heavy numerical algorithms to run with considerable compute resource. The IMS T9000 and IMS C104 form an environment with scalable high performance distributed computing power. Applications include:
 - Geophysical data analysis

- Simulation and modelling
- CAD
- High performance graphics
- **Communications:** the embedded communications market can be segmented into two main areas that require high performance microprocessors. These are:
 - Networking: low cost LAN interfacing – FDDI, Ethernet; inter-networking systems – bridges, gateways and routers.
 - Packet switching systems.

The IMS T9000 transputer is highly applicable to the communications market due to its integrated architecture combining high performance CPU and communication links with a packet based protocol. The IMS C104 packet routing switch has been designed to support the IMS T9000, and is useful in a range of telecommunications switching applications.

The transputer family provides a range of price/performance solutions for all the above applications.

2 The IMS T9000 transputer

The IMS T9000 is part of a broad range of 16 and 32 bit microprocessors. As well as providing high performance processing, they are designed to be simple to use and enable the construction of low cost systems. Transputers include functions to enable multitasking on a single processor and the building of multi-processor systems.

2.1 Overview

The IMS T9000 integrates a high performance central processing unit (CPU), a 16 Kbyte cache, communications system and other support functions on a single chip. The main functional blocks of the IMS T9000 are shown in figure 2.1. The function of each of these is outlined below, more details will be found in the following sections.

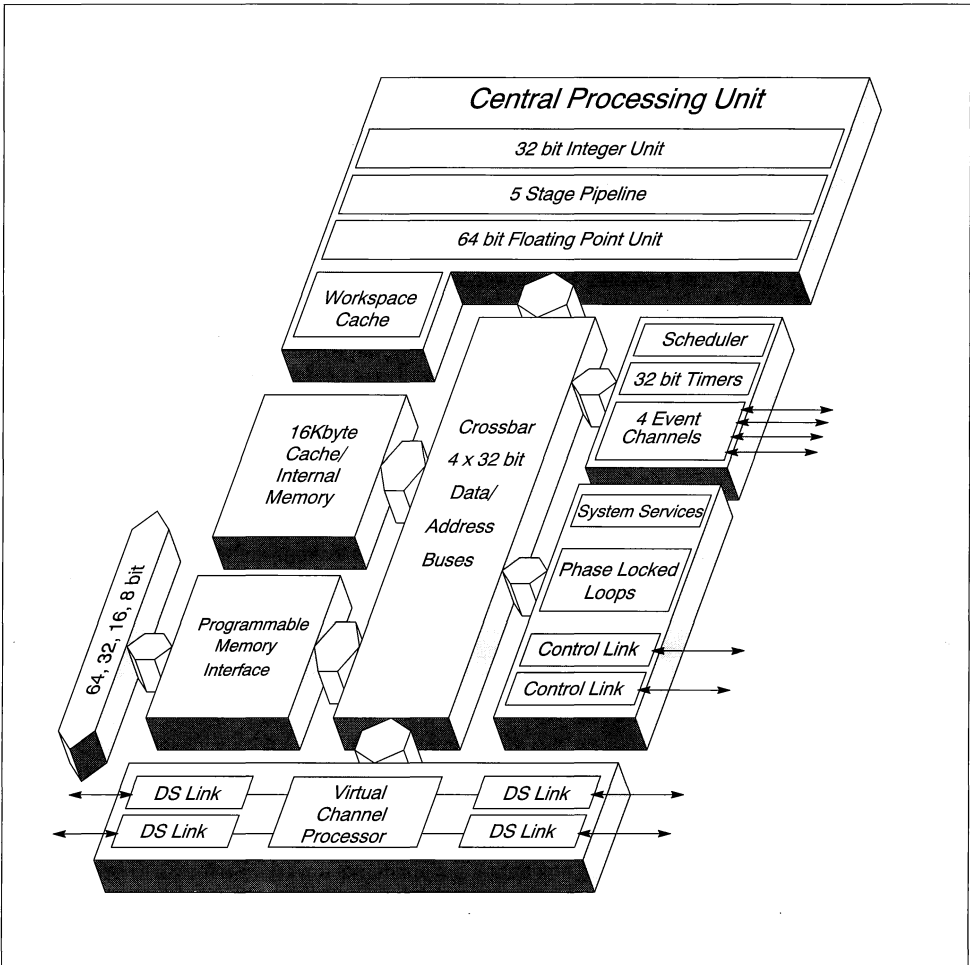


Figure 2.1 Block diagram of IMS T9000

2.1.1 Processor

The IMS T9000 CPU contains a 32 bit arithmetic and logic unit (ALU) and a 64 bit floating point unit (FPU). The FPU operates on 32 and 64 bit floating point numbers as specified by the IEEE 754 standard. The

CPU also includes instructions for byte and half word operations. The CPU uses 32 bit linear addressing and can address up to 4 Gbytes of memory.

The instruction set is designed for efficient execution of compiled code and there is a wide range of language compilers available for the transputer including a Plum-Hall validated ANSI C compiler, OCCAM and C++ compilers. These are complemented by a full set of software tools for developing and debugging programs for single transputers and networks of transputers. In addition there are a number of system level software products, such as real-time kernels and distributed operating systems.

The transputer includes hardware support for scheduling processes and performing communications. These operations are directly supported in the instruction set.

The IMS T9000 can run code in protected mode. In this mode all memory accesses are made through a memory management unit which checks and translates addresses before using them to address the memory system. Further, a restricted subset of the full instruction set may be executed, preventing protected code from executing privileged instructions.

There is improved support for error handling over earlier transputers; errors can be trapped and handled independently for each process in addition to the global error handling provided previously.

2.1.2 Hierarchical memory system

The IMS T9000 includes a 16 Kbyte unified cache to provide single cycle access to instructions and data. The cache provides a peak bandwidth of 200 Mwords/s. The CPU also includes another small cache for the most frequently used local variables of a program which provides an additional 150 Mwords/s of memory bandwidth.

The external memory interface is highly programmable, allowing large memory systems, containing different types of devices, to be built with little or no external logic. There are four independent sets of memory control signals simplifying the use of different device types in the same system. The memory interface operates with 8, 16, 32 or 64 bit wide devices. The maximum data transfer rate across the memory interface is 50 Mwords/s.

2.1.3 Communications system

An important issue in multiprocessor system design is the communications architecture. To achieve efficiency and ease of use, communications must be properly integrated into the entire processor architecture.

The transputer hardware and instruction set provides simple and efficient communications between processes and between processors. Both internal and external communications are handled identically, using the same source code and machine instructions.

To support interprocessor communications, there is a complete communications subsystem on chip. This includes four 100 Mbits/s full-duplex, serial communication links each with its own pair of direct memory access (DMA) channels. The links can be directly connected between transputers with no external buffering or other glue logic. The use of serial links simplifies routing of links on circuit boards and the interconnection of boards in a system. A communications processor, which manages all link communications, operates concurrently with the main CPU so that data transfers do not adversely affect CPU operation.

The communications subsystem also includes four 'Event' channels. As well as acting as interrupt inputs, these can be used, as inputs or outputs, for more general synchronization and signalling.

2.1.4 Multiple internal buses

To support the high degree of concurrent operation on the IMS T9000, and to maintain the high internal data rates required, there are four sets of 32 bit address and data buses internally. These provide multi-port access to the on-chip cache from the various functional units of the IMS T9000.

2.1.5 Control system

The control system handles most of the general facilities necessary for the operation of the IMS T9000 transputer including: initialization and configuration; reporting of errors; and resetting.

The control system comprises a control unit and a pair of control links. The control links are in addition to the four communication links, and are provided for system control and monitoring independent from program execution. Initialization and booting of the processor can optionally be done through these links. The control links are connected in a chain which allows a control master at the head of the chain to communicate with, control and monitor, any device in the network through this fully independent communications system.

The IMS T9000 transputers which do not boot from ROM are generally connected to a control processor, possibly another IMS T9000. A high-level protocol is defined for the controlling network to allow the controlling process to issue commands to, and receive responses from, devices in the control network. The control unit handles commands and issues responses to the controlling processor.

There is also a reset input – however, as the IMS T9000 includes on-chip power-on reset circuitry, external reset logic may not be required in an embedded control application.

2.1.6 Clocks

Two on-chip phase locked loops (PLL) generate all the internal high frequency clocks from a single clock input (5 MHz), simplifying system design and avoiding problems of distributing high speed clocks externally. The PLL's require a decoupled power supply for satisfactory operation. The decoupling is performed externally by connecting a 1uF ceramic capacitor.

The processor internal clock rate is variable in discrete steps. The clock rate at which the IMS T9000 runs is determined by the logic levels applied on the three processor speed select lines.

2.2 The transputer architecture

An important design decision was that transputers should be programmed in a high-level language. The instruction set has, therefore, been designed for simple and efficient compilation. The instruction format eliminates redundancy ensuring maximum performance through fast internal datapaths. Variable length instructions are chosen to give a compact representation of the operations most frequently occurring in programs.

The CPU of the IMS T9000 contains three registers (**Areg**, **Breg** and **Creg**) used for expression evaluation, which form a hardware stack. Loading a value into the stack pushes **Breg** into **Creg**, and **Areg** into **Breg**, before loading **Areg**. Storing a value from **Areg** pops **Breg** into **Areg**, and **Creg** into **Breg**. Similarly, the FPU includes a three register floating-point evaluation stack. When values are loaded onto, or stored from, the stack the floating-point registers push and pop in the same way as the **Areg**, **Breg** and **Creg** registers. Analysis of a large number of programs, shows that 3 registers provides an effective balance between code compactness and implementation complexity.

A separate floating point evaluation stack is provided, consisting of **FPAreg**, **FPBreg**, and **FPCreg**. The floating point evaluation stack behaves in a similar way to the integer evaluation stack.

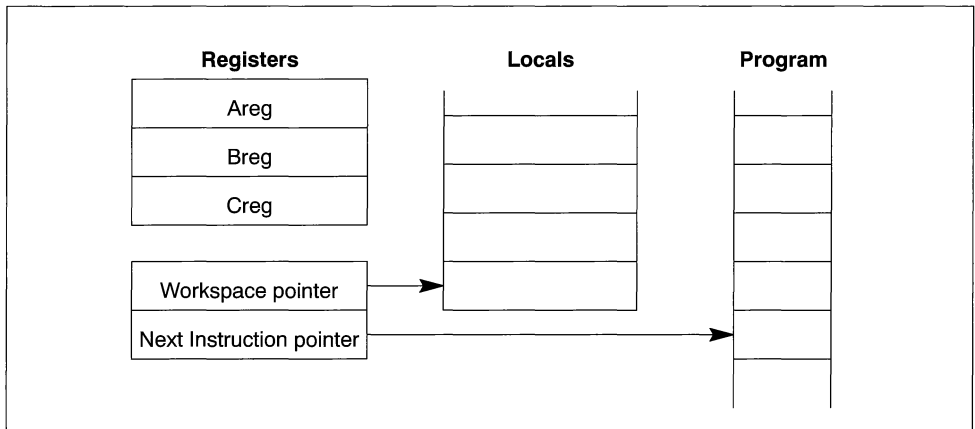


Figure 2.2 Processor registers and memory

The transputer has two other registers used when executing code. These are:

- The instruction pointer which points to the next instruction to be executed.
- The workspace pointer which points to an area of store where local variables are kept. This area is also used as a stack for procedure calls, etc.

The addresses of floating-point values are formed on the CPU register stack, and values are transferred between the addressed memory locations and the FPU register stack under the control of the CPU.

Most transputer functions use the contents of these stacks, and most instructions reference the stacks implicitly. For example the *add* instruction adds the top two values in the CPU stack, leaving the result on the top of the stack. The use of a stack reduces the need for instructions to specify the location of their operands, which reduces the size of instructions and hence of compiled code. Transputer object code is typically 70% of the size of equivalent CISC code.

2.3 Support for concurrent processes

Most computers have the ability to effectively run several user tasks or processes concurrently. These processes are created and scheduled by the host operating system. The operating system kernel provides the ability for processes to communicate with the operating system and with each other.

Every transputer includes a hardware scheduler with the ability to execute many software processes at the same time, to create new processes rapidly, and to perform communication between processes within a transputer and between processes on different transputers. All of these operations are integrated into the hardware and instruction set of the transputer and are very efficient. Further details of the transputer's scheduling mechanism will be found in section 5.

2.4 Pipelined, superscalar implementation

To increase the execution rate of the transputer instruction set, the IMS T9000 is able to issue several instructions per cycle. A superscalar pipelined architecture was designed which implements the same high-level architecture as the IMS T805 but with much higher performance.

Some recent implementations of pipelined and superscalar microprocessors have required very careful programming to obtain the claimed performance. They require that instructions are presented to the pipeline in a sequence that will keep the processor busy. This makes developing effective compilers very diffi-

cult, often forcing programmers to resort to assembly code to achieve the required performance. This puts the burden of arranging the correct sequencing of instructions on the programmer, adding to the development time and hence costs of a product.

The IMS T9000 incorporates an instruction grouping mechanism which scans the instruction stream, collating groups of instructions, to achieve optimum loading of the CPU pipeline. This hardware grouper relaxes the requirement on compilers or users, presenting instructions in particular sequences to obtain peak performance.

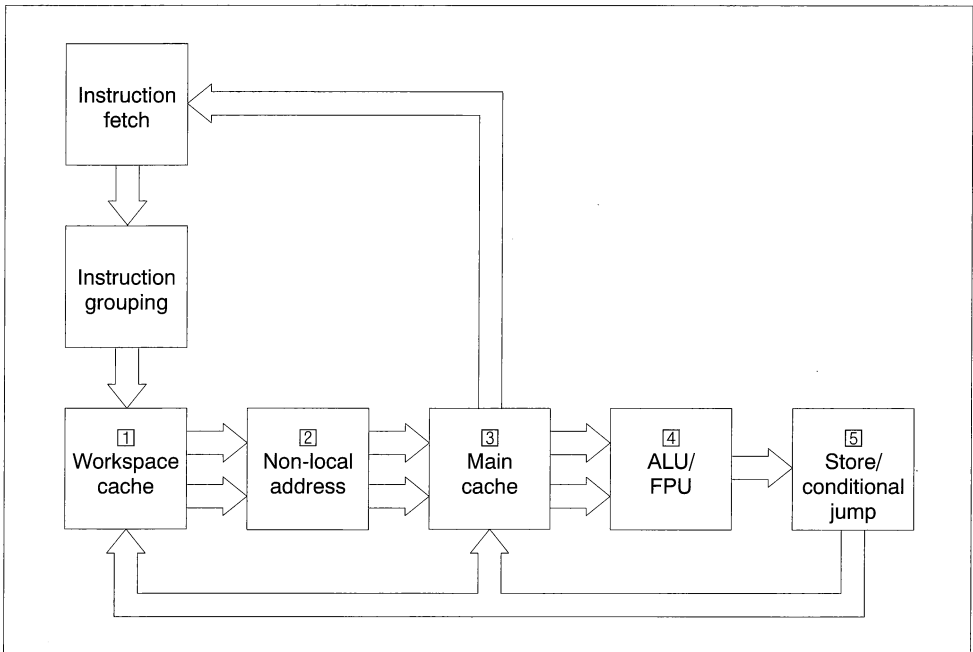


Figure 2.3 Block diagram of grouper and pipeline

The details of the IMS T9000 pipeline are transparent to the programmer. The processor appears to be the simple transputer architecture described above and straightforward code written for that programming model will get nearly the best performance out of the processor. INMOS' optimizing compilers for the IMS T9000 take account of the internal architecture to generate even more efficient code.

2.4.1 The pipeline

Instructions are executed in a five stage pipeline: the first stage can fetch two local variables; the second stage can perform two address calculations, for accessing non-local or subscripted variables; the third stage can load two non-local variables; the next can perform an ALU or FPU operation; and the final stage can do a conditional jump or write.

A conventional pipeline is designed to allow several instructions to be executed simultaneously; different parts of each instruction being handled in different stages of the pipeline. In order to allow multiple instructions to be *issued* per cycle (as well as multiple instructions being executed in each cycle) the IMS T9000 does not simply send a sequence of instructions through the pipeline but has hardware which assembles groups of instructions from the instruction stream. These groups are chosen to make the best use of the available hardware and one group can be sent through the pipeline every cycle. Instructions are put into groups in the order that they arrive at the CPU; dependencies within the group are handled automatically by the pipeline.

The grouper can be thought of as a hardware optimizer; it recognizes commonly occurring code sequences that the processor can execute efficiently. The design of the grouping mechanism and the pipeline is based on analysis of the code typically generated by high-level language compilers.

2.5 Hierarchical memory system

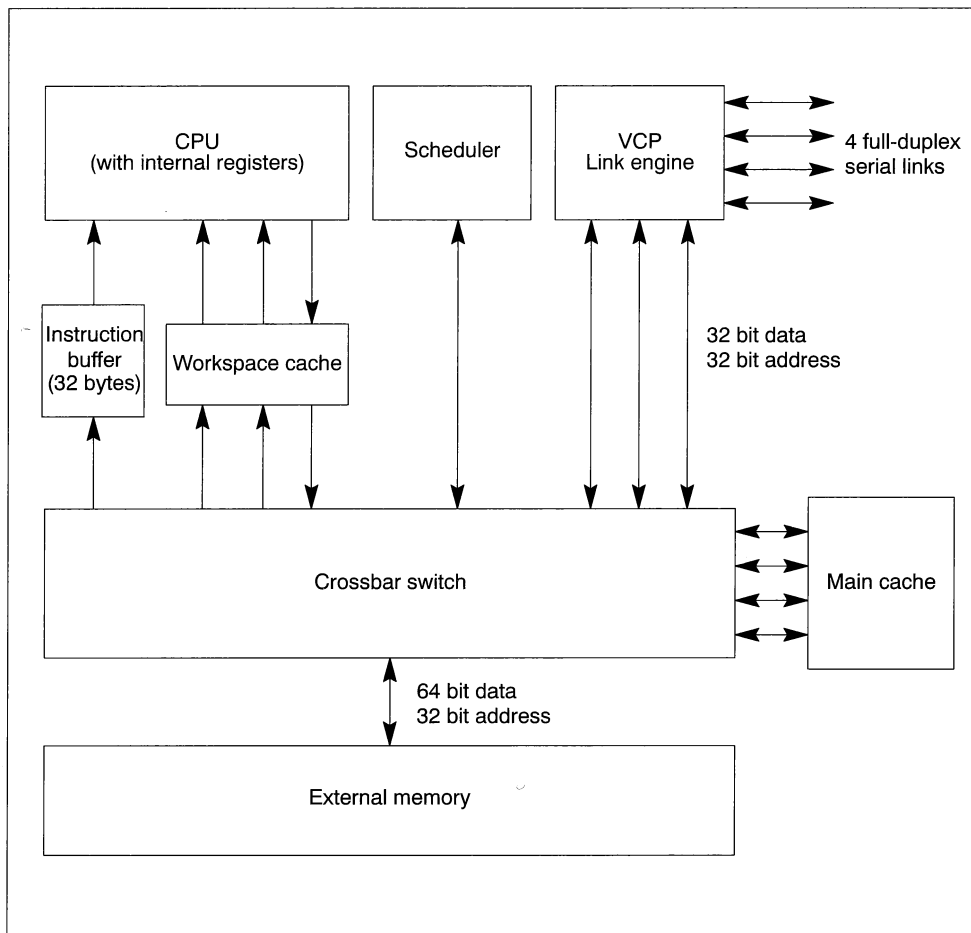


Figure 2.4 IMS T9000 hierarchical memory system

The IMS T9000 has a complete, hierarchical memory system providing fast and efficient access to data and instructions. There are two separate caches on chip, a general purpose unified (code and data) cache and a small cache for local variables.

These caches can provide fast, multi-ported access to data because they are on chip. They also reduce the number and frequency of accesses to external memory, allowing lower cost, external components to be used without degrading performance. Finally, because the majority of external memory accesses will be cache refills (and therefore multiple word reads and writes) fast memory access methods, such as page mode, can be used.

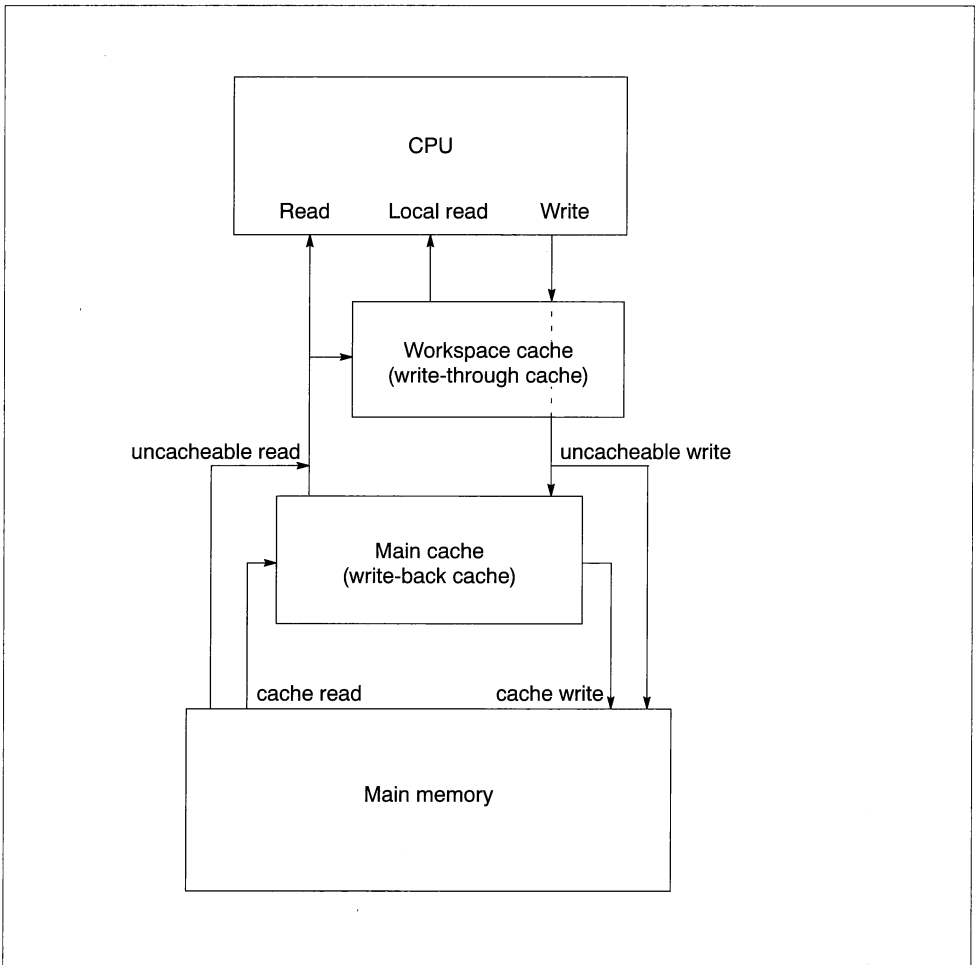


Figure 2.5 Levels of caching

2.5.1 Main cache

The main cache consists of four independent banks, each containing 256 lines. Each line holds data from four consecutive words (16 bytes) in memory. An access can be made to every bank on every cycle which, with the multiple internal buses, means there is a very high bandwidth between the cache and different functional units within the IMS T9000.

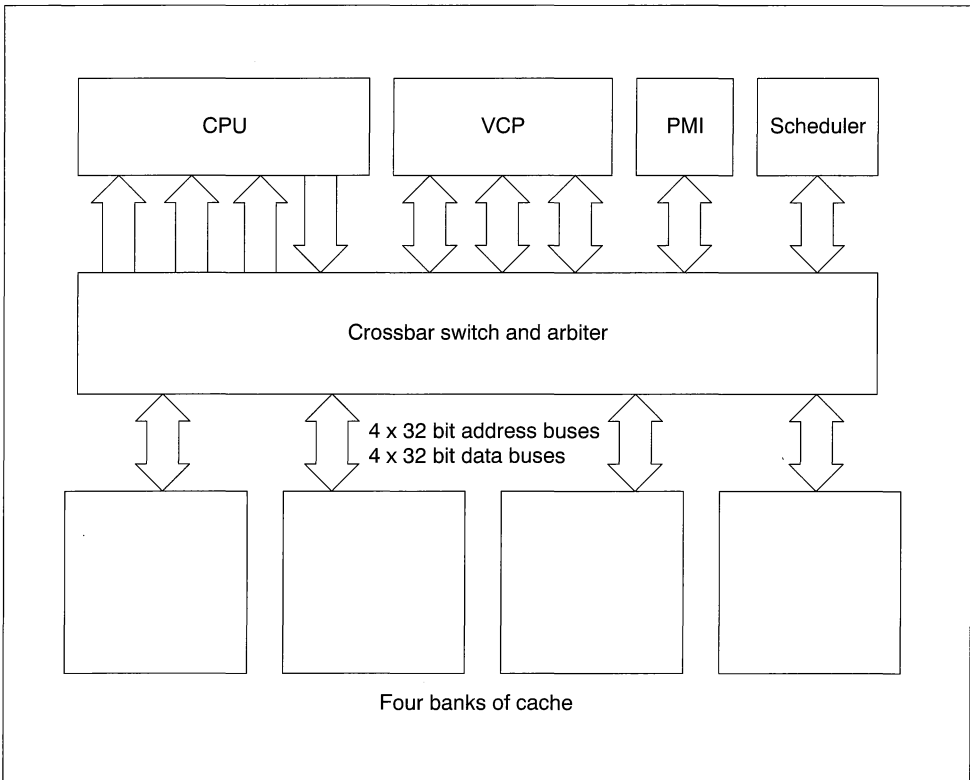


Figure 2.6 Diagram of four banks of cache

The four cache banks are accessed by a number of different functional units in the IMS T9000, some of these units have multiple ports into the cache. To allow four simultaneous reads and writes to take place in each cycle, there are four sets of address and data buses. An arbiter controls access from the various functional units to the cache banks.

Cache operation

Each of the four banks is addressed by a quarter of the memory space. This division of the address space is done using bits 4 and 5 of the address, the bottom four bits are used to select a byte within a line. Each line consists of: 16 bytes of RAM for the data; 26 bits of associative memory which holds the address of this line of data; and two control bits to indicate if the line is valid and if it has been modified since it was read in (is 'dirty'). When a memory access is made, the address is checked against the contents of the appropriate bank. If the address is present (and the line is valid) then the access can go ahead, reading or writing the data in a single cycle.

A cache refill engine ensures that there is always one empty line available. Then, if a requested address is not in the cache (a 'cache miss'), the four words containing the data are read from memory into the empty line. The refill engine then has to ensure that a new empty line is created. It does this by choosing a line at random and, if the data has been modified since it was read into the cache, writing it out to memory. The line is then marked as invalid, i.e. empty and available for use. This is known as 'early write-back' as it writes the chosen line out to memory before a cache miss occurs.

The reading and writing of cache lines takes advantage of any fast memory access methods that are available (e.g. 64 bit wide accesses or page mode DRAM).

Use as on-chip RAM

At reset, the cache behaves as 16 Kbytes of on-chip RAM, enabling the IMS T9000 to be used with no external memory. There may be many applications where a number of transputers are used, each requiring little or no external memory – used in this way the IMS T9000 provides extremely high performance (single cycle memory reads and writes) combined with extremely low cost (possibly no external components except a clock). Starting up in this mode provides compatibility with earlier transputers which have a fixed amount of on-chip RAM. It also makes it possible to test the hardware of a new transputer system as it is known that there is 16 Kbytes of working RAM which can be used by test software.

During the initialization of the IMS T9000 the cache may be programmed to behave as 16 Kbytes of cache, as 16 Kbytes of RAM, or as half cache and half RAM. This can be very useful when certain data or code, e.g. an interrupt handler, must be accessed quickly and in a more deterministic way than a cache provides. The remaining 8 Kbytes of cache will be large enough to achieve high performance.

2.5.2 Workspace cache

The workspace cache holds a copy of the first 32 words of procedure stack and workspace. It is triple ported, allowing two reads and a write in every cycle. The workspace cache allows local data to be accessed without going outside the CPU, effectively giving zero cycle access and reducing the load on the main cache and external memory. It also means that the pipeline can do four data reads (as well as an instruction fetch) in each cycle: 2 from the local cache and 2 from the main cache.

Because local variables can be accessed quickly, they can be read in the first stage of the pipeline and can then be used for non-local address calculations in the next stage. The workspace cache is write-through; whenever data is written into the local cache it is also written to the main cache. This means there is no overhead for flushing the cache on interrupt or context switch.

The workspace cache is part of the processor pipeline and, in many ways, it is equivalent to the general purpose register set found on other microprocessors, providing fast access to frequently used data. To make use of this architecture, the INMOS ANSI C compiler recognizes the 'register' keyword and places those variables lower in the function's workspace so they are more likely to be cached.

3 Simplicity of system design

Many features of the IMS T9000, as with the original transputer range, exist to simplify the user's design task and to reduce the amount of support hardware and software that is required. This means that designers can spend more time working on their application and less time worrying about details. Using transputers results in smaller, simpler designs, easier system debugging, faster time to market and lower system cost. Some of these features and their benefits are outlined below.

3.1 Single 5 MHz clock input

All transputers, no matter what the processor speed, and all support devices require only a single 5MHz clock input; on-chip phase locked loops generate all the high frequency internal clocks required for the processor and links. Transputer link operation is independent of clock phase, thus differences in the clock phase between devices is not important. This means that each processor can have a local clock or a single system clock may be used for multiple devices.

This simplifies system clock generation and distribution, especially where multiple transputers are used. The use of low frequency signals around a system can be particularly important in electrically noisy environments such as industrial control systems.

3.2 Programmable memory interface

The first generation of 32 bit transputers have a memory interface which can be programmed to generate all the timing signals required by a memory system, meaning that little external logic is required to build a complete system.

The IMS T9000 takes this idea further by providing greater functionality and flexibility. The IMS T9000 programmable memory interface (PMI) provides complete support for DRAM including multiplexing of row/column addresses, refresh, and page mode accesses. It is possible to connect many Megabytes of external DRAM with no external logic. The amount of memory which can be connected directly is limited only by capacitive loading; larger amounts of memory will require only the addition of buffering on the address and data lines.

The IMS T9000 memory interface automatically exploits any fast access modes for the memory system. For example if 64 bit wide DRAM is used then an entire cache line can be read in two memory operations. If page mode DRAM is available, then reads or writes with the same row address will be done using page mode, greatly reducing the cycle time. This will always be used for cache line reads and writes, where four consecutive words will be transferred, but it will also work for any set of reads and writes from the same page.

In addition to supporting fast DRAM, the IMS T9000 will also efficiently interface to other devices, such as SRAM, ROM or memory mapped peripherals. The PMI on the IMS T9000 divides the address space into four banks (not to be confused with the four banks in the main cache). Each bank provides separate decoding and timing control, generating all the signals needed for the device types in that bank. The address range, timing, memory type and bus width can be programmed independently for each bank. There is an additional preset bank for slow, byte-wide ROM. This is intended for systems where the processor is booted from ROM. Only memory reads can be done from this bank.

The parameters for the memory interface are programmed into a number of configuration registers. A software tool is provided in the development toolsets to simplify the task of designing with the PMI. This tool can be used interactively to describe the parameters for each memory bank. It then produces an output file which can be used by other parts of the development system for initializing and loading transputers. The program also produces timing diagrams and descriptions which can be used in documenting the system design.

3.3 Control links and configuration

The IMS T9000 has a pair of control links. One is used by the IMS T9000 for receiving commands and sending status information, the other provides a cascade connection so that some or all devices in a

system can be daisy-chained together. The control links use the same link protocol as the IMS T9000 data links and provide a control network which is completely independent of the normal data communication network.

The control links have through routing hardware so that the controlling processor (possibly an IMS T9000) appears to have a direct connection to every device in the system.

The control links are kept totally separate from the links used for program communication in a system. A program running on the IMS T9000 cannot send messages down the control links. The separation of control and data links ensures that the control links are reliable and secure. For extra reliability, they can be run at a lower bit rate.

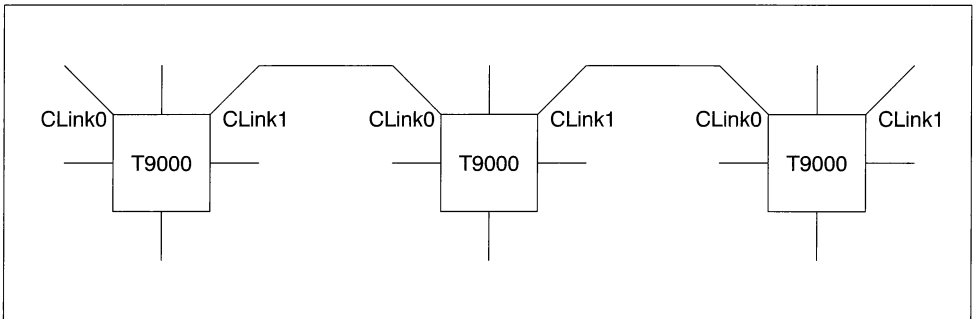


Figure 3.1 Network of control links

The control links provide an independent communication network which can be used to load code, do hardware debugging, monitor a running system for errors and perform diagnostic functions, both for a single IMS T9000 and a network.

Because of the great flexibility of the memory interface and the communications system of the IMS T9000 there are a number of configuration registers that need to be programmed. For all of these, the development tools will program the registers using high-level descriptions of the system. For example, as noted above, there is an interactive tool for developing configuration data for the PMI. Similarly, the communication system is set up using high-level language descriptions of the software and hardware networks.

There are two ways of programming the configuration registers: by writing to them from a program running on the IMS T9000 itself; or via a control link from the host system. The first method is used when the system is booted from ROM, for example in an embedded system. The second method can be used in a development environment or, in a multi-transputer system, where only one processor is initialized (or 'configured') from ROM and all the others are configured via their control links from that root processor. In both cases the IMS T9000 development tools will generate the data to be programmed into ROM or sent to the control link of a processor.

There are a number of stages of initializing and loading code onto the IMS T9000 after it has been reset. These are known as 'reset levels' and during the initialization process, every IMS T9000 must go through each level from complete reset, to having application code running. This can be done from ROM or through a control link. The toolsets allow ROM systems to be built with a single system ROM or multiple local ROM's.

3.4 Loading and bootstrapping

The transputer can also be bootstrapped in two ways: from code received down a link or from ROM. All INMOS development tools generate programs to be loaded by either method as required during development or in a production system.

There are a number of advantages to the ability to load code from a link. It greatly simplifies the development cycle – there is no need to keep programming new EPROMs with new versions of code (or use an

EPROM emulator); it can simply be loaded down a link. It simplifies testing of hardware – a transputer provided with the minimal essential external signals (5 volts, clock, etc) will be guaranteed to work; there is then 16 Kbytes of on-chip RAM in which to load test code. In a multiprocessor system, only the root processor needs to be booted from ROM – the others can be booted down a link with code contained in that system ROM. It is even possible to switch between ROM and link booting, in order to do field testing and diagnose faults in an installed system.

3.5 Examples

To show how simple it is to build systems using the IMS T9000, a few example block diagrams are given here. In the simplest cases these are almost complete circuit diagrams.

The first example (figure 3.2) is a complete working system using the IMS T9000's internal RAM as the system memory. The processor boots from ROM which contains the application software. This processor can communicate with other transputers or peripherals through its data links. It can be set to boot from ROM or from link for development and test purposes. The full 16 Kbytes of on-chip RAM is available for program workspace.

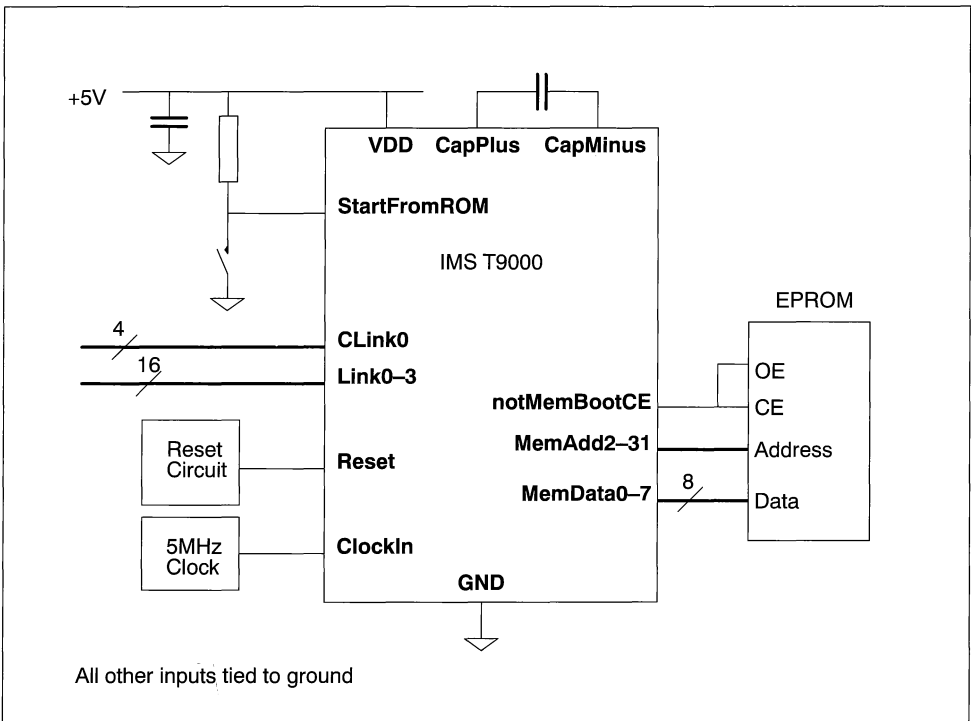


Figure 3.2 Complete IMS T9000 system with EPROM

Figure 3.3 shows how a low cost system can be built using a small amount of SRAM. This could be combined with ROM and peripherals for a low cost embedded application.

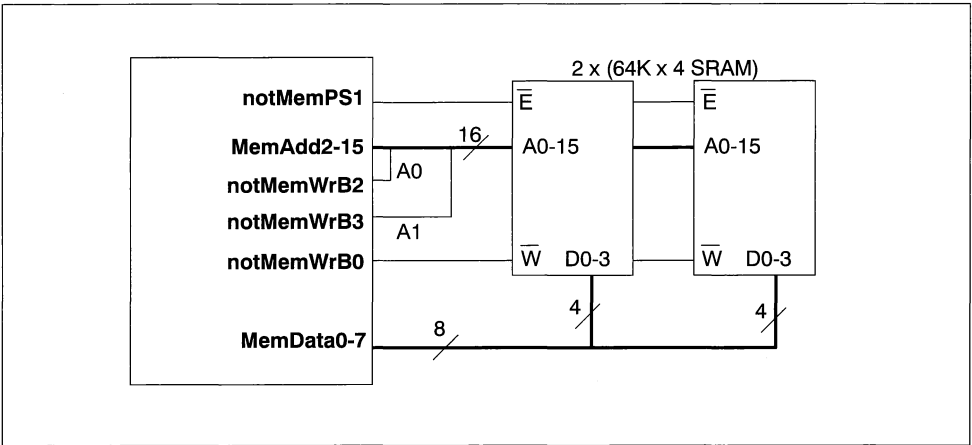


Figure 3.3 Low cost system with 64 Kbyte of SRAM

The third example in figure 3.4 shows how a large amount of DRAM can be connected to the IMS T9000 with no external logic for decoding, control signal generation or buffering.

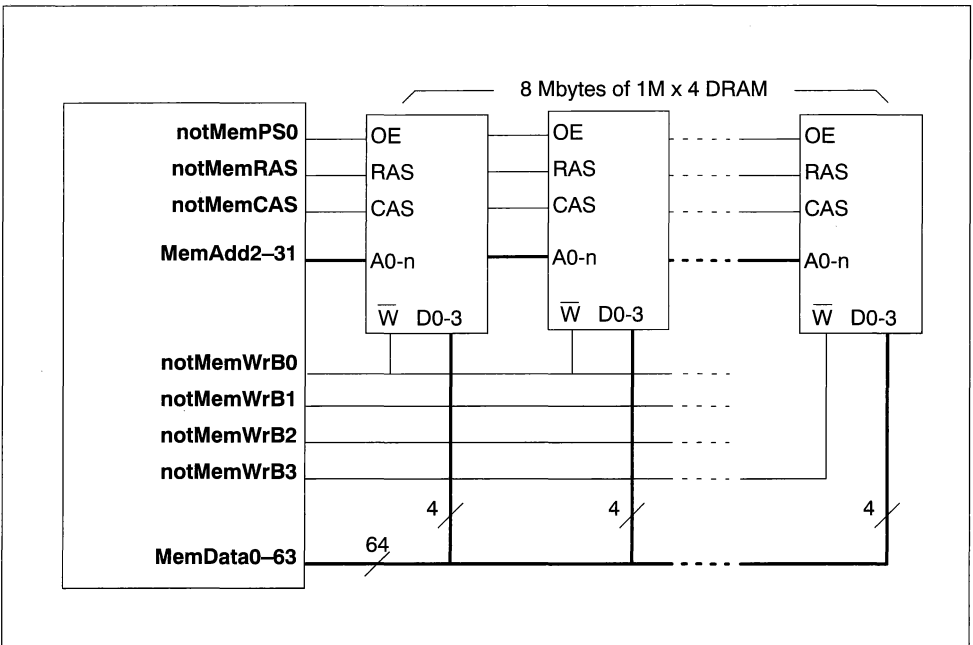


Figure 3.4 High performance system with 8 Mbytes of DRAM.

The memory in each bank is enabled by separate strobe signals so all of the above memory types could be combined on a single IMS T9000.

4 Protection and error handling

The IMS T9000 extends the error handling of earlier transputers to allow error conditions to be easily trapped and handled in software. It can run code in a protected mode where all memory accesses are checked and certain, privileged, instructions cannot be executed.

4.1 Error handling

The first transputers have only a global mechanism for trapping errors; stopping the entire processor when an error is detected. The IMS T9000 extends this to allow a trap handler to be associated with a process to provide more localized error handling (hence these processes are known as L-processes for local error handling). When an error occurs, control is transferred to the trap handler with information about the nature of the error and where it occurred.

The action of a trap handler will, in general, be dependent on the language or operating system being used and will be invisible to the applications programmer. Some languages may include support for user written error handlers. After taking the appropriate action, for example to report or correct the error, the trap handler can return control to the process which caused the error which can then continue execution. Each process can have its own trap handler, or one trap handler can be shared by several (or all) of the processes on the transputer.

4.2 Protected mode

The IMS T9000 can run code in protected mode. This is designed to allow run-time checking of programs written in 'unsafe' languages such as C and also to provide memory management. For example, C allows pointers to data or functions. Without checks for valid pointers, these could contain an illegal memory address such as: another process's data or code; a non word aligned address; or a function pointer which does not point to valid code. As no checks are defined in the language it is important to be able to check such accesses at run time, if needed.

The protection mechanism is intended to support software development and debugging, and programming secure systems. It protects the user's processes or tasks from each other and also protects an operating system kernel, or other run time support, from user code. Although code run in this mode is frequently referred to as a 'protected process', it is not the process which is protected but the rest of the world that is protected from errors in the process.

4.2.1 Protected mode processes

Any L-process can run a piece of code as a protected mode process (or P-process); the processor saves the state of the L-process and starts executing the P-process. The P-process is executed until control is returned to the L-process because of an error, protection violation or some other reason. It is important to realize that P-processes are not scheduled by the transputer's own scheduler – they only run under the control of a supervisor L-process. Any of the instructions or other events that might cause a P-process to be descheduled, will cause control to be returned to the supervisor. The relationship between a P-process and its supervisor is analogous to that between an L-process and its trap-handler. In both cases the processor can be thought of as swapping between the two pieces of code.

4.2.2 Executing illegal instructions

Because control is returned to the supervisor when a P-process attempts to execute a privileged or illegal instruction, it is possible to provide communication and other facilities to a P-process in a controlled way, but one which is invisible to the programmer. For example, the input and output instructions are privileged, so if a P-process attempts a communication then it will trap to the supervisor L-process. This L-process can examine the state of the P-process and, if the attempted communication is 'legal', perform the communication and return control to the P-process. The P-process will continue as if a normal communication had occurred.

There is also a *syscall* (system call) instruction which can be used by a P-process to explicitly request some action by the supervisor.

4.2.3 Memory management

When running in protected mode, all memory accesses are checked and translated. Each P-process can access four regions of memory. The size and base address of each region can be set, and each can have different protections. Each area can be given permission for code to be executed from it and for data to be read or written. For example an area of memory containing code would normally be marked with execute permission but write protected.

All addresses generated when the processor is running in protected mode are logical addresses. These are translated to physical addresses by combining the low order bits of the logical address with the high order bits from the control register for that region. The translation and checking is done in parallel with other address generation operations and so imposes no overhead on memory access time.

The IMS T9000's memory management can be used to implement swapping of memory to and from disk and relocation (although it does not support page-based virtual memory). This can be used to implement operating system kernels. It can also be used for 'stack extension'. All the instructions which move the workspace pointer are checked for a valid address after the operation. If it is found that the workspace address is no longer valid then a trap occurs, the supervisor process can then allocate more memory for the processes stack and restart it.

5 Support for multiprocessing

The requirement for processing performance in embedded systems is continuously increasing as control algorithms become more sophisticated and as systems become more complex. In the long term, the only solution to these ever increasing demands for performance is the use of multiple processors to perform independent co-operating system functions.

Transputers are the only microprocessors specifically designed to tackle the problems of building multiprocessor systems. There are advantages other than just performance to using multiple transputers in a system: it allows ease of system partitioning; it allows scaleable systems to be built, where more processors can be added as demand increases, or to provide the optimum balance of price versus performance. The communications facilities of the transputer family can also be used to build distributed systems where, for example, the processors are located near the equipment or components they control and use links to communicate with other processors in the system. In addition, transputers can be used to build high reliability, fault tolerant systems.

Fast interrupt response and process switch

In most embedded applications, there is a need for fast real-time response (both to external interrupts and for context switching in multitasking systems). The design of the IMS T9000 processor exploits the presence of the two on-chip caches by having only a small number of registers in the CPU. This means that there is little state to be saved when an interrupt or task switch occurs, so these operations are extremely fast. These types of operations are very efficient on the transputer because of the hardware scheduler.

The register stacks are duplicated so that, when a process running on the IMS T9000 is interrupted, the contents of the stacks do not need to be written to memory. This results in a sub-microsecond interrupt response. Furthermore, the duplication of the register stacks enables floating-point arithmetic to be used in an interrupt routine without any performance penalty.

5.1 The transputer model of concurrency

The model of concurrency and communication implemented by the transputer hardware is based on the ideas of communicating sequential processes. All the features for creating processes and communicating between them are accessible from any high-level language for the transputer and are implemented directly by the *Occam* programming language, see the *Occam 2 reference manual*.

5.1.1 Processes and channels

Each process can be regarded as a black box with internal state, which can communicate with other processes using communication channels. Each channel is a point to point connection between two processes. One process always inputs from the channel and the other always outputs to it. Communication is synchronized: the first process ready to communicate waits until the second is also ready, then the data is copied from the outputting process to the inputting process and both processes continue.

Each process starts, performs a number of actions and then terminates. An action may be a set of sequential processes performed one after another, as in a conventional programming language, or a set of parallel processes to be performed at the same time as one another. Since a process is itself composed of processes, some of which may be executed in parallel, a process may contain any amount of internal concurrency, and this may change with time as processes start and terminate. Ultimately, all processes are constructed from three primitive processes: assignment; input and output.

Within the context of C programming, transputer processes with different semantics may also be used, these are known as threads. A thread is a special case of a transputer process. Whereas processes have a purely message based interface, threads of a process share code, static data and heap space. This allows communication through techniques using shared memory objects such as semaphores, shared memory areas and shared variables.

5.1.2 Program structure

Figure 5.1 shows an example of a system constructed from three communicating processes. In this case there are separate processes to handle the external hardware (the screen and keyboard) and to execute

the main, application, process. This is a modular design – only the hardware handling processes have to be changed if the software is moved to a new environment, the same interface (the data sent and received on channels or ‘protocol’) can be presented to the application process. The keyboard handler can be interrupt driven, only being scheduled when a character is typed, the interrupts appearing as communications. The input and output processes can provide buffering and other filtering of the data, all of which is invisible to the main application process, which could even be placed on a separate processor. This use of separate processors need not just be for performance reasons but might be done, for instance, if there are a large number of peripheral devices which could be better handled by a low cost 16 bit transputer. One or more high performance transputers could then be used for the main computing processes.

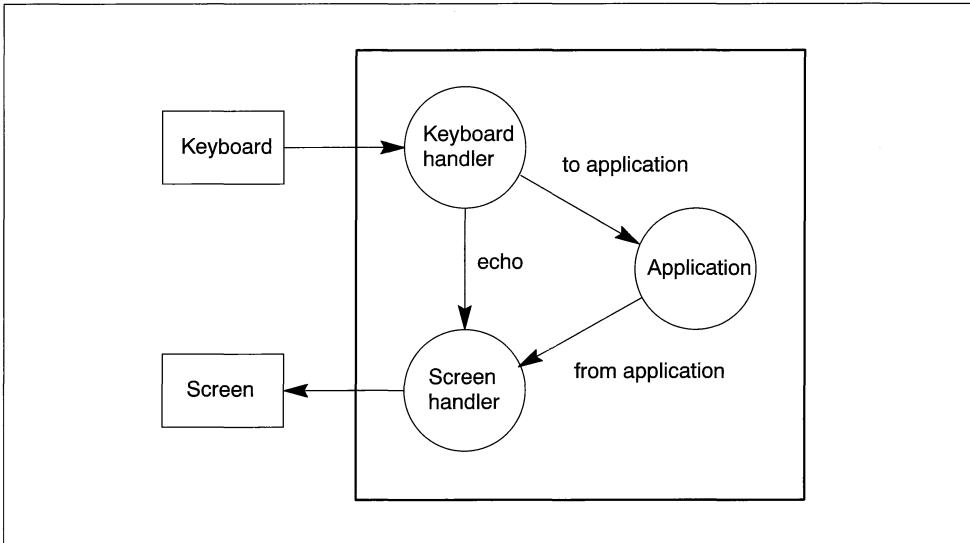


Figure 5.1 Processes and channels

Example

The code for creating parallel processes in C is very simple. For example, if the three processes in the example above are external functions, then the following code is all that is needed to run them in parallel:

```

#include <stdlib.h>
#include <channel.h>
#include <process.h>

/*
   declare externally defined functions
*/
extern keyboard_handler (Process *p, Channel *to_app, Channel
*echo);
extern screen_handler (Process *p, Channel *echo, Channel
*from_app);
extern application (Process *p, Channel *to_app, Channel *from_app);

/*
   declare pointers to process and channel data structures
*/
Process *kbd_p, *scrn_p, *appn_p;
Channel *to_app, *from_app, *echo;

```

```

/*
    allocate and initialize channel data structures
*/
to_app  = ChanAlloc();
from_app = ChanAlloc();
echo    = ChanAlloc();

/*
    allocate and initialize the process data structures
*/
kbd_p   = ProcAlloc (keyboard_handler, 0, 2, to_app, echo);
scrn_p  = ProcAlloc (screen_handler, 0, 2, echo, from_app);
appn_p  = ProcAlloc (application, 0, 2, to_app, from_app);

/*
    now run the three processes in parallel, this call
    will return when all three processes have terminated
*/
ProcPar (kbd_p, scrn_p, appn_p, NULL);

```

A more complete explanation of how parallel programs can be written for the transputer can be found in INMOS Technical Note 68, "Developing parallel C programs for transputers", which is included in *The Transputer Development and iq Systems Databook* (2nd edition).

The equivalent program in OCCAM would be:

```

CHAN OF BYTE to.app, from.app, echo :
PAR
    keyboard.handler (to.app, echo)
    screen.handler (echo, from.app)
    application (to.app, from.app)

```

5.1.3 Multiprocessor programs

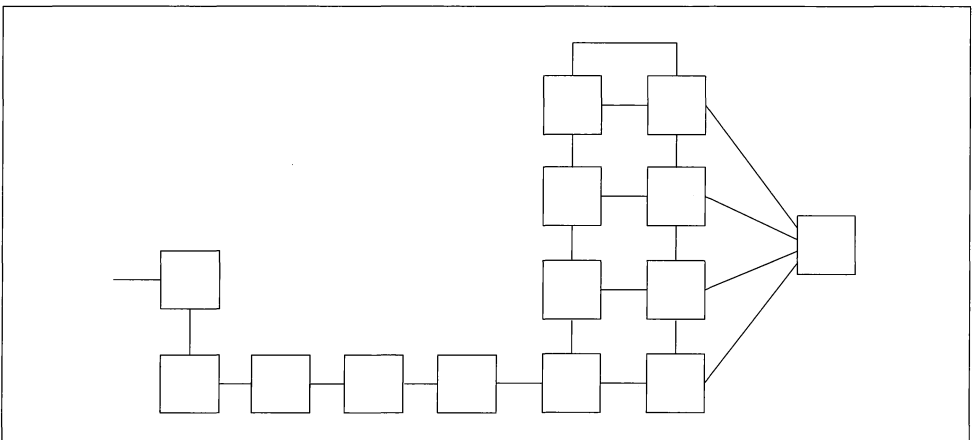


Figure 5.2 Transputers and links

Every transputer implements these concepts of concurrency and communication. As a result, the same model can be used to program an individual transputer or to program a network of transputers. Figure 5.2 shows a typical network of transputers connected by serial links. When a number of processes run on an

individual transputer, the processor shares its time between the concurrent processes, and channel communication is implemented by moving data within memory. When this programming model is used to program a network of transputers, each transputer executes the process, or processes, allocated to it.

Communication between processes on different transputers is implemented directly by transputer links. Thus the same program can be implemented on a variety of transputer configurations, with one configuration optimized for cost, another for performance, or another for an appropriate balance of cost and performance as illustrated in figure 5.3.

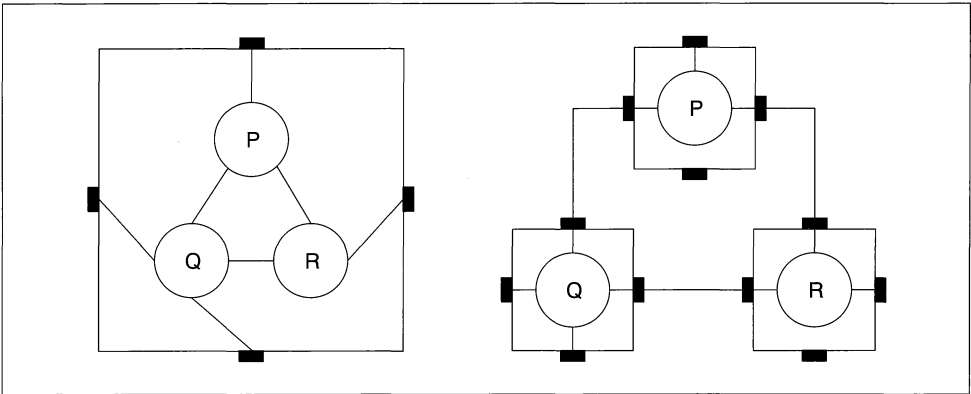


Figure 5.3 Mapping processes onto one or several transputers

5.2 Other models of concurrency

Although the transputer has direct support for concurrent processes which communicate via channels, it is possible to use the same features of the transputer to build other types of multiprocessor system or to support different scheduling models. The IMS T9000 includes a number of instructions for manipulating the transputer process queues; these make it simple to write real-time kernels, exploiting the efficient task switching of the transputer architecture. There are also instructions for ensuring that the data in the cache and in memory are consistent. These can be very useful when implementing a shared memory system.

5.2.1 Shared memory

In a shared memory system, a number of processors have some sort of common area of memory which they can all access. This has some advantages over the channel communication model, especially where very large amounts of data need to be shared or moved between processors. The transputer has hardware and software support for shared memory systems.

The PMI has a set of signals for controlling access to the external memory interface by an external device. This is primarily intended for use with a DMA based co-processor. It can also be used, with external arbitration logic, to allow all of the processors in a system to access the shared memory.

Alternatively, there may be a number of blocks of memory that can be switched into the memory map of different processors under software control. These blocks can be used for exchanging data and passing messages between processors. To synchronize the switching of these blocks of memory between processors, the ideal method is to pass messages over the transputer links; as the memory is switched to a processor's address space, it is sent a message from the previous user of the memory to inform it that it is now the new 'owner' of the memory. This allows large amounts of data to be moved from one processor to another but without the overhead of copying all of it over a link.

In any shared memory system, the use of a cache can be a problem. In the IMS T9000 there are instructions for forcing changed data in the cache to be written out to main memory and for marking data in the

cache as invalid so that it will be read from main memory. As the exchange of data is synchronized between processors, these instructions can be used to make sure that the correct data is in both the main memory and the cache of the processors involved.

It is also possible to mark banks of external memory to be 'un-cacheable'; data from that area of memory will never be put in the cache. This ensures that a number of processors or other devices which make random reads and writes of that memory will always get the most up to date data. In this case there must still be some synchronization of the memory accesses to make sure that information is not read by a processor until it has been written; again, this synchronization can be done over the transputer links.

5.3 Hardware scheduler

The IMS T9000 processor includes a hardware scheduler which implements the transputer model of concurrency. In many applications this will remove the need for a software kernel. However, the transputers own scheduling mechanisms can be accessed from software to provide efficient support for the implementation of standard real-time kernels.

At any time, a transputer process may be:

- Active*
 - Being executed.
 - Interrupted by a higher priority process.
 - On a list waiting to be executed.
- Inactive*
 - Ready to input.
 - Ready to output.
 - Waiting until a specified time.
 - Waiting on a semaphore.

The scheduler operates in such a way that inactive processes do not consume any processor time. The active processes waiting to be executed are held on a list of process workspaces. This is implemented using two registers, one of which points to the first process on the list, the other to the last. In figure 5.4, **P** is executing, and **Q**, **R** and **S** are active, awaiting execution.

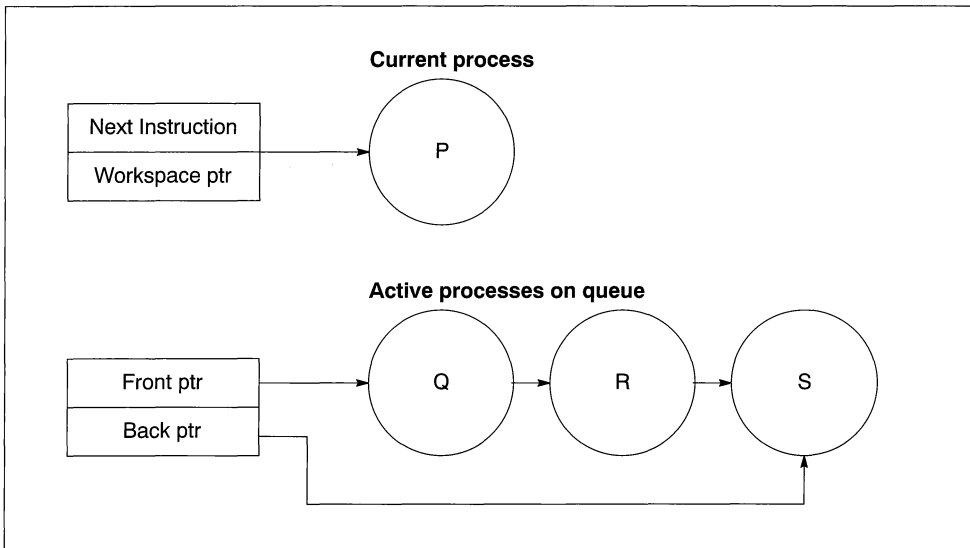


Figure 5.4 Transputer process queue

A process runs until it is unable to proceed because it is waiting to input or output, or waiting for the timer. Whenever a process is unable to proceed, its instruction pointer is saved in its workspace and the next process is taken from the list. Actual process switch times are very small as little state needs to be saved; it is not necessary for the processor to save the evaluation stack on descheduling.

5.4 Interrupts, events and timers

As well as process scheduling and communications, the scheduling hardware also supports simple handling of interrupts and timers. Any event that a process might need to wait for (whether it be a communication, an interrupt or a timeout) can be treated in the same way as a communication. For example, an interrupt handler simply has to wait for an input from a special channel which is mapped onto an interrupt ('Event') input. Because inputs are synchronized, that process will not proceed until the 'input' becomes ready, i.e. until there is an interrupt.

This makes interrupts on the transputer very easy to use. An interrupt handler is simply a process like any other waiting on an input from the interrupt 'channel'. This contrasts greatly with the traditional idea of an interrupt handler as something difficult which needs to use special instructions and be written in a very different way from other program code (usually in assembler).

The IMS T9000 has four sets of pins, known as 'Event' channels, which can be used for control and synchronization purposes. Each Event channel can be configured either as an input or an output. As inputs they can be used as interrupts, to cause a fast processor response to external signals. When an Event channel is configured as an output, the process outputting to it will be descheduled until the external device provides the necessary handshake signal.

The transputer has two timers; one of which 'ticks' every microsecond, the other ticks every 64 microseconds. The current value of the processor timer can be read, or a process can perform a *timer input* in which case it will become ready to execute when a specified time has been reached. Both these uses of the timer are treated as inputs similar to channel communication. If the timer is simply being read then the current timer value is provided immediately; if the process is waiting for a particular time, then it is descheduled until that time.

5.5 Shared resources

The IMS T9000 also provides efficient hardware support for controlling access to a shared resource. This could be a hardware resource (e.g. a printer) or a piece of software running on a particular processor in a network. Each process which wants to use the resource (a 'client') can make a request to the controlling process (the 'server'). This request is done in the form of a channel communication and can, therefore, be done across a network by using transputer links. If the resource is available then the requesting client is given access to it, otherwise it is put on a queue until the resource becomes free. If multiple clients request a resource then they are all automatically queued until it is available.

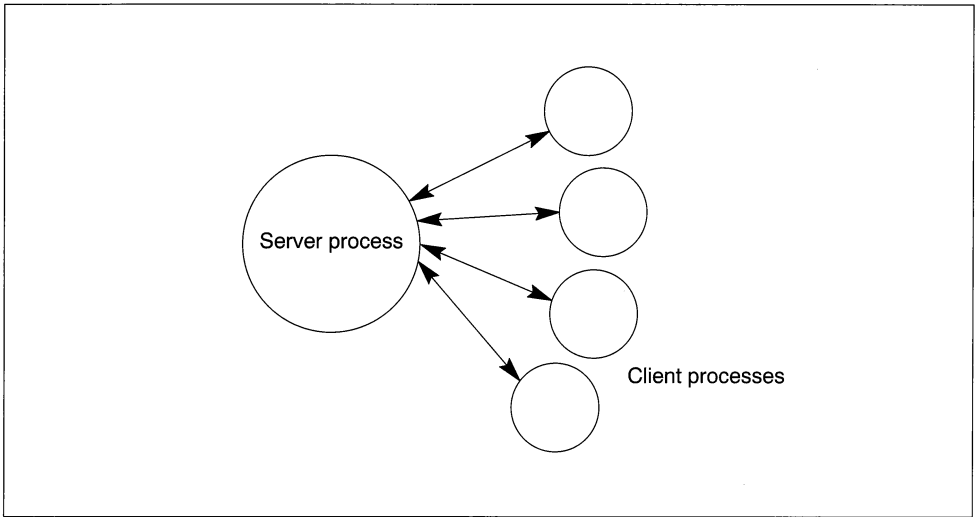


Figure 5.5 Client/server model of resources

The resource mechanism can provide pairs of channels between the server and the processes accessing it. This can be used, for example, to implement remote procedure calls across a transputer system.

6 Communication links

Transputer links provide a simple and regular way of interfacing to peripherals and host systems as well as communicating between transputers. On a single transputer, processes can communicate via channels; the provision of links allows processes on different transputers to communicate in the same way. The IMS C104 routing device enables this communication to take place across a network, even between transputers that are not directly connected.

The same communication model can be used to communicate with peripheral devices or a host system using a link adaptor which converts from the bit-serial protocol of the links to a parallel port.

6.1 Using links between transputers

Transputer links can be used to implement point to point communication between transputers. This allows transputer networks of arbitrary size and topology to be constructed. Point to point links have many advantages over bus based communications in a multiprocessor system:

- There is no contention for the communication mechanism, regardless of the number of processors in the system.
- There is no capacitive load penalty as more processors are added to the system.
- The communications bandwidth does not saturate as more communicating devices are added to the system. Rather, the larger the number of transputers, the greater the total communications bandwidth of the system.
- Because each transputer in a system uses its own local memory, overall memory bandwidth is proportional to the number of transputers in the system. This is in contrast to a large, global memory where the processors must share the available memory bandwidth.

For small systems, the four transputer links on the IMS T9000 can provide complete connection between up to five devices. By using additional message routing devices such as the IMS C104, networks of any size can be built with complete connection between all IMS T9000s. If a system does not need complete connection or the flexibility of routing that the IMS C104 provides, then networks can be built just from directly connected transputers.

6.2 Advantages of using links

The advantages of using links for communication are efficiency, simplicity and hardware independence.

6.2.1 Efficiency

There is a separate DMA controller for every input and every output channel which allows data to be transferred without processor involvement. To exploit this, the transputer deschedules a process which is waiting for a communication to complete, freeing the processor to execute another process. When the communication is complete, the process is requeued, providing automatic synchronization with the data transfer.

6.2.2 Simplicity

The communication links are very simple to use. The transputer has simple instructions for performing input and output and these are available to the programmer either as function/procedure calls in a high-level language or, in the case of OCCAM, as an integral part of the language. For example, in a C program, to transfer an array of 256 bytes from the array `data` to a channel `c`, the following call could be used:

```
ChanOut (c, data, 256);
```

In OCCAM, a similar operation could be written as:

```
c ! 256::data
```

This output operation requires four instructions: three to load the address of the channel, the address of the data and the number of bytes, followed by the output instruction itself. It is worthwhile comparing this with the complex code required to do the equivalent transfer on a traditional microprocessor. For example, it would require a DMA controller to be programmed and, in order to allow some degree of multitasking, it would be necessary to set up the interrupt hardware and write an interrupt handler to control the data transfer. All of this is done automatically by the input and output instructions on the transputer.

As a more concrete example, consider the case of a file server running on a host system talking to a program running on the transputer. This would provide the transputer program with all the host operating system facilities such as filing system, terminal i/o etc. At the transputer end, the communication is very simple: a single line of code, as outlined above. At the host end, a lot of complex code (probably written in assembler) is required to handle the data transfer, either programming a DMA controller or polling the status registers of the memory mapped port. In the case of a Unix system, it will also be necessary to write a device driver to interface to the hardware.

Of course, when the communication is between two transputers, then both ends of the communication are equally simple.

6.2.3 Hardware independence

As well as being fast and easy to use, channel communications provide a degree of hardware independence.

The same communication mechanism can be used to communicate between concurrent processes, with peripherals or a host system, and even to handle interrupts. This simplifies the development and testing of code as each process can be functionally tested before being used in the complete system. A good description of program development for transputers can be found in INMOS Technical note 5: *“Program design for concurrent systems”*, which is included in *The Transputer Applications Notebook – Systems and Performance*.

Furthermore, exactly the same code can be used to communicate between processes on the same transputer (using so called ‘soft channels’) and to communicate between transputers (using links, or ‘hard channels’). Not only is the source code the same, but the same transputer instructions are used – the transputer determines at run time whether it is using a hard or a soft channel. This saves the programmer from having to make decisions about the final hardware implementation while developing and testing code. The IMS T9000 takes this separation of software from hardware one step further than previous transputers.

6.3 IMS T9000 links

On previous transputers the programmer was limited to assigning two channels, one in each direction, to each link. To map a particular piece of software onto a given hardware configuration the programmer has to map processes to processors within the constraints of available connectivity. The problem is illustrated in figure 6.1 where 3 channels are required between two processors, but only a single link connection is available.

One possible solution, and one that is frequently suggested by transputer users, is the addition of more links. However this does not really solve the problems; there is still limited connectivity available. The number of extra links that can be added is limited by VLSI technology. This ‘solution’ does not address the more general communication problems in networks, such as communication between non-adjacent processors, or combining networks in a simple and regular way.

As an intermediate solution INMOS has incorporated software routing mechanisms into the C and OCCAM toolsets to provide ‘unlimited’ connectivity.

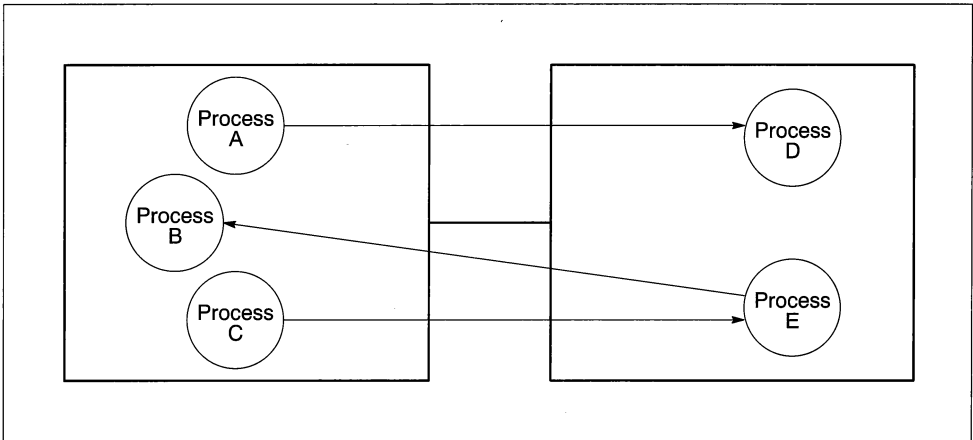


Figure 6.1 Multiple communication channels required between processors

6.3.1 Virtual channels

The solution chosen in the IMS T9000 was to add multiplexing hardware to allow any number of processes to use each link, so physical links can be shared transparently. These channels which share a link are known as 'virtual channels'; they have the same behavior as software channels.

The IMS T9000 has four data communication links, each with a DMA controller and the ability to synchronize with the scheduling of processes. The links and DMA engines are controlled by a separate communications processor, the virtual channel processor (VCP), which works concurrently with the CPU. This supports practically a large number of virtual channels on each link.

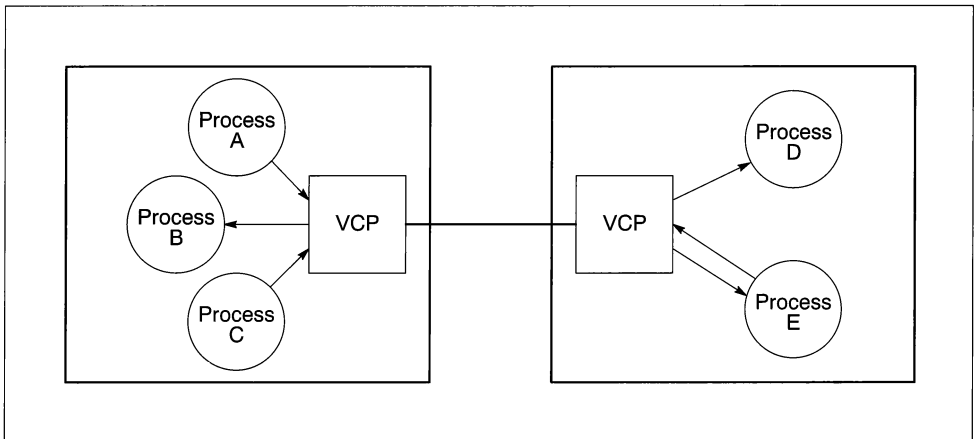


Figure 6.2 Shared links between IMS T9000s

Virtual links

Each message sent across a link is divided into packets. Every packet requires a header to identify its destination process. Packets from different messages are interleaved on the link. There are a number of advantages to this, as detailed below.

- It makes the transputer simpler to use as it separates the software configuration from the hardware. The programmer does not need to limit the number of channels between processors or explicitly allocate channels to links.

- Channels are, generally, not busy all the time therefore the VCP can make better use of hardware resource by keeping the links busy with messages from different channels.
- Messages from different channels can effectively be sent concurrently – the processor does not have to wait for a long message to complete before sending another.

Virtual channels are always created in pairs to form a 'virtual link'; this means there is no need for a return address in packets, the acknowledgements are simply sent back along the other channel of the virtual link.

Sending packets

The IMS T9000 sends the first packet of a message and then waits for an acknowledgement from the receiving processor before sending the next. The process which sent the message cannot proceed until the last packet of the message has been acknowledged. Messages and acknowledgements from other virtual links can be sent while waiting for an acknowledgement on a virtual link. This ensures that a single virtual link cannot monopolize a physical link or cause deadlock.

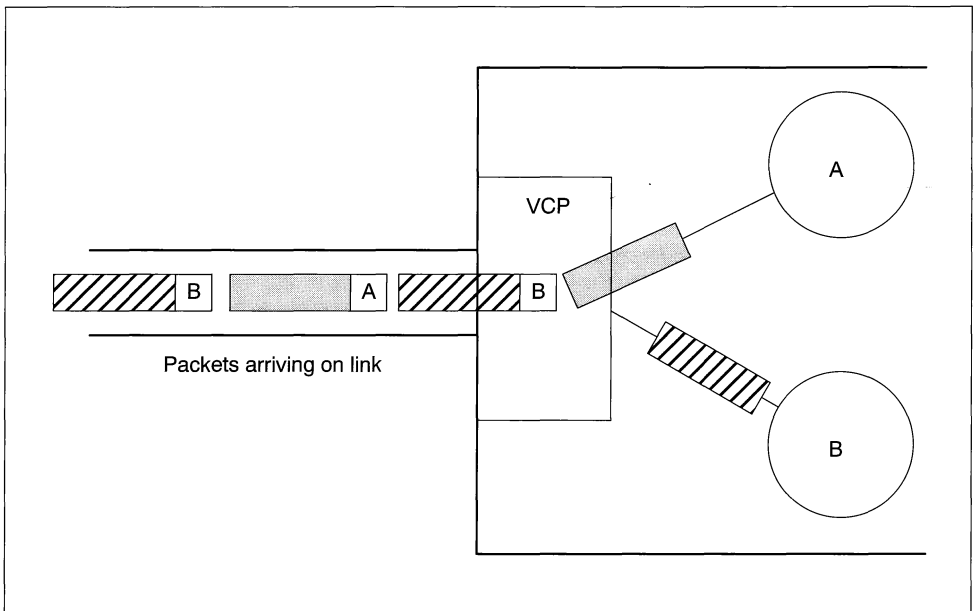


Figure 6.3 Multiple channels sharing a link

Receiving packets

The initial packet of a message is acknowledged if a process has requested a message on that virtual link. The acknowledgement can be sent as soon as the inputting process is identified, as long as the inputter is able to accept another packet. This means that the entire packet does not have to be received before the acknowledgement is sent. In this way the acknowledgement can be received by the transmitter before all of the data packet has been sent and the transmitter can send the next message packet immediately.

The IMS T9000 provides one packet buffer for each virtual link so that each input can be ready to accept an unsolicited packet. This means that other virtual channels sharing a physical link are not delayed if one virtual channel is not ready to input. This buffering of the first packet only takes place if the receiving process is not ready to input, otherwise the data is written directly to the inputting process's workspace. This buffer is not visible to the programmer; all communications are still synchronized at the message level.

The virtual channel processor

The VCP routes messages to and from processes on IMS T9000s. It shares each physical link between any number of processes. It also supports non-local communications by using the IMS C104 to route mes-

sages in a network of transputers. This can provide multiple virtual channels between any two transputers in a network. Requests to send messages are queued by the VCP so that the main CPU is not delayed waiting for packets to be sent.

Implementation

To achieve the speed required to match a faster processor, and to support the virtual channel protocol, a new, simple link standard has been implemented. The original transputer links are referred to as over-sampled (OS) links and use a pair of wires. The IMS T9000 links have four wires for each link (a data and strobe line in each direction) and are known as DS-Links. All signals are TTL compatible.

The links are asynchronous; the receiving device synchronizes to the incoming data. This simplifies clock distribution within a system, the exact phase or frequency of the clock on a pair of communicating IMS T9000s is not critical. It also means that devices with different processor speeds can communicate.

6.3.2 Levels of link protocol

As with many communications systems, the links can be described at a number of levels with a hierarchy of protocols. At the highest level a message consists of the data that the user sends down a channel from one process to another. Any type of data or message can be sent in this way. This communication is synchronized; it will not take place until both processes are ready and the two processes will not continue until the message transfer is complete.

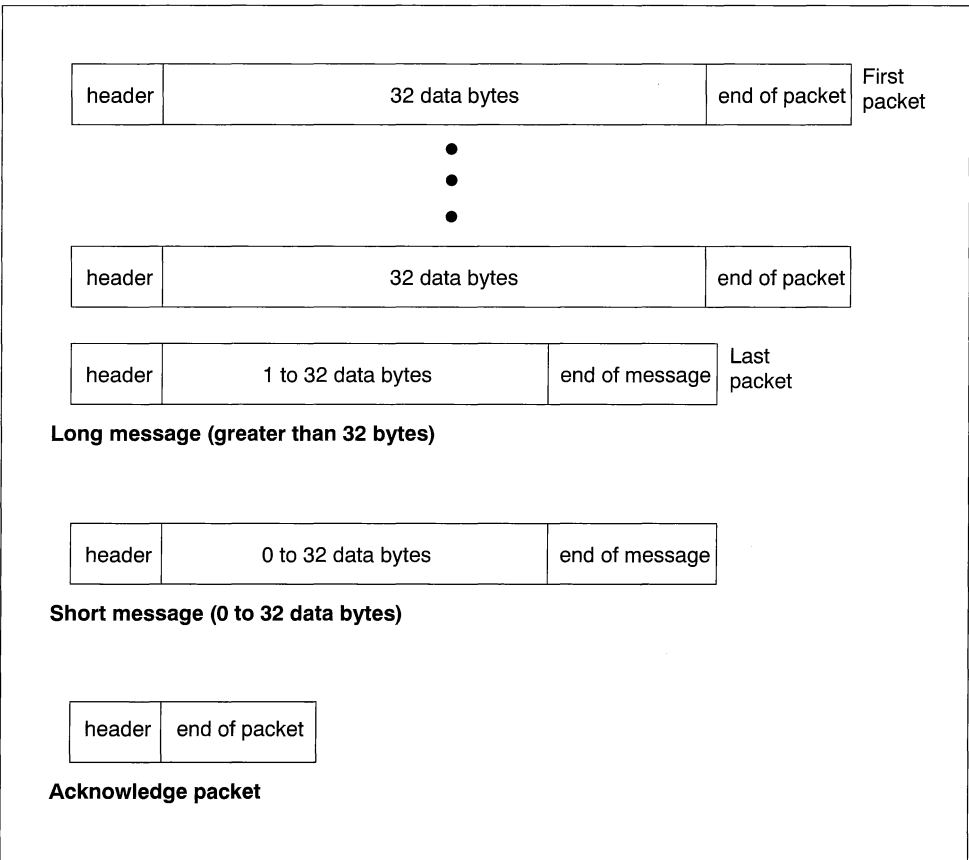


Figure 6.4 High level protocol

Packet level protocol

In order to transfer a message from one IMS T9000 to another, the virtual channel processor sends it as one or more packets. This allows packets from a number of different channels to be interleaved on the same link. Each packet is acknowledged by the receiving IMS T9000, to maintain synchronized communication and to limit the amount of buffering required.

Every packet has a header defining the destination address followed by the data bytes and, finally, an 'end of packet' or 'end of message' token. See figure 6.4. This simple protocol supports messages of any length; the receiving device knows when each packet and message ends without needing to keep track of the number of bytes received. It also maintains synchronization at the message level.

A packet can contain up to 32 data bytes. If a message is longer than 32 bytes then it is split up into a number of packets all, except the last, terminated by an 'end of packet' token. The last packet of the message, which may contain less than a full 32 bytes, is terminated by an 'end of message' token.

Shorter messages can be sent in a single packet, containing 0 to 32 bytes of data, terminated by the 'end of message' token. With this protocol zero length messages can be sent, allowing efficient synchronization between processors.

Packet acknowledgements are sent as zero length packets terminated with an 'end of packet' token. This type of packet can never occur as part of a message because a zero length data packet must always be the last, and only, packet of a message, and will therefore be terminated by an 'end of message' token.

Token level protocol

In order to support the packet level protocol described above, a lower level protocol is needed for encoding tokens which may contain a data byte or control information. Each token has a parity bit plus a control bit which is used to distinguish between data and control tokens. In addition to the parity and control bits, data tokens contain 8 bits of data and control tokens have two bits to indicate the token type (e.g. 'end of message').

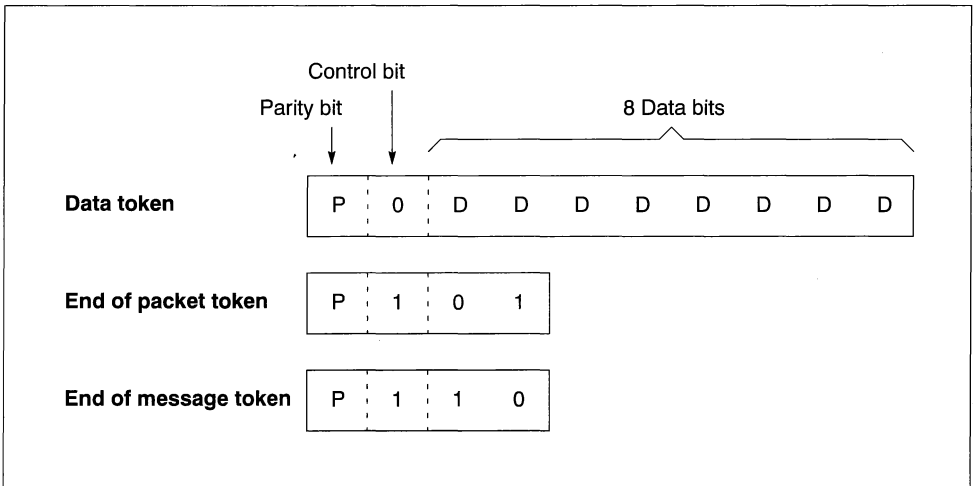


Figure 6.5 Low level protocol

Bit level protocol

At the lowest, hardware, level the signals on the data and strobe lines of a link encode a sequence of bit values. The protocol guarantees that only one of the two wires has an edge in each bit time. The levels on the data wire give the values of the transmitted bits. The strobe signal changes state whenever the data wire does not. These two signals encode a clock with the data bits, enabling asynchronous detection of the data at the receiving end.

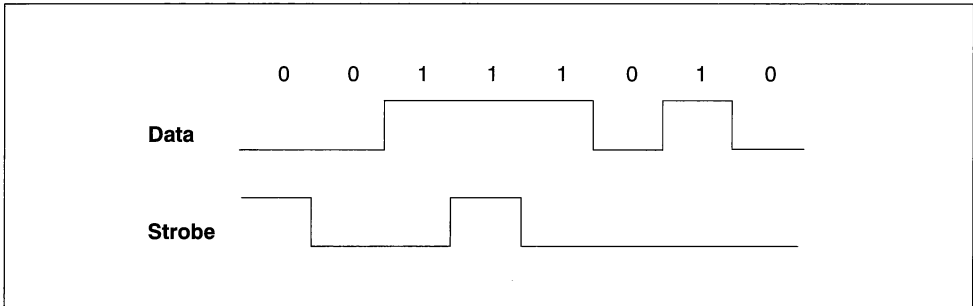


Figure 6.6 Hardware level

The first generation of transputers use a phase locked loop to synthesize a high frequency clock signal which is then used to sample the link data. This is adequate for the data rates involved, but would not easily support the bit rates of 100 Mbits/s and greater used by the IMS T9000.

7 Network communications

The use of INMOS links for directly connecting transputers has already been described. The new link protocol not only simplifies the use of links between processors but also provides hardware support for routing messages across a network.

7.1 Message routing

The VCP (virtual channel processor) on the sending IMS T9000 packetizes messages to be sent over a link and adds a header to each packet to identify the destination process. At the receiving end, the VCP uses the header to send the data in each packet to the intended process. These headers can also be used for routing packets through a communication system connecting a number of IMS T9000s together. This extends the idea of multiple channels on a single hardware link to multiple channels through a communications system; a communications channel can be established between any two processes even if they are running on transputers that are not directly connected. The header still just specifies the destination of the packet; the programmer does not need to know how to route that message to its destination.

Advantages for the programmer

The ability to have channels between any two processes in a network has a number of significant advantages for the programmer. It simplifies the description of multiprocessor systems by separating the hardware architecture from the software configuration. The programmer doesn't need to be concerned with the details of placing channels on links or routing messages through the network. This removes a lot of the problems with placing of processes on processors – the decision now can be made just on the basis of the resources (memory size, etc.) available on each processor without worrying about the available connectivity.

The programming model for networks of IMS T9000 transputers is unchanged from that for the first generation of transputers. There is, however, greater flexibility in configuring software. An important feature is that the hardware and software configurations, and therefore their descriptions, can be kept completely independent. The same hardware, and the same description of that hardware, can be used for many different programs.

Routers

The routing components in a network can be separated from the processing elements. Messages can be passed from one processor, through any number of routing devices, to the destination processor. This creates a temporary path through the routing system for that message so, from the programmers point of view, there still appears to be a single channel directly connecting a process on one transputer with a process on another.

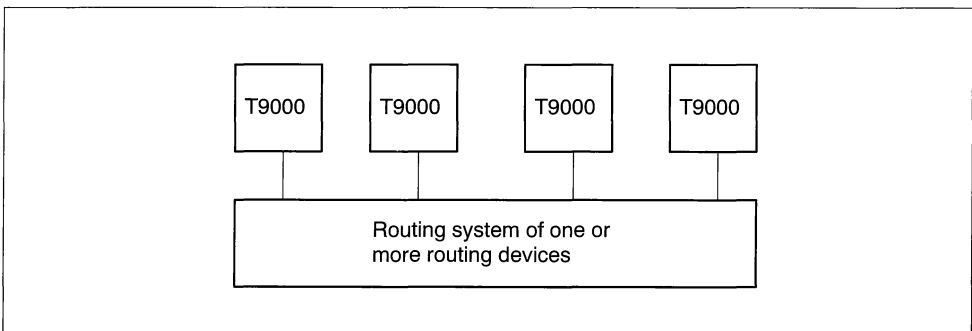


Figure 7.1 A routing system

As a packet arrives on a link, the destination address must be inspected before the outgoing link can be determined. The time before the output link can be determined is therefore proportional to the address

length. Further, the address itself must be transmitted through the network and consumes network bandwidth. It is therefore important that this address be as short as possible, both to minimize latency and maximize bandwidth.

The router needs to arbitrate between packets which arrive at the same time and have to be sent out of the same link. Ideally, it should start to output the packet as soon as possible; i.e. immediately after the output link is determined, provided that the link is not already in use by another packet. This keeps the latency through the network small, in contrast to a typical packet switching network which uses a 'store and forward' algorithm in which each packet is read into a buffer, the address information is decoded and then the packet is sent out. The delay that would be introduced by this may be unacceptable for many applications. Also the amount of buffering needed would make a VLSI implementation of a large routing switch impractical.

Separating routers and processors

There are a number of advantages to keeping the communications devices and processing elements separate in a system. Processors can be directly connected where appropriate, which avoids the silicon costs and extra routing delays in a small system that doesn't need to use the routers. Also, the design of the routing devices and processing elements can be optimized for their different roles. For example, the routing component can have a larger number of links than would be possible if the two devices were integrated, because the processor already needs a large number of pins for the memory interface and other functions. Having a routing device with many links means that large network with a small number of routers can be built, hence minimizing cost and latency and maximizing bandwidth. If messages had to flow through the processor, it would increase the pin count, power consumption and packaging costs. This approach also allows the construction of scaleable architectures where the communications throughput and processing power can be balanced.

Parallel networks

Because the new link architecture allows all the virtual channels of a transputer to use a single link, complete, system-wide connectivity can be provided by connecting just one link from each transputer to the routing network. This means that the IMS T9000, with its four links, can be connected to several different networks. This can be exploited in a number of ways. For example, two or more networks can be used in parallel to increase bandwidth, to provide a general purpose communications network and an independent monitoring/debugging network, or as a 'user' network running in parallel with a physically separate 'system' network.

7.2 The IMS C104

An important benefit of the IMS T9000's serial links is that it is easy to implement a full crossbar in VLSI, even with a large number of links. The use of a crossbar allows packets to be passing through all links at the same time, making the best possible use of the available bandwidth.

If the routing logic can be kept simple it can be provided for all the input links in the router. This avoids the need to share the hardware, which would cause extra delays when several packets arrive at the same time. It is also desirable to avoid the need for the large number of packet buffers commonly used in routing systems. The use of small buffers and simple routing hardware allows a single VLSI chip to provide efficient routing between a large number of links.

Wormhole routing

The IMS C104 (figure 7.2) is one of a family of compatible communications support devices for the IMS T9000. It includes a full 32 x 32 non-blocking crossbar switch, enabling messages to be routed from any of its links to any other link. In order to minimize latency, the switch uses 'wormhole routing' – the connection through the crossbar is set up as soon as the header has been read. The header and the rest of the packet can start being transmitted from the output link immediately. The path through the switch disappears after the 'end of packet/message' token has passed through. This is illustrated in figure 7.3. This method is simple to implement and provides very low latency as the entire packet doesn't have to be read in before the connection is made.

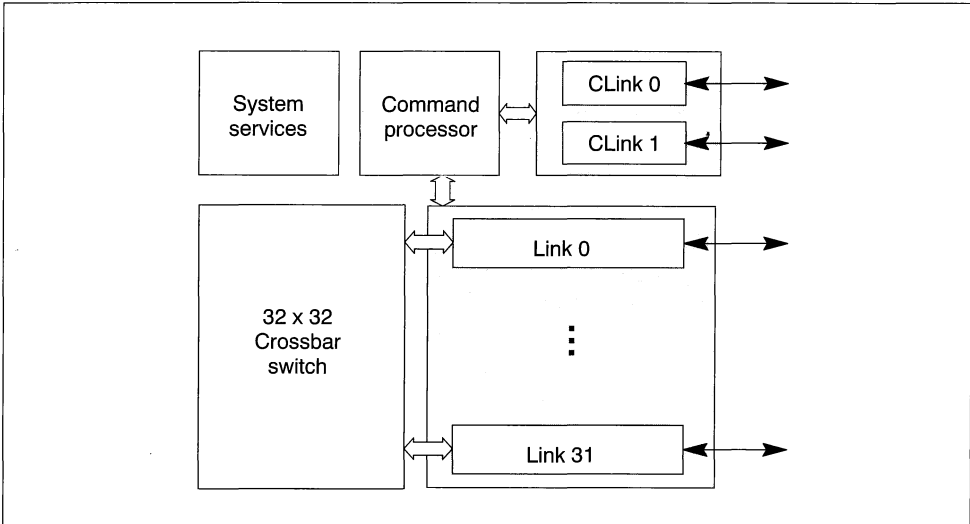


Figure 7.2 Block diagram of IMS C104

Minimizing routing delays

The ability to start outputting a packet while it is still being input can significantly reduce delay, especially in lightly loaded networks. The delay can be further minimized by keeping the headers short and by using fast, simple hardware to determine the link to be used for output. The IMS C104 uses a simple routing algorithm based on interval routing (described in section 7.3.1).

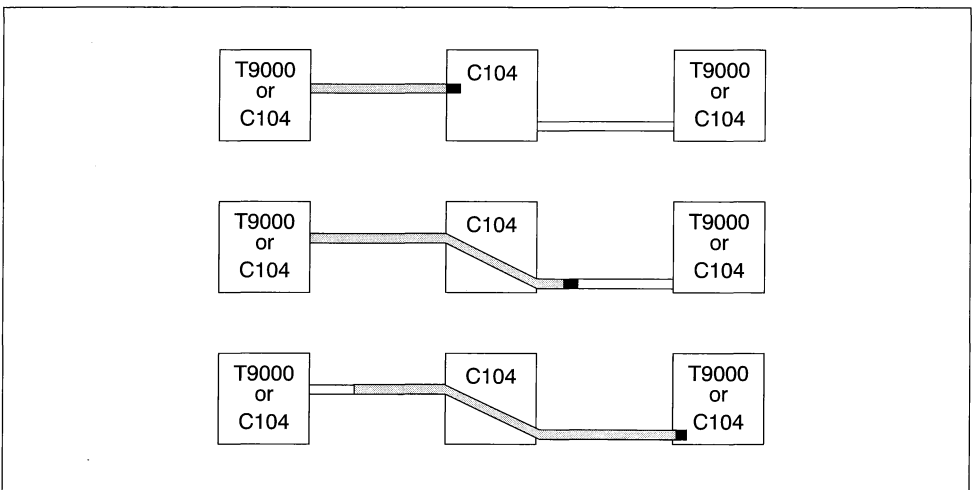


Figure 7.3 Packet passing through IMS C104

Because the route through each IMS C104 disappears as soon as the packet has passed through and the packets from all the channels that pass through a particular link are interleaved, a single virtual channel cannot 'hog' a route through a network. Messages will not be blocked waiting for another message to pass through the system, they will only have to wait for one packet.

Control links

Like the IMS T9000, the IMS C104 has two control links. One link receives control and programming information, the other enables all the devices in a system to be daisy-chained. The routing information for each link of each IMS C104 is programmed, via the control link, from the controlling processor.

7.2.1 Using IMS T9000s with IMS C104s

A single IMS C104 can be used to provide full connectivity between 32 IMS T9000s. It can also be used to connect other compatible communications devices, for example to provide an interface to first generation transputers via a protocol converter, or to peripheral devices via a link adaptor. IMS C104s can also be connected together to build larger switches connecting bigger networks of IMS T9000s.

The IMS C104s that the packets pass through do not need to have information about the complete route to the destination, only which link each packet should be sent out of at each point. Each of the IMS C104s in the network programmed with information that determines which output link should be used for each header value. In this way, each IMS C104 can route packets out of whichever link will send it towards its destination.

Header deletion

An approach that simplifies the construction of networks is to provide two levels of header on each packet. Note, headers are added by the VCP and values and lengths are set at initialization by the development system. The first header specifies the destination transputer (actually, the output link from the routing network), this header is removed as the packet leaves the routing system. This exposes the second header which tells the VCP in the destination transputer which process (actually, which virtual channel) this packet is for. To support this, the IMS C104 can route packets of any length. Any information after the initial header bytes used by the IMS C104 is just treated as part of the packet, even if it is going to be interpreted as a header elsewhere in the system. The IMS C104 can set any output link to do header deletion, i.e. to remove the routing header from the front of a packet after it has been used to make the routing decision. The first part of the remaining data is then treated as a header by the next device that receives the packet.

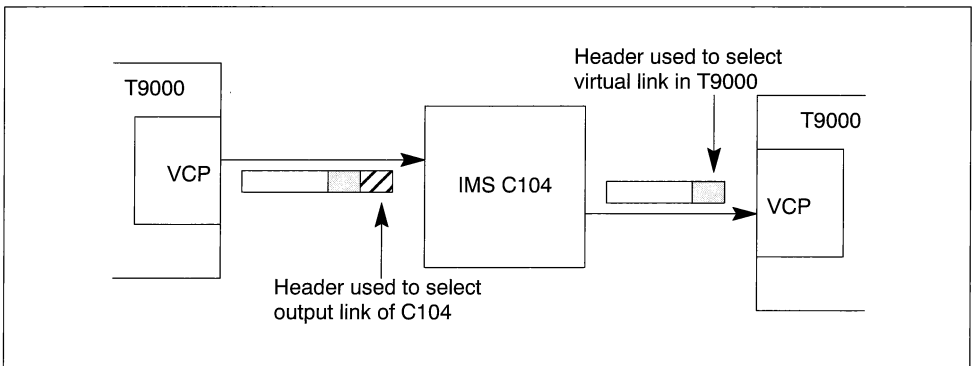


Figure 7.4 Header deletion

As can be seen from figure 7.5, by using separate headers to identify the destination processor and a channel within that processor, the labeling of links in a routing network is separated from the labeling of virtual channels within each processor. For instance, if the same 2 byte header were used to do all the routing in a network, then the virtual channels in all the transputers would have to be uniquely labelled with a value in the range 0 to 64K. However, by using two 1 byte headers, all the IMS T9000s can use virtual channel numbers in the range 0 to 255. The first byte of the header will be used by the routing system to ensure that the packets reach the appropriate IMS T9000 before the virtual channel number is decoded.

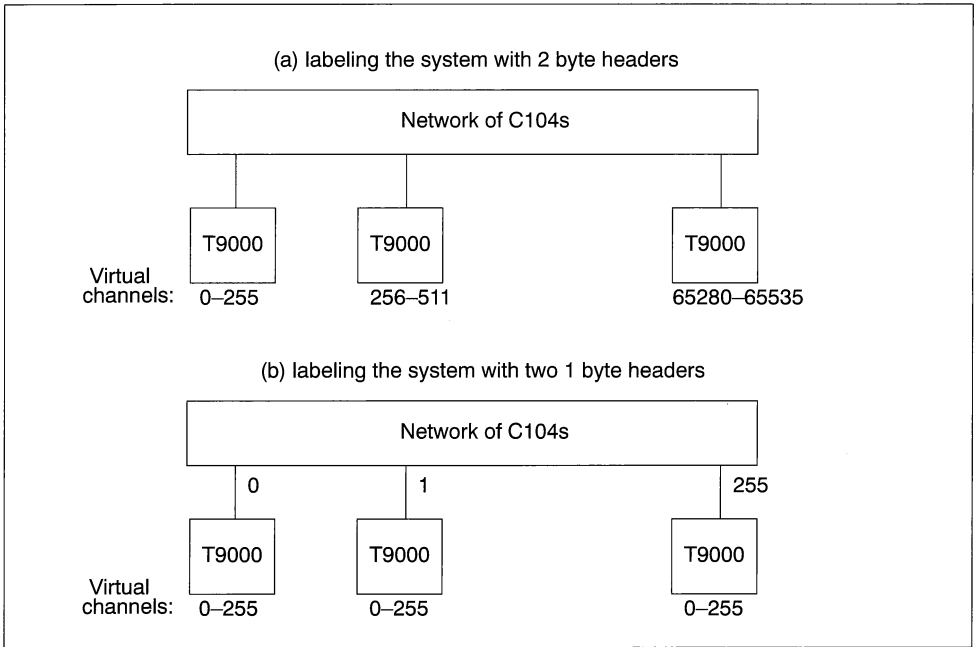


Figure 7.5 Using header deletion to label a network

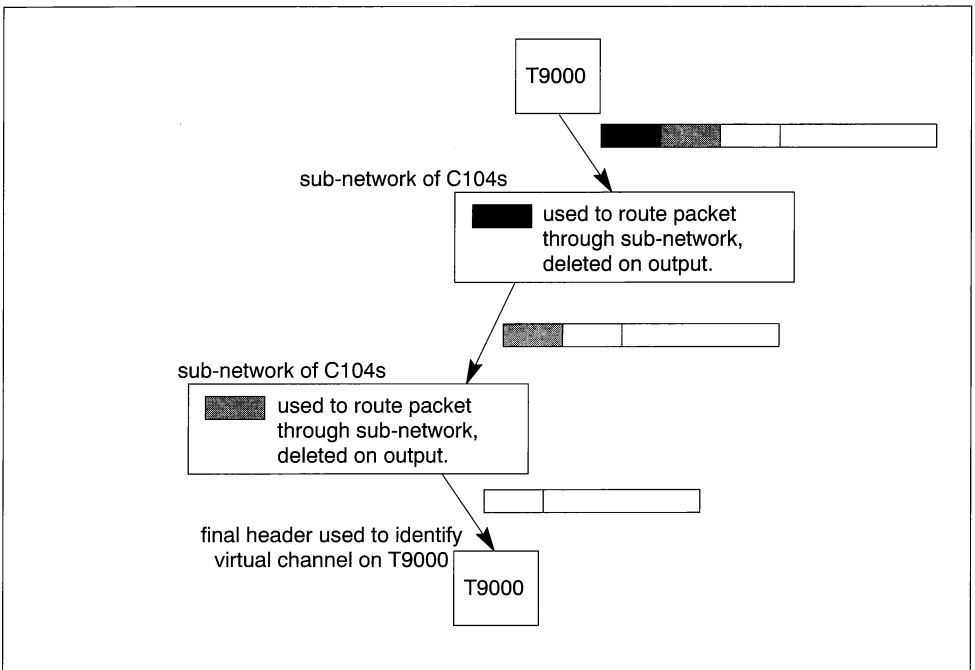


Figure 7.6 Using header deletion to route through sub-networks

The advantages of using header deletion in a network are:

- It separates the headers, and therefore the routing information, for virtual channels from those for the routing network.
- The labeling of the network can be done independently of the application software running on the network.
- There is no limit to the number of virtual channels that can be handled by a system.

Any number of headers can be added to the beginning of a packet so that header deletion can also be used to combine hierarchies of networks as shown in figure 7.6. An extra header is added to route the message through each network. The header at the front of each packet is deleted as it leaves each network to enter a sub-network.

Routing control channels

For very large networks, the usual method of connecting control links, in a chain, might introduce an undesirable delay. In this case, because of the common virtual link protocol, an IMS C104 can be used to route the control links to all the devices in a system more directly, as shown in figure 7.7.

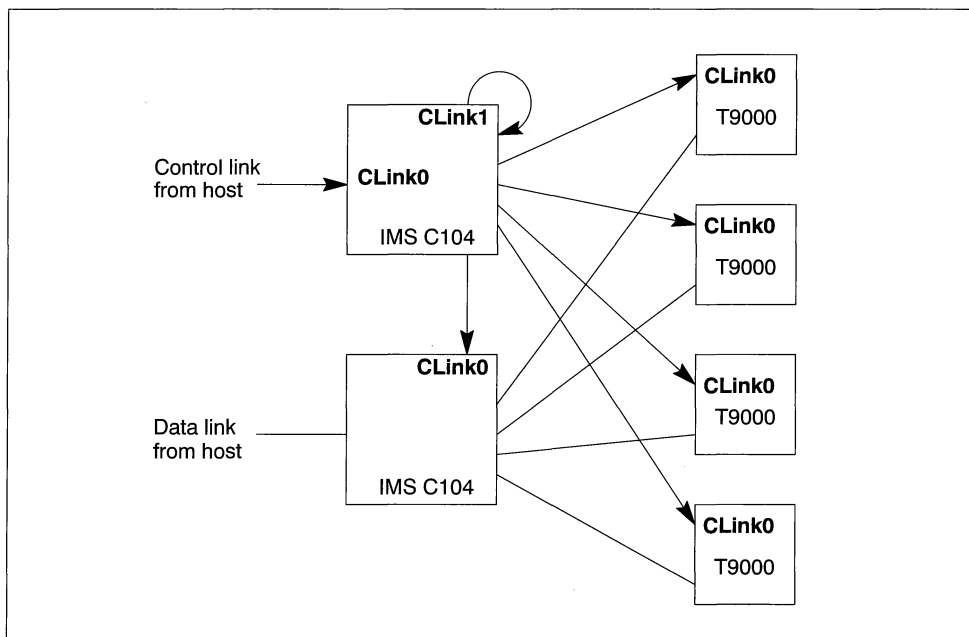


Figure 7.7 Routing control links through an IMS C104

7.3 Routing algorithms

In order to route a message through a network, an algorithm is required which is: complete (ensures that all messages arrive); deadlock free; optimal (packets take the shortest route); scalable (networks of any size can be built) and simple to implement.

7.3.1 Labeling networks

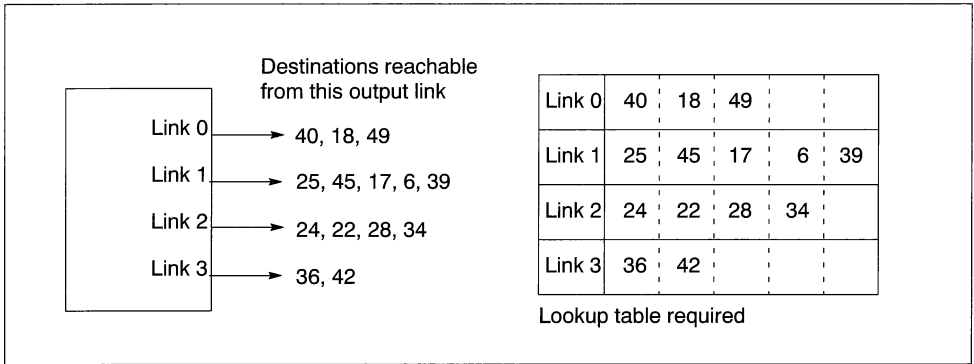


Figure 7.8 Labeling a network

For each routing component there will be a number of destinations which can be reached via each of its output links. Therefore, there needs to be a method of deciding which output link to use for each packet that arrives. The addresses that can be reached through any link will depend on the way the network is labelled. An obvious way of determining which destinations are accessible from each link, is to have a lookup table associated with all the outputs (see figure 7.8). In practice, this is difficult to implement. There must be an upper bound on the lookup table size and it may require a large number of comparisons between the header value and the contents of the table. This is inefficient in silicon area and also potentially slow.

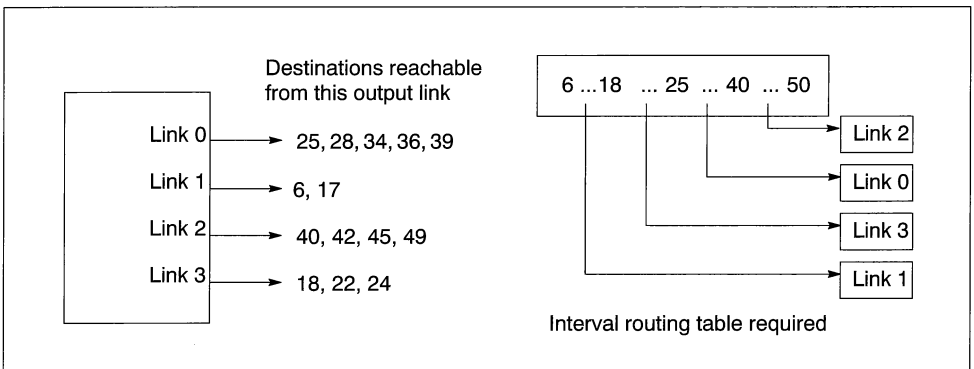


Figure 7.9 Interval labeling

However, a labeling scheme can be chosen for the network such that each output link has a *range* of node addresses that can be reached through it. If it is then ensured that the ranges for each link are non-overlapping, a very simple test is possible. The header just has to be tested to see into which range, or interval, it falls and, hence, which output link to use. For example, in figure 7.9, a header with address n would be tested against each of the four intervals shown below:

Interval	Output link
$6 \leq n < 18$	➤ 1
$18 \leq n < 25$	➤ 3
$25 \leq n < 40$	➤ 0
$40 \leq n < 50$	➤ 2

The advantages of interval labeling are that:

- It is 'complete' – any network can be labelled.
- It is simple to implement in hardware – it requires little silicon area which means it can be provided for a large number of links as well as keeping costs and power dissipation down.
- Because it is simple, it is also very fast, keeping routing delays to a minimum.

7.3.2 Avoiding deadlock

Deadlock can occur in a network unless the routing algorithm is designed to avoid it. Any program with communicating processes can also deadlock if not designed carefully. It is important here, to distinguish between deadlock as a property of the network and as a property of a program running on the network. A deadlock free network cannot *cause* a program to deadlock (but, of course, neither can it prevent a badly designed program from deadlocking). An essential property of a router in a deadlock free network is that, like a transputer or an IMS C104, it can communicate on all of its links concurrently.

As a simple example consider a network of four nodes (see figure 7.10) with one link in each direction between each node. If the routing algorithm sends all messages clockwise and all nodes start sending to the opposite corner at the same time, every link will become busy and the network will deadlock. It is possible to add buffers to the network, but this will only delay the point at which deadlock occurs. The amount of buffering needed to avoid deadlock is dependent on the network size and the application program running on the network.

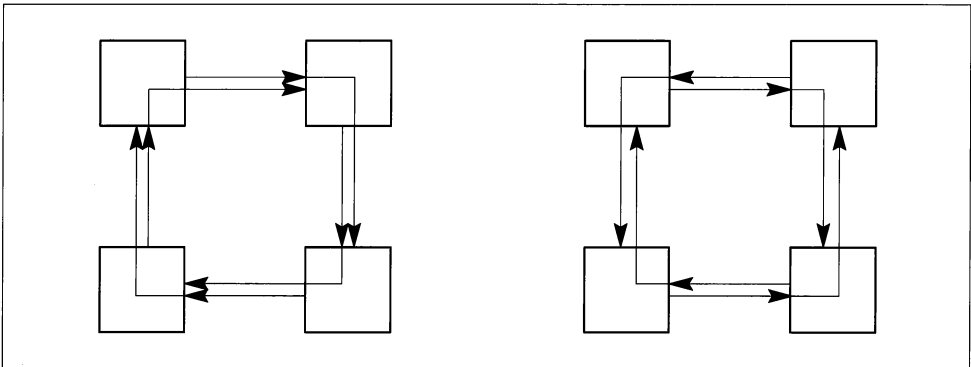


Figure 7.10 Deadlock in a network

In this example, deadlock can easily be avoided by modifying the routing algorithm to send messages in opposite directions from alternate nodes. In this case, each node will only need to send one message in each direction at any time. In this network, buffering can be added just to smooth the flow of data (i.e. to prevent a process having to wait to send a message when the network is busy) but it is not needed to prevent deadlock.

It is possible to use interval labeling to label any network in a deadlock free way. Many regular networks have optimal, deadlock free routing algorithms. Examples are trees, hypercubes and grids. These networks can then be combined, so that any network can be optimally labelled as if constructed from these sub-networks.

8 Other communications devices

To complete the IMS T9000 family, a full range of communications products are planned. These will provide the ability to interface transputers to a range of devices and technologies.

8.1 Mixing transputer types: the IMS C100

The first of these devices is the IMS C100. This allows an IMS T9000 to communicate with a first generation transputer. The two transputer families have different electrical characteristics and data protocol. The IMS C100 converts between the four wire DS-Links of the IMS T9000 and the two wire OS-Links of the earlier transputers.

The other conversion done by the IMS C100 is between the IMS T9000 control links and the **Reset, Error** and **Analyse** signals used to control the IMS T805 and similar device.

The IMS C100 provides an inter-networking solution for transputers, allowing transputer systems to be constructed using the optimum mix of devices. The IMS C100 has four modes of operation to enable:

- A single IMS T9000 to work in a network of first generation transputers.
- An existing transputer system to control a sub-system of IMS T9000s.
- An IMS T9000 network to interface to a network of first generation transputers.
- A first generation transputer to emulate an IMS T9000.

The IMS C100 converts both data and control protocols between the two transputer types and is intended to be used in conjunction with software running on the attached transputers.

Full details of the IMS C100 are given in Part 3: Communications Support Devices.

8.2 Interfacing to peripherals and host systems

To complete the family of communications devices, a range of interface devices are being designed. These will convert between the serial link format and a parallel interface, for example. The first of these devices will be the IMS C101 link adaptor. The IMS C101 link adaptor interfaces between IMS T9000 DS-Links and external systems such as buses, peripheral devices and other microprocessors.

9 Software and systems

INMOS provides a wide range of standard software and hardware products to support development for the transputer. These have been designed to enable users to evaluate transputers and to develop systems easily and within the shortest possible timescales.

Development tools include compilers for languages such as C, C++ and OCCam as well as the software needed to test, program and debug systems built from one or many transputers. All the special features of the transputer are available from high-level languages (either as part of the language or as library calls).

INMOS also supplies a range of modular hardware products. These exploit the ability to build very compact transputer systems (such as an IMS T805 with 4 Mbytes of memory on a board measuring approximately 2.5 cms by 9 cms) to provide a range of small, cost effective 'Transputer Modules' (TRAMs). These modules can be mounted on a variety of motherboards, which are available for a range of host systems. The motherboards provide an interface to the host development system and can be connected to build larger systems. The standard sizes and interfaces of the modules and motherboards have been adopted by a number of third party developers to extend the range of compatible systems products available to transputer users.

More details of the systems and software products currently available for the transputer family can be found in *The Transputer Development and iq Systems Databook*.

To support the IMS T9000, a new industry standard based on 100 Mbits/s links has been developed called HTRAMs. Refer to *The T9000 Development Tools Preliminary Datasheets (document number 72-TRN-249-00)* for further details.

9.1 Development software

INMOS has a range of development software, running on different hosts, for the transputer family. These tools are aimed mainly at developing code for embedded systems, i.e. not necessarily running under the control of an operating system. It is expected that the end products will either be connected to a host system or will be completely self-contained units.

Software can be developed in standard high-level languages using cross-compilers running on a range of host machines. Programs for single transputers can be developed using conventional programming tools, such as compilers and linkers. All the languages include extensive support, in the form of run-time libraries, for concurrency and communication. It is possible to write a program consisting of many concurrent processes entirely in C (or any other language available for the transputer).

Programs are written for transputer networks, of any size, using a standard set of tools and libraries. The language toolset contains the appropriate language compiler plus support tools for linking and configuring for arbitrary network architectures (from a single IMS T9000 to large T9000/C104/C10X). The toolset includes configuration tools which are used for describing the hardware, mapping processes to transputers and setting up the communications channels. It is possible to boot and load a network with code from the host development system, or from ROM connected to one of the transputers in the network; configuration is a separate phase preceding booting a network and can be via control links from a host or system ROM or configured by a local ROM. Programs can communicate with a server on the host system to get access to host facilities such as file I/O. T9000 network initialization and checking tools are also provided. IN-QUEST builds upon the standard language toolsets to provide advanced debugging and performance maximization utilities.

The programming tools for the IMS T9000 are source compatible with those for the first generation transputer family. Separate compatible toolsets will be maintained, each toolset targeting the hardware configuration and development systems hardware product most efficiently.

9.1.1 Configuration tools

In discussing IMS T9000 transputer systems, the word 'configuration' is used in two senses. The first is when an IMS T9000 transputer, or an IMS C104, is initialized – at this time a number of internal 'configura-

tion' registers have to be written to program the PMI, the VCP and other subsystems. The process of preparing a program for loading onto a transputer network is also referred to as 'configuration' (and the software tools used are known as 'configurers'). In this description of the development process, the word 'configuration' is reserved for the latter meaning of software configuration; the setting up of the hardware will be called 'initialization'.

The configuration tools are used to build programs consisting of a number of processes or sub-programs running in parallel on one or more transputers. Input files are used to describe the hardware, the software and a mapping of the software onto the hardware. From these, the configuration tools produce the files which are used to initialize and load the transputer network.

Hardware description

The hardware is described using a Network Description Language (NDL). For each transputer in the system, this specifies the processor type, the amount and types of memory and peripheral devices. It also describes the routing network used, if any, and how the data and control links of all the devices in the system are connected.

The configuration tools use this description to program the PMI and VCP registers of the IMS T9000 and to label the links of any IMS C104s used. The information in this file is also used to create the bootable version of a program to run on the network.

If certain simple rules are followed in the construction and labelling of networks, then the tools can check the descriptions for errors and deadlock freedom. The NDL description can also be checked against the actual hardware.

Software description

The NDL file for a particular system will normally be provided by the hardware vendor or designer. The programmers using the system only need to include a reference to the NDL file in the software configuration file. The NDL description exports the names of the processors and routes in the network for use in the software and mapping description.

The software description has to specify the linked process image for each process in the system and the procedure interface (parameters and their types). Optionally, other language dependent attributes can be defined. For example, the size of stack and heap areas for a C program can be specified. The software description must also specify the way that any communication channels are used between processes.

Mapping software to hardware

A mapping of software (processes) onto hardware (transputers) must also be given. The mapping can be as simple as a series of statements of the form: 'place *process* on *processor*' for each process in the program. Any number of processes can be placed on each processor, allowing a program to be initially tested on a single processor before the multi-processor version is tried. The configuration tools automatically work out the mapping of channels onto virtual links. If necessary, for example to access the host system or a particular piece of hardware, the programmer can explicitly map channels onto links or routes through the network.

Configuration languages

To provide a degree of flexibility for the user, there are two 'dialects' of configuration language: a C-like one and an OCCAM-style one. These perform identical functions but each has a different syntax, loosely based on these languages. These configuration languages are used for describing the structure of the software and how it is mapped onto the hardware.

Types of networks

The INMOS development tools support development of programs for:

Networks consisting of IMS T9000 transputers only ('non-routed' networks).

Networks consisting of IMS T9000 transputers and IMS C104 routers ('routed' networks).

Networks consisting of mixed T2/T4/T8 transputers and IMS T9000 transputers.

The tools do not directly support arbitrary, mixed networks of IMS T9000 transputers and first generation devices. However, it is possible to connect the two types of networks, via an IMS C100, although the code for the two sub-networks has to be developed separately. The two networks can then be loaded from the host, via separate routes, or by getting one network to load the other.

In the case of non-routed networks (of any transputer type) the configuration tools automatically add routing software to the program to provide any communications required between processors which are not directly connected. This software routing technology compliments IMS T9000 and IMS C104 hardware virtual routing.

9.1.2 Initializing and loading a network

Transputer systems can initialize and boot either from ROM or from control link. The initialization stage defines device configuration, this configuration may be via a control link or from ROM. Subsequent to this booting of the device occurs, again this can be via a control link or from ROM. This flexible approach allows for a number of options for optimizing the use of ROM's in embedded systems.

Levels of initialization

The initialization and loading of code for the IMS T9000 are done in a number of stages. The various levels of initialization can be done either by code running on an IMS T9000 booted from ROM, or from the host system via the control link. In a network, different processors may be initialized to different levels from ROM with the later stages being done via the control link.

Booting a system from link

The 'boot from link' option is normally used during program development or whenever a system needs to be able to run different programs at different times.

Boot from link is used to free each processor from having a program fixed in its memory (either in development environment or reducing ROM's in a system down a single system boot ROM).

In order to load a network from a host system, connections to a single control link and a single data link are required. This data link normally goes directly to an IMS T9000, the rest of the network being loaded via this processor. The development tools generate data files which are used to do all the initialization and loading of code onto the network.

Booting a system from ROM

The development tools can produce a number of different types of ROM. These range in function from performing the (partial) initialization of a single IMS T9000, to booting an entire system.

When booting a system completely from ROM, it is possible to have a single ROM on one processor. This root processor boots from the ROM and then initializes and loads the rest of the network via links; all other transputers in the network being set to boot from link.

9.1.3 Host servers

A server is a program that runs on the host machine to give software, running on an attached transputer system, access to various host facilities such as i/o and disk storage. The server typically loads the executable code onto the transputer network via a link interface. It then waits for requests and data to be sent by the transputer program. These requests generally come from the run-time library, when the program makes calls to standard input and output functions (e.g. `printf()` in C).

The server allows the development tools running on the host to control the target transputer system in order to reset the system, do any initialization needed and then load a bootable program file. Software running on the host can also use the server to access the transputer system for testing and debugging.

The nature of the connection from the host to the transputer system depends on the type of the host system, but generally provides access to transputer links either directly, via a link adaptor on the host bus,

or through some other standard communications system such as shared memory or Ethernet. In many cases the server software includes a device driver, which handles the low-level details of the hardware interface, plus a set of functions to access the data transfer of the device driver.

9.1.4 Debugging

INMOS provides an interactive symbolic debugger for debugging programs running on networks of transputers. This supports source level debugging of programs which consist of a number of parallel processes running on any number of processors. The user can set breakpoints, inspect the state of processes (including expression evaluation, modification of variables, backtracing procedure calls, etc) as well as examining the low-level state of each transputer in the system.

INMOS debugging technology offers a leading environment for the development of embedded applications through to massively parallel systems. The debugger offers a wide range of facilities for sequential code execution, monitoring and control. It also provides advanced features specifically designed for working with parallel processes and communications. This debugger is included in the INQUEST product.

The INQUEST package contains a windowing debugger and performance optimization tools and has the following features:

- Windowing operation (OSF/Motif and Microsoft Windows 3)
- Single stepping of transputer instructions and source statements (with threads)
- Conditional/Programmable breakpoints and watchpoints
- Source or assembly level views
- Programmable command language
- Program interrupt and restart facilities
- Percentage of time executing each procedure
- Percentage of time at high priority for transputer
- Idle time analysis for transputer
- Percentage of time in each process
- Utilization of network over time - displays an interactive chart
- Remote download for booting over network
- Access to host O/S
- Integration of user supplied host services
- Distributed access to services

9.2 IMS T9000 systems products

Assembled IMS T9000 transputers are available on small printed circuit boards, known as High performance TRAnsputer Modules (or HTRAMs). These supercomponents integrate an IMS T9000 processor and up to 16 Mbytes memory on a board approximately the size of a credit card. HTRAMs are plugged into specially designed motherboards, and communication is achieved via the transputer's four 100 Mb/s links. This packaging option offers a number of benefits:

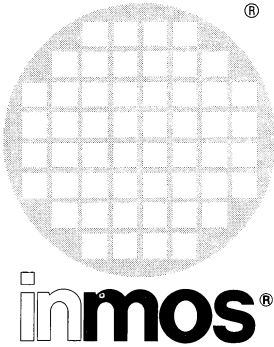
- Simplified design-in – An IMS T9000 surface mount component has 208 pins on a 0.5 mm pitch. An HTRAM supercomponent has just 40 pins on a 2.0 mm pitch.
- Reduced time to market – HTRAMs are fully-tested assemblies that are ready-to-use straight from the box.
- Extra performance as required – system performance can be upgraded at any time by simply adding more HTRAMs.
- Reduced cost of ownership – using HTRAMs minimizes high value inventory, and eases the transition to faster devices.
- Inter-operability – public HTRAM specification is supported by vendors worldwide.
- Cost effective – volume manufacturing operation makes HTRAMs affordable in production systems.

Refer to the *The T9000 Transputer Development Brochure* and *The T9000 Development Tools Preliminary Datasheets* for further details.



Part 2

IMS T9000 transputer preliminary data



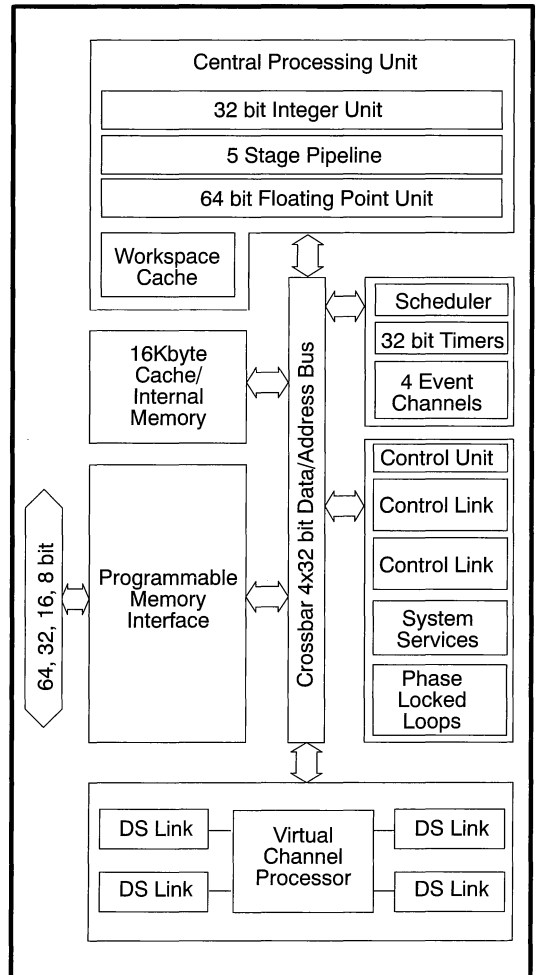
IMS T9000 transputer

Preliminary Data

The information in this datasheet is subject to change

FEATURES

- Pipelined superscalar micro-architecture
- Workspace cache
- Programmable memory interface
- 4 Gbyte physical address space
- 16 Kbyte instruction and data cache
- 200 MIPS peak
- >70 MIPS sustained
- 25 MFLOPs peak
- >15 MFLOPs sustained
- Sub-microsecond interrupt response
- Per process error handling
- Enhanced support for pre-emptive schedulers
- Memory protection and address translation
- 64 K virtual communication channels
- Support for message routing
- 80 Mbytes/s total bi-directional link bandwidth
- Separate control system
- Single 5 MHz clock input
- 40, 50 MHz speed options
- 208 pin CLCC package
- Single 5 V \pm 5% power supply



1 IMS T9000 introduction

This part contains hardware information for the IMS T9000 transputer.

The IMS T9000 transputer is a 32-bit CMOS microprocessor designed to be used in applications which require high performance combined with high integration and simplicity of use. Software support for the IMS T9000 transputer includes ANSI C compilers and OCCAM toolsets developed and supported by INMOS.

Figure 1.1 shows the major operational units of the IMS T9000 transputer.

The IMS T9000 has a pipelined superscalar architecture, which allows multiple instructions to be executed every processor cycle. Compilers can generate code without considering any details of the pipeline as the hardware organizes the incoming instruction stream into optimum groups of instructions. Other features which contribute to performance are a 16 Kbyte instruction and data cache, a 64-bit floating point unit, and a high bandwidth programmable memory interface. A separate workspace cache stores 32 locations relative to the workspace pointer to provide zero latency access to local variables. The IMS T9000 has four communication links for fast inter-processor communications.

The floating point unit (FPU) incorporates hardware to perform divide and square root. The FPU provides single and double length arithmetic to floating point standard IEEE 754-1985. The IMS T9000 running at a processor speed of 50 MHz is able to perform floating point operations at a rate of 15 Mflops sustained and 25 Mflops peak.

The 16 Kbyte cache provides a peak bandwidth of 200 Mwords/sec. It can also be programmed to function as 16 Kbyte of on-chip memory, or as 8 Kbyte of on-chip memory and 8 Kbyte of cache. This allows small applications to run with no external memory, and guarantees deterministic code behavior for applications where this is critical.

The highly integrated programmable memory interface has a 4 Gbyte physical address space, and provides a peak bandwidth of 50 Mwords/sec. Four independent banks of external memory are supported, and this allows the implementation of mixed memory systems, with support for DRAM, SRAM, EPROM and VRAM. It has a 64-bit data bus, and each bank of memory can be configured to be 8, 16, 32 or 64 bits wide. The full performance of the IMS T9000 can be exploited using low-cost DRAM, and up to 8 Mbytes of DRAM can be connected with no external components.

Transputers provide hardware support for scheduling processes, and this can be used directly by applications written, for example, in C or OCCAM. It can also be used to simplify the software implementation of real-time kernels and operating systems. The process model of the IMS T9000 transputer provides per process error handling and debugging support, and allows programs to be run in a protected logical address space. To improve the efficiency of real-time kernels access to the state of the processor has been simplified, and full control over interrupts and timeslicing has been provided.

Communication between processes takes place over channels, and is implemented in hardware. The same machine instructions are used for communication between processes on the same processor as for communication between processes on different IMS T9000 processors. On the IMS T9000, communication between processes on different processors takes place over *virtual* channels. Virtual channels are multiplexed onto each physical link by the virtual channel processor. Communication between IMS T9000 transputers that are not directly connected is achieved by using a separate dynamic routing switch, the IMS C104.

With virtual channels it is not necessary for the programmer to allocate channels to physical links, and the allocation of processes to processors is simplified. The programming of powerful multiprocessor systems is therefore flexible and elegant.

The IMS T9000 has four high-bandwidth serial communication links which support virtual channels and dynamic message switching and provide a high data bandwidth with high data integrity. Each physical link consists of four wires, two in each direction, one carrying data and one carrying a strobe. The links are therefore referred to as data-strobe (DS-Links). The four DS-Links support a total bidirectional data bandwidth of 80 Mbytes/sec.

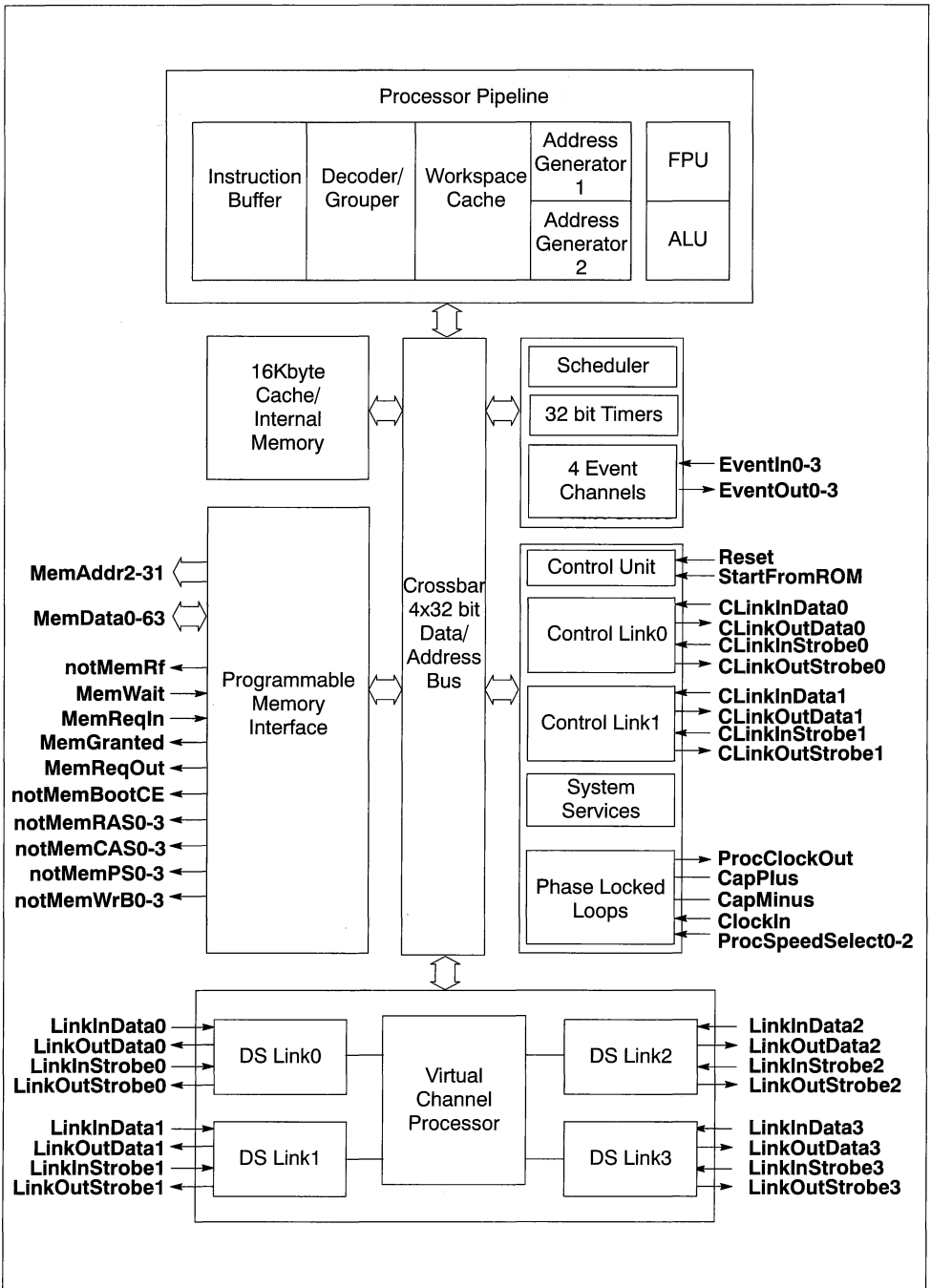


Figure 1.1 IMS T9000 block diagram

Two separate control links are provided to enable networks of IMS T9000 processors to be controlled and monitored for errors, even during the presence of faults in the normal data communications network. The control links of IMS T9000s and IMS C104s can be daisy chained, and/or connected into a tree by connection to a IMS C104. Whatever the physical connectivity the controlling network forms a logical tree, and a control processor is connected at its root. For small systems (such as a single IMS T9000 transputer) there is no need to use the control links as all necessary functionality can be controlled from software.

The hardware scheduler enables the creation and execution of any number of high and low priority processes. It handles timeslicing and message passing, often eliminating the need for a software kernel. Context switch time is sub-microsecond. Two 32-bit clocks tick at $1\mu\text{s}$ and $64\mu\text{s}$ intervals. The scheduler provides each process with a timer, simplifying real-time programming.

Four Event I/O pins provide asynchronous handshake interfaces between external events and internal processes and can be used as interrupts or as control for external peripherals. Response time is sub-microsecond.

Two on-chip phase locked loops generate all the internal high frequency clocks from a single clock input, simplifying system design and avoiding problems of distributing high speed clocks externally. The nominal input clock frequency used by all transputer family components is 5 MHz, regardless of device type, transputer word length or processor cycle time.

2 Pin designations

The following tables outline the function of each of the pins. Pinout details are given in chapter 16.

Signal names are prefixed by **not** if they are active low, otherwise they are active high.

Supplies

Pin	In/Out	Function
VDD		Power supply
GND		Ground

Table 2.1 IMS T9000 supplies

Phase locked loops

Pin	In/Out	Function
CapPlus, CapMinus		External capacitor for internal clock power supply
ClockIn	in	Input clock
ProcSpeedSelect0-2	in	Processor speed selectors
ProcClockOut	out	Processor clock

Table 2.2 IMS T9000 phase locked loops

Programmable memory interface

Pin	In/Out	Function
MemAddr2-31	out	Address bus
MemData0-63	in/out	Data bus
notMemRAS0-3	out	RAS strobes – one per bank
notMemCAS0-3	out	CAS strobes – one per bank
notMemPS0-3	out	Programmable strobes – one per bank
notMemWrB0-3 †	out	Byte-addressing write strobes
MemWait	in	Memory cycle extender
MemReqIn	in	Direct memory access request
MemGranted	out	Direct memory access granted
MemReqOut	out	Processor requires memory bus
notMemBootCE	out	Bootstrap ROM chip enable
notMemRf	out	Dynamic memory refresh indicator
notMemStrobe	out	Reference strobe for external bus cycles.

† these pins have different functions depending on the external port sizes

Table 2.3 IMS T9000 programmable memory interface

Control system

Pin	In/Out	Function
StartFromROM	in	Boot from external ROM or from link
Reset	in	System reset
CLinkInData0-1	in	Control link input data channels
CLinkInStrobe0-1	in	Control link input strobes
CLinkOutData0-1	out	Control link output data channels
CLinkOutStrobe0-1	out	Control link output strobes

Table 2.4 IMS T9000 control system

Communication links

Pin	In/Out	Function
LinkInData0-3	in	Link input data channels
LinkInStrobe0-3	in	Link input strobes
LinkOutData0-3	out	Link output data channels
LinkOutStrobe0-3	out	Link output strobes

Table 2.5 IMS T9000 communication links

Events

Pin	In/Out	Function
EventIn0-3	in	Event inputs
EventOut0-3	out	Event outputs

Table 2.6 IMS T9000 event

Miscellaneous

Pin	In/Out	Function
HoldToGND		Must be connected to GND
HoldToVDD		Must be connected to VDD
DoNotWire		Must not be wired

Table 2.7 IMS T9000 miscellaneous pins

3 Central processing unit

The IMS T9000 central processing unit (CPU) contains a 32 bit arithmetic and logic unit (ALU) and a 64 bit floating point unit (FPU). This chapter describes the architecture and operation of the CPU. Specific details on the FPU are described separately in the Floating point unit chapter 4.

3.1 Registers

The design of the IMS T9000 transputer processor exploits the availability of a fast on-chip cache and a workspace cache by having only a small number of registers; five registers are used in the execution of a sequential integer process. The five registers are:

- The workspace pointer (**Wptr**) which points to an area of store where local variables are kept.
- The instruction pointer (**lptrReg**) which points to the next instruction to be executed.
- The **Areg**, **Breg** and **Creg** registers which form an evaluation stack.

Areg, **Breg** and **Creg** are sources and destinations for most arithmetic and logical operations. Loading a value into the stack pushes **Breg** into **Creg**, and **Areg** into **Breg**, before loading **Areg**. Storing a value from **Areg**, pops **Breg** into **Areg** and **Creg** into **Breg**, the value left in **Creg** is undefined.

Expressions are evaluated on the evaluation stack, and instructions refer to the stack implicitly. For example, the *add* instruction adds the top two values in the stack and places the result on the top of the stack. The use of a stack removes the need for instructions to re-specify the location of their operands. No hardware mechanism is provided to detect that more than three values have been loaded onto the stack. It is easy for the compiler to ensure that this never happens.

A separate floating point evaluation stack is provided, consisting of **FPAreg**, **FPBreg**, and **FPCreg**. The floating point evaluation stack behaves in a similar way to the integer evaluation stack.

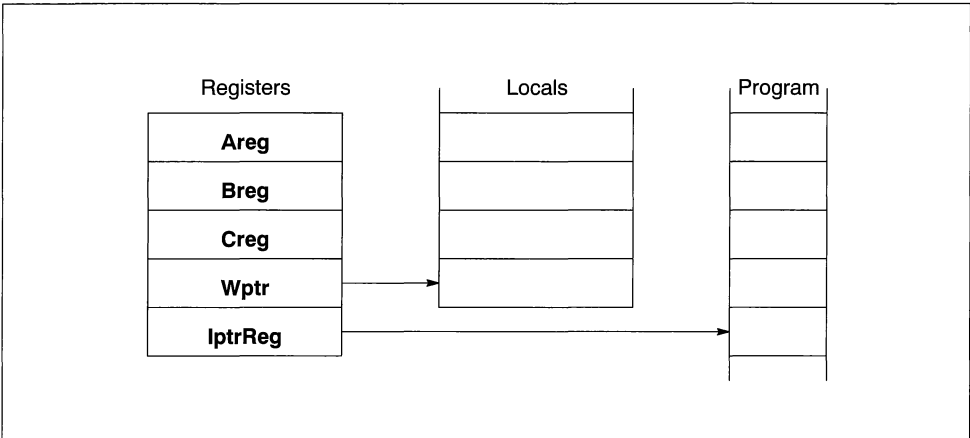


Figure 3.1 Registers used in sequential integer processes

Any location in memory can be accessed relative to the workspace pointer, enabling the workspace to be of any size. The first 32 words relative to the workspace pointer may be cached by the workspace cache.

3.2 Workspace cache

The workspace cache can hold a copy of the first 32 words of procedure stack and workspace. It is triple ported, allowing two reads and a write in every cycle. The workspace cache allows local data to be accessed without going outside the CPU, effectively giving zero cycle access and reducing the load on the main cache and external memory. It also means that the pipeline can do four data reads (as well as an instruction fetch) in each cycle: 2 from the local cache and 2 from the main cache.

Local variables can be accessed quickly and therefore can be read in the first stage of the pipeline and can then be used for non-local address calculations in the next stage. The workspace cache is write-through; whenever data is written into the local cache it is also written to the main cache. Thus there is no overhead for flushing the cache on interrupt or context switch.

Cache operation

The cache is organized as a 32 word circular buffer and is addressed using the bottom five bits of the workspace pointer. As the workspace pointer moves up and down, it rolls around the cache. When the workspace pointer is moved down, on a procedure call for instance, the locations that 'roll into' the cache are marked as invalid and become valid as they are read or written. The first time a variable is read, it is copied from the main cache (and fetched from main memory if it is not in the main cache). Lines are marked as invalid when they 'roll out' of the cache as the workspace pointer is moved up (e.g. on a return from a procedure call). On a context switch or interrupt, the entire contents of the cache are marked as invalid.

This is illustrated in figure 3.2, where the state of the workspace cache during a procedure call and return sequence is shown. Before the call, the locations in the workspace cache above the workspace pointer which have been read or written by the program contain valid data. After the call, the workspace pointer moves down – initially the locations which are above the workspace pointer are invalid; as they are accessed by the program they are filled with data and marked as valid. When the procedure returns, the locations which it used will be marked as invalid. As long as the workspace of the called procedure is less than 32 words, some of the workspace of the calling procedure will still be valid after the return. Nested procedure calls, or calls of procedures with a large workspace requirement will cause the workspace pointer to wrap around so that some of the data at the top of the program workspace is no longer in the cache.

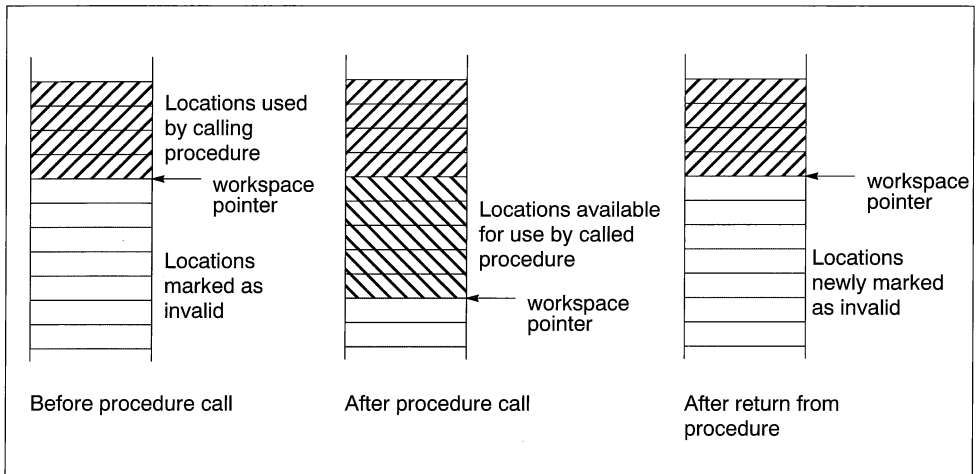


Figure 3.2 Effects of call and return on workspace cache

As the cache is a circular buffer, moving the workspace pointer by 32 or more will cause the pointer into the cache to wrap right round, marking every line as invalid.

3.3 Processes and concurrency

A process starts, performs a number of actions, and then either stops without completing or terminates complete. Typically, a process is a sequence of instructions. A transputer can run several processes in parallel (concurrently). Processes may be assigned either high or low priority, and there may be any number of each.

The processor has a microcoded scheduler which enables any number of concurrent processes to be executed together, sharing the processor time. This removes the need for a software kernel, although kernels can still be written.

At any time, a process may be

- Active*
 - Being executed.
 - Interrupted by a higher priority process.
 - On a list waiting to be executed.

- Inactive*
 - Ready to input.
 - Ready to output.
 - Waiting until a specified time.
 - Waiting on a semaphore.

The scheduler operates in such a way that inactive processes do not consume any processor time. Each active high priority process executes until it becomes inactive. The scheduler allocates a portion of the processor's time to each active low priority process in turn (see section 3.4). Active processes waiting to be executed are held in two linked lists of process workspaces, one of high priority processes and one of low priority processes. Each list is implemented using two registers, one of which points to the first process in the list, the other to the last. In the linked process list shown in figure 3.3, process *S* is executing and *P*, *Q* and *R* are active, awaiting execution. Only the low priority process queue registers are shown; the high priority process ones behave in a similar manner.

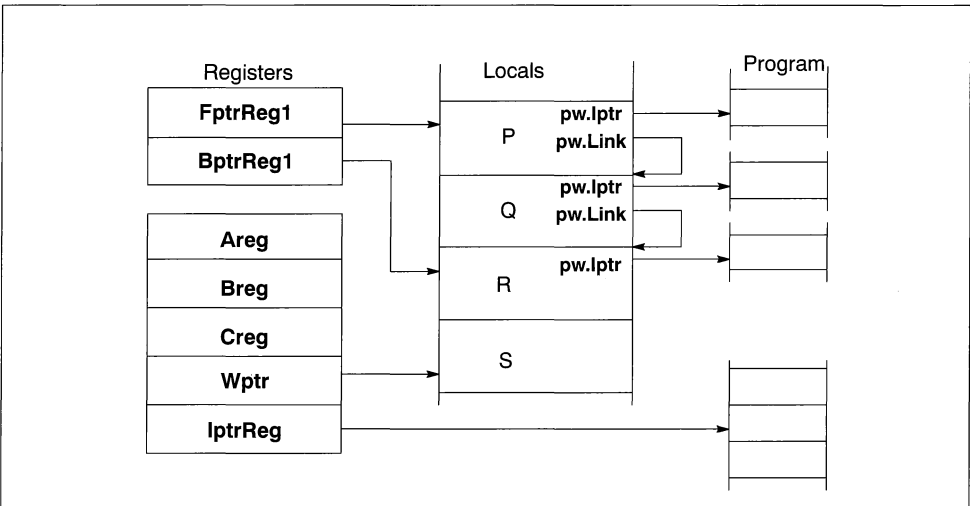


Figure 3.3 Linked process list

Function	High priority	Low priority
Pointer to front of active process list	Fptr0Reg	Fptr1Reg
Pointer to back of active process list	Bptr0Reg	Bptr1Reg

Table 3.1 Priority queue control registers

Each process runs until it has completed its action or is descheduled. In order for several processes to operate in parallel, a low priority process is only permitted to execute for a maximum of two timeslice periods. After this, the machine deschedules the current process at the next timeslicing point, adds it to the end of the low priority scheduling list and instead executes the next active process. The timeslice period is 256 μ s.

There are only certain instructions at which a process may be descheduled. These are known as descheduling points. A process may only be timesliced at certain descheduling points. These are known as timeslicing points. As a result, an expression evaluation can be guaranteed to execute without the process being timesliced part way through.

Whenever a process is unable to proceed, its instruction pointer is saved in the process workspace and the next process taken from the list.

The processor provides a number of special instructions to support the process model, including *startp* (start process) and *endp* (end process). When a main process executes a parallel construct, *startp* is used to create the necessary additional concurrent processes. A *startp* instruction creates a new process by adding a new workspace to the end of the scheduling list, enabling the new concurrent process to be executed together with the ones already being executed. When a process is made active it is always added to the *end* of the list, and thus cannot pre-empt processes already on the same list.

The correct termination of a parallel construct is assured by use of the *endp* instruction. This uses a data structure that includes a counter of the parallel construct components which have still to terminate. The counter is initialized to the number of components before the processes are started. Each component ends with an *endp* instruction which decrements and tests the counter. For all but the last component, the counter is non zero and the component is descheduled. For the last component, the counter is zero and the main process continues.

3.4 Priority

The IMS T9000 transputer directly supports two levels of priority: low priority and high priority. Low priority processes are executed whenever there are no active high priority processes.

High priority processes are expected to execute for a short time. If one or more high priority processes is able to proceed, then the first on the queue is selected and executes until it has to wait for a communication, a timer input, or until it completes processing.

If no process at high priority is able to proceed, but one or more processes at low priority is able to proceed, then one is selected. Low priority processes are periodically timesliced to provide an even distribution of processor time between computationally intensive tasks.

If there are n low priority processes, then the maximum latency from the time at which a low priority process becomes active to the time when it starts processing is $2n - 2$ timeslice periods. It is then able to execute for between one and two timeslice periods, less any time taken by high priority processes. This assumes that no process monopolizes the transputer's time; i.e. it has a distribution of timeslicing points.

When the processor is executing a low priority process and a high priority process becomes ready to execute, an interrupt occurs. The state of the low priority process is saved into 'shadow' registers and the high priority process is executed. When no further high priority processes are able to run, the state of the interrupted low priority process is re-loaded from the shadow registers and the interrupted low priority process is re-started.

Instructions are provided on the IMS T9000 transputer to allow a high priority process to store the shadow registers to memory and to load them from memory. Instructions are also provided to allow a process to exchange an alternative process queue for either priority process queue. These instructions enable a pre-emptive scheduler to be constructed.

3.5 L-processes: local error handling and debugging

When running a process, the IMS T9000 transputer provides a set of localized, per-process, trap-handling and debugging mechanisms. Processes are therefore referred to as L-processes.

An L-process allows error conditions to be handled. The layout of the workspace for an L-process is shown in table 3.2.

Word offset	Slot name	Purpose
0	pw.Temp	Slot used by some instructions for storing temporary values
-1	pw.lptr	Instruction pointer of a descheduled process
-2	pw.Link	Address of the workspace of the next process in scheduling list
-2	pw.Count	Message length in variable length communication
-3	pw.TrapHandler	Pointer to trap-handler data structure (THDS)
-4	pw.Pointer	Saved pointer to communication data area
-4	pw.State	Saved alternative state
-4	pw.Length	Length of message received in variable length communication
-5	pw.TLink	Address of the workspace of the next process on the timer list
-6	pw.Time	Time that a process on a timer list is waiting for

Table 3.2 Word offsets from **Wptr** and names for data slots in an L-process workspace

Each L-process has a trap-handler, a set of error flags, and a set of trap enable bits. Whenever an error is detected either the appropriate error flag is set or, depending on the state of the trap enable bits, the trap-handler is invoked. Trap-handlers may be shared between processes of the same priority.

When an L-process is executing, the pointer to the trap-handler is held in the trap-handler register (**ThReg**). When an L-process is inactive the pointer to the trap-handler is held in the **pw.TrapHandler** slot of the process workspace.

If the value of the trap-handler pointer in the workspace of an L-process is *NotProcess.p* this indicates a null trap-handler (refer to Appendix A for values and definitions of special constants). Any process which executes with the null trap-handler ignores any floating point errors and any invalid non-word aligned accesses. Any other error results in the processor halting and an error message being output on the control link **CLink0**.

A trap-handler consists of a trap-handler data structure (THDS) and a process to be executed when an error occurs. The THDS must be word-aligned but can be placed anywhere in memory. It contains: a block of store into which state can be saved when an error occurs; a pointer to the trap-handler code; and a queue of processes waiting to use the trap-handler. The layout of a THDS is shown in table 3.3.

Word offset	Slot name	Purpose
11	th.sCreg	L-process C register
10	th.sBreg	L-process B register
9	th.sAreg	L-process A register
8	th.slptr	L-process instruction pointer
7	th.sWptr	L-process descriptor
6	th.eWu	Upper bound of L-process watchpoint region
5	th.eWl	Lower bound of L-process watchpoint region
4	th.Eptr	Pointer to instruction causing trap
3	th.Bptr	Back of trap sharing process queue
2	th.Fptr	Front of trap sharing process queue
1	th.lptr	Trap-handler instruction pointer
0	th.Cntl	Control word

Table 3.3 Layout of the THDS (trap-handler data structure)

The control word is used to control the operation of the trap-handler, and to store the flags and trap enable bits whenever the trap-handler is not being used by an executing process. When an L-process starts to execute its trap-handler pointer is loaded from its workspace into **ThReg**, the trap control bits in its control word are loaded into the status register and the process is allowed to execute. The status register (**StatusReg**) contains status and control information for the current process. The 32-bit word held in the status register comprises 'status bits' ('flags') and 'control bits'. Status bits describe current state, such as the mode of operation (protected/unprotected) and any errors which may have occurred. Control bits specify future behavior which may occur, such as trapping and timeslicing.

The IMS T9000 can detect various error conditions, including integer errors, word alignment errors, illegal instructions and IEEE floating point exceptions. If a trap enable bit for a particular error is set, a trap will be taken. Although for some errors if this bit is not set, a flag will be set instead. Further details can be found in the *T9000 Transputer Instruction Set Manual*.

The IMS T9000 prevents more than one process using the same trap-handler. It achieves this by setting a bit (**sb.ThInUse**) in the control word when the trap-handler is entered and clearing it when the trap-handler is exited. Before an L-process is executed the processor checks the **sb.ThInUse** bit in the control word of its trap-handler. If the trap-handler is found to be in use then the L-process is queued onto the trap-handler's process queue. All of the processes on the trap-handler's process queue are dequeued and inserted onto the front of the appropriate priority scheduling list when the trap-handler is exited.

When a trap-handler is invoked the integer state of the processor is written to the THDS. The floating point state is restored to the state which was present *before* the operation was performed. (This makes it simple for the trap-handler to compute the correct value to be delivered to an IEEE exception handler.). The floating point and block move state of the processor is not saved by the hardware, and it is left to the trap-handler to save this state as necessary using the *fpstall* (floating point store all) and *stmove2dinit* (store 2D move) instructions. The error flags and trap enable bits are written from **StatusReg** to the control word of the trap-handler, and the **sb.ThInUse** bit is set. The trap-handler code is then started with the trap reason and error type being returned in **Areg** and **Breg**.

Once a trap-handler has completed, it loads any floating point and block move state using the *fpldall* and *move2dinit* instructions respectively, and executes the *tret* (trap return) instruction. **Areg** contains a conditional argument to the *tret* instruction. If the value in **Areg** is *not* zero then the errant process will be descheduled. If it *is* zero then the error flags and trap enable bits will be re-loaded into **StatusReg** from the trap-handler control word, the integer state of the processor will be re-loaded from the THDS, and the trapped process will be allowed to continue.

The current error flags and trap enable bits may be examined by using the *ldflags* (load error flags) instruction, which copies the error flags and trap enable bits from **StatusReg** into **Areg**. The error flags and trap enable bits in **StatusReg** may be set to the value in **Areg** using the *stflags* (store error flags) instruction.

When running an L-process the IMS T9000 transputer provides support for breakpointing, for single-stepping of instructions, and for a watchpointed region. The IMS T9000 transputer interprets the *j0* instruction as a breakpoint which causes the trap-handler to be called, with the state of the process being saved as described above.

3.6 Timers

The transputer has two 32-bit timer clocks which 'tick' periodically. The timers provide accurate process timing, allowing processes to deschedule themselves until a specific time.

One timer is accessible only to high priority processes and is incremented every microsecond, cycling completely in approximately 4295 seconds. The other is accessible only to low priority processes and is incremented every 64 microseconds, giving exactly 15625 ticks in one second. It has a full period of approximately 76 hours.

Register	Function
ClockReg0	Current value of high priority (level 0) process clock
ClockReg1	Current value of low priority (level 1) process clock
TnextReg0	Indicates time of earliest event on high priority (level 0) timer queue
TnextReg1	Indicates time of earliest event on low priority (level 1) timer queue
TptrReg0	High priority timer queue
TptrReg1	Low priority timer queue

Table 3.4 Timer registers

The current value of the processor clock can be read by executing a *ldtimer* (load timer) instruction. A process can arrange to perform a *tin* (timer input), in which case it will become ready to execute after a specified time has been reached. The *tin* instruction requires a time to be specified. If this time is in the 'past' then the instruction has no effect. If the time is in the 'future' then the process is descheduled. When the specified time is reached the process is scheduled again.

A *Stop* command (see section 8.4, page 115 for a complete description of control commands sent via the control links), causes the timers to stop scheduling processes, however it does not stop the clock registers from ticking, thus the processes remain on the timer queue.

Figure 3.4 shows two processes waiting on the timer queue, one waiting for time 21, the other for time 31.

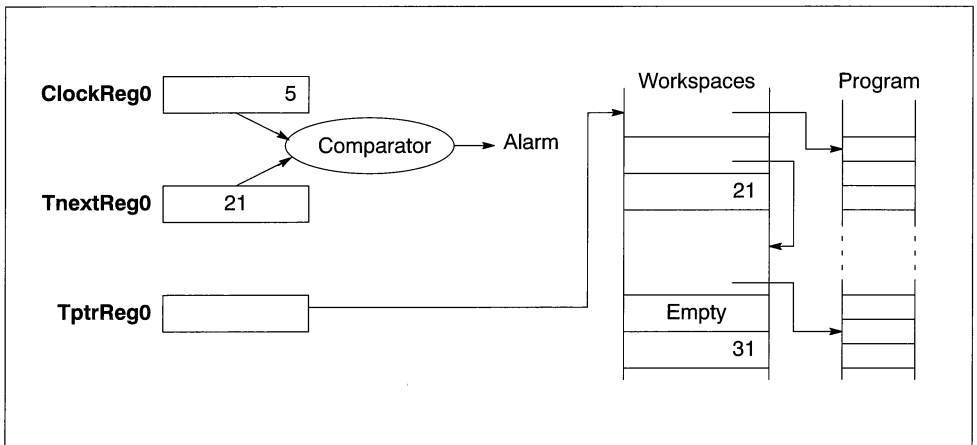


Figure 3.4 Timer registers

3.7 Block move

The block move instruction (*move*) on the transputer moves any number of bytes from any byte boundary in memory, to any other byte boundary, using the smallest possible number of word reads, and word or part-word writes. *move* is interruptible, to guarantee low latency.

3.8 Semaphores

The IMS T9000 transputer provides an efficient implementation of an n-valued semaphore for processes on the same processor. *signal* and *wait* instructions are provided which operate on a data structure which may be located at any address in memory. A semaphore is implemented by a three word data structure. The word slots in the data structure are shown in figure 3.5. For further details refer to the *T9000 Transputer Instruction Set Manual*.

Slot name	Purpose
s.Count	Number of extra processes that the semaphore will allow to continue running after a <i>wait</i> on the semaphore.
s.Front	Pointer to workspace of first process waiting on the semaphore queue.
s.Back	Pointer to workspace of last process waiting on the semaphore queue.

Table 3.5 Contents of a semaphore data structure

3.9 Pipeline

The CPU of the IMS T9000 performs its computation on a processor pipeline. This pipeline consists of 5 stages and, where possible, multiple instructions are combined into a group and passed down the pipeline together. This allows more than one instruction to be executed on each processor cycle. The details of the pipeline are transparent to the programmer. The processor appears to be the simple transputer architecture described above and code can be generated for the IMS T9000 transputer without considering the details of the pipeline. However, optimizing compilers can produce more efficient code if these details are taken into consideration.

Instructions are executed in a 5 stage pipeline: the first stage can fetch two local variables; the second can perform two address calculations, for accessing non-local or subscripted variables; the third stage

can load two non-local variables; the fourth can perform an ALU or FPU operation; and the final stage can do a conditional jump or write.

The IMS T9000 incorporates hardware to assemble groups of instructions from the instruction stream, enabling multiple instructions to be *issued* per cycle (as well as multiple instructions being executed in each cycle). These groups are chosen to make the best use of the available hardware and one group can be sent through the pipeline every cycle. Instructions are put into groups in the order that they arrive at the CPU; dependencies within the group are handled automatically by the pipeline.

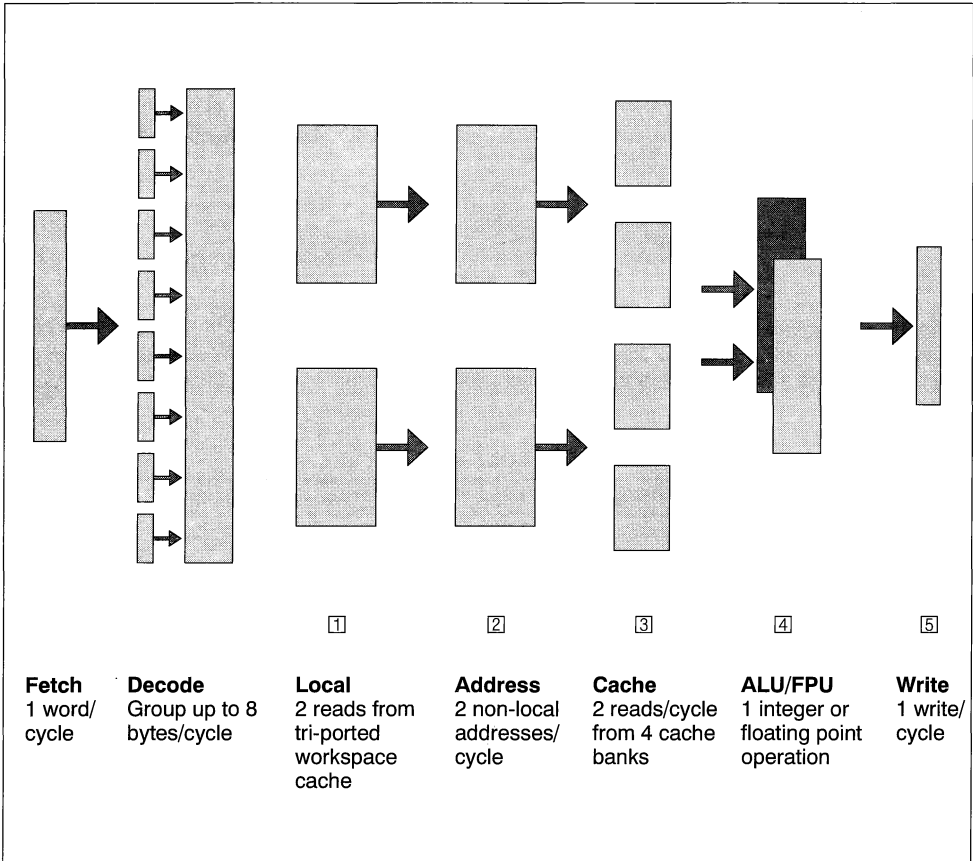


Figure 3.5 Pipeline operation

The grouper can be thought of as a hardware optimizer; it recognizes commonly occurring code sequences that the processor can execute effectively. The design of the grouping mechanism and the pipeline is based on analysis of the code typically generated by high-level language compilers.

3.9.1 Grouping of instructions

The grouping of instructions takes advantage of the high degree of concurrency and multiple buses in the processor. For example, both caches are multi-ported and can each support two reads by the CPU simultaneously. This allows two *ldl* (load local) instructions to go into one group, and the group could also contain two sets of instructions to calculate addresses and fetch non-local variables. These could all be combined with an arithmetic operation such as *add*.

As an example of how the grouper works, consider the assignment and expression evaluation shown below. The code produced is shown along with the number of the pipeline stages in which it is executed.

```
a[i+1] = b[j+15] + c[k+7];
```

ldl	j	1	load local variable j
ldl	b	1	load base address of array b
wsub		2	calculate address of b[j]
ldnl	15	2 3	load value of element b[j+15]
ldl	k	1	
ldl	c	1	
wsub		2	
ldnl	7	2 3	load value of c[k+7]
add		4	add two values on top of stack
ldl	i	1	
ldl	a	1	
wsub		2	
stnl	1	2 5	store into a[i+1]

This code sequence will be executed as three groups as shown below (i.e. in 3 cycles achieving 217 MIPS at a processor speed of 50 MHz). The exact contents of each group will depend on the code which precedes and follows this. The first group might contain other instructions from earlier in the instruction stream.

first group	ldl, ldl, wsub, ldn1
second group	ldl, ldl, wsub, ldn1, add
third group	ldl, ldl, wsub, stnl

Since the processor can fetch one word, containing four bytes of instructions and data, in each cycle it is possible to achieve a continuous execution rate of four instructions per cycle (200 MIPS). However, if any of the instructions require more than one cycle to execute, then the instruction fetch mechanism can continue to fetch instructions so that larger groups can be built up. Up to 8 instructions can be put into one group and there may be five groups in the pipeline at any time.

3.10 CPU configuration registers

The CPU (in common with a number of other sub-systems of the IMS T9000) is controlled via registers in the configuration space. The registers are accessed via the *ldconf* and *stconf* instructions, or via *CPeek* and *CPoke* command messages received along control link **CLink0**.

Three of the CPU configuration registers are described below. In addition to these three registers there are a number of CPU configuration registers which are shared between the VCP and also the scheduler, details of these are given in the Communication chapter 11. For the complete list of registers refer to chapter 15, Register Reference Guide.

Reason

The **Reason** register contains a reason code if the IMS T9000 is halted for any reason. The reason code is either a trap reason code, indicating that an L-process with a null trap handler caused an unmasked error, or one of 'stopped', 'halted' or 'external memory error'. Further details on the reason code can be found in the *T9000 Transputer Instruction Set Manual*.

EmiBadAddress

The address of the memory error is stored in the **EmiBadAddress** register if the reason code is 'external memory error'.

A reason code of 'external memory error' is returned if a memory error other than an access error is returned by the memory interface and if it is detected by the processor. Memory errors not detected by the processor, for example VCP memory accesses, cause a 'halted' reason code.

The **Reason** and **EmiBadAddress** registers must be read after resetting but before rebooting the IMS T9000, by the control link *CPeek* command, as the values of these registers become undefined after rebooting.

InitialPtr and InitialWptr

After reset the processor is initially inactive. A **Wptr** and **lptr** are written into the **InitialPtr** and **InitialWptr** configuration registers by the control unit over the configuration bus, which the processor uses when it starts executing.

4 Floating point unit

The IMS T9000 has an on-chip scalar floating point unit (FPU) which performs floating point operations sustaining a rate of 15 Mflops and a peak of 25 Mflops at a processor speed of 50 MHz.

The FPU has been designed to operate on both single precision (REAL32) and double precision (REAL64) floating point numbers, and returns results which fully conform to the IEEE 754-1985 floating point arithmetic standard. The FPU incorporates hardware to perform divide and square root. Denormalized numbers are fully supported in hardware. All rounding modes defined by the standard are implemented, with the default being round to nearest.

4.1 Floating point registers

The FPU consists of a computing engine with a three deep floating point evaluation stack for manipulation of floating point numbers and a status register.

4.1.1 Floating-point stack

The FPU contains a three deep stack of floating-point registers:

FPAreg	floating-point stack register A
FPBreg	floating-point stack register B
FPCreg	floating-point stack register C

Each floating-point register can hold a value in either single precision or double precision floating-point format and has a tag associated with it (stored in the floating-point status register) to signify the precision of the data it contains. The floating-point stack behaves in a similar manner to the integer stack. When a value is loaded in **FPAreg** the values in **FPAreg** and **FPBreg** are pushed down into **FPBreg** and **FPCreg** respectively. When a value is stored from **FPAreg**, **FPBreg** is popped into **FPAreg** and **FPCreg** into **FPBreg**.

The addresses of floating point values are formed on the integer stack, and values are transferred between the addressed memory locations and the floating point stack.

The representation of single precision and double precision floating point numbers in memory is as set out in the IEEE standard.

- Single precision format floating-point values are represented in memory within a single machine word (32-bit).
- Double precision format floating-point values are represented in memory by two contiguous machine words. The word which contains the sign bit is held at the memory location with the higher address of the two.

4.1.2 Floating-point status register

The currently executing process has a floating-point status word associated with it. This is stored within the floating-point status register (**FPstatusReg**). The type information and rounding modes are detailed in **FPstatusReg**, see table 4.1, and can be accessed using the *fpdall*, *fpstall*, *ldshadow* and *stshadow* instructions.

Bit position	Bit field	Function
1:0	Roundmode	Rounding mode
3:2	FPA type	Type of floating-point value in FPAreg
5:4	FPB type	Type of floating-point value in FPBreg
7:6	FPC type	Type of floating-point value in FPCreg
31:8		INMOS reserved

Table 4.1 **FPstatusReg** fields

Note: All INMOS reserved bits must always be written with 0's.

When an operation yields a floating-point result, this result is by default rounded to the nearest representable value to the exact result (*Round-to-nearest*). In addition to this default, the other three rounding modes specified in the IEEE standard are provided on the IMS T9000. The rounding mode can be set in the **FPstatusReg** to one of the four modes. This is reset as soon as the floating point load, operate or store is complete. The rounding mode is encoded into a two bit field and the binary value for each mode is shown in table 4.2.

Code	Rounding mode
00	IEEE round zero
01	IEEE nearest
10	IEEE round + infinity
11	IEEE round – infinity

Table 4.2 Floating-point rounding mode

The format of the floating-point value stored in any of the floating-point stack registers, can be single precision or double precision. This information is represented as shown in table 4.3. Note, the effect of loading any value other than 00 or 01 into these bits is undefined.

Code	Floating-point type
00	single precision – IEEE format 32-bit floating-point number
01	double precision – IEEE format 64-bit floating-point number

Table 4.3 Floating-point type

4.2 Floating point instructions

Instructions are provided to perform floating point arithmetic on the FPU. In the IMS T9000 all basic FPU operations can be performed by an equivalent single instruction coding, the names of these instructions begin with *fp*. These instructions allow floating point values to be transferred from memory to the FPU evaluation stack and vice versa, and to manipulate values on the FPU evaluation stack.

Operation	T9000 running at 40 MHz		T9000 running at 50 MHz	
	Single length	Double length	Single length	Double length
add	50 ns	50 ns	40 ns	40 ns
subtract	50 ns	50 ns	40 ns	40 ns
multiply	50 ns	75 ns	40 ns	60 ns
divide	200 ns	375 ns	160 ns	300 ns
square root	200 ns	375 ns	160 ns	300 ns

Timing is for operations where both operands are normalized floating point numbers

Table 4.4 Typical floating point operation times

Further details on the operation of the FPU and the floating point instructions can be found in the *T9000 Transputer Instruction Set Manual*.

5 Memory management

This chapter describes the protection and memory management system of the IMS T9000. The memory management mechanism in the IMS T9000 transputer is designed to support the development and debugging of programs, to allow the safe execution of programs written in insecure languages, and to support address translation. It also supports the dynamic extension of a calling stack. The mechanism does not provide virtual memory, or page-based memory protection.

The memory management mechanism is only invoked for a special type of protected process - known as a P-process. A P-process is run under the control of a parent L-process, known as the *supervisor* (or sometimes the *stub*). A P-process is created by the supervisor process executing a *goprot* (go protected) instruction. This instruction loads the state of the P-process from memory, loads the memory management registers, and starts to execute the P-process.

The P-process may execute only a subset of the IMS T9000 instruction set, known as privileged instructions. The addresses of all memory accesses generated by the P-process are treated as logical addresses; they are checked and translated into physical addresses by hardware. If the P-process attempts to access an illegal address, execute a privileged instruction, or causes an error, control is returned to the supervisor process. Control will also be returned to the supervisor process if the P-process exceeds its timeslice or executes a *syscall* (system call) instruction. When a trap occurs the P-process's state is saved to memory and the supervisor process is restarted.

5.1 Protection, stack extension, and logical to physical address translation

5.1.1 Protection

The memory management mechanism in the IMS T9000 provides for the checking and translation of four independently sized regions of addresses. The P-process may read from any region, but may only write to or execute code out of regions which have the appropriate permissions. All read, write and instruction fetch accesses attempted by the executing P-process are checked. If an illegal access is attempted then the P-process traps back to the supervisor process. It is not normally possible to continue execution of a P-process after an illegal access has been attempted.

5.1.2 Stack extension

In addition to checking the validity of memory accesses, the hardware checks that the location pointed to by the workspace pointer (**Wptr**) is writable. If a *call*, *ajw* (adjust workspace) or *gajw* (general adjust workspace) instruction causes the workspace pointer to address a non-writable address then the P-process traps. However, in this case, the supervisor process can restart execution of the P-process after extending the region. In this way it is possible to execute stack extension on demand.

5.1.3 Logical to physical address translation

A region may be of size 2^n bytes, with a minimum size of 256 bytes (64 words) and a maximum size of 2^{30} bytes. A region of size 2^n bytes may be translated onto any 2^n byte boundary in the physical address space. The physical addresses associated with the four regions must not overlap. The legal logical addresses within a region either occupy the top 2^n addresses within that region or occupy the bottom 2^n addresses within that region. A consequence of this is that, except for when the maximal sized region (2^{30} bytes) is in use, it is possible to ensure that the addresses 0 and #80000000, which are commonly used as null pointers, do not correspond to legal addresses and so access to such an address is immediately detected as a violation.

5.2 Regions

The logical address space of a P-process is divided into four regions. Each region is sized, assigned access permissions, and has its address accesses translated independently of the others. The two most significant bits of a logical address are used to determine to which region reference is being made. The terms *region 0*, *region 1*, *region 2*, and *region 3* are used to refer to the regions having addresses with the most significant bits set to 00, 01, 10 and 11 respectively.

Associated with each region is a region descriptor which:

- identifies the region as read-only, read-write, read-execute or read-write-execute.
- indicates whether valid addresses are located in the bottom or the top of the region.
- specifies the size of the region and the address of the physical region to which the logical region should be relocated.
- indicates whether device access instructions must be used to access the region. (Refer to the *T9000 Transputer Instruction Set Manual* for details on the device access instructions).

The legal logical addresses within a region either occupy the top 2^n addresses within that region or occupy the bottom 2^n addresses within that region. The following table shows the legal addresses within each region. The memory mapping for the logical addresses is illustrated in figure 5.1.

Region	Size	Positioned from top of region		Positioned from bottom of region	
		Most positive address	Most negative address	Most positive address	Most negative address
0	2^l	$2^{30} - 1$	$2^{30} - 2^l$	$2^l - 1$	0
1	2^k	$2^{31} - 1$	$2^{31} - 2^k$	$2^{30} + 2^k - 1$	2^{30}
2	2^n	$-2^{30} - 1$	$-2^{30} - 2^n$	$-2^{31} + 2^n - 1$	-2^{31}
3	2^m	-1	-2^m	$-2^{30} + 2^m - 1$	-2^{30}

Table 5.1 Region addresses

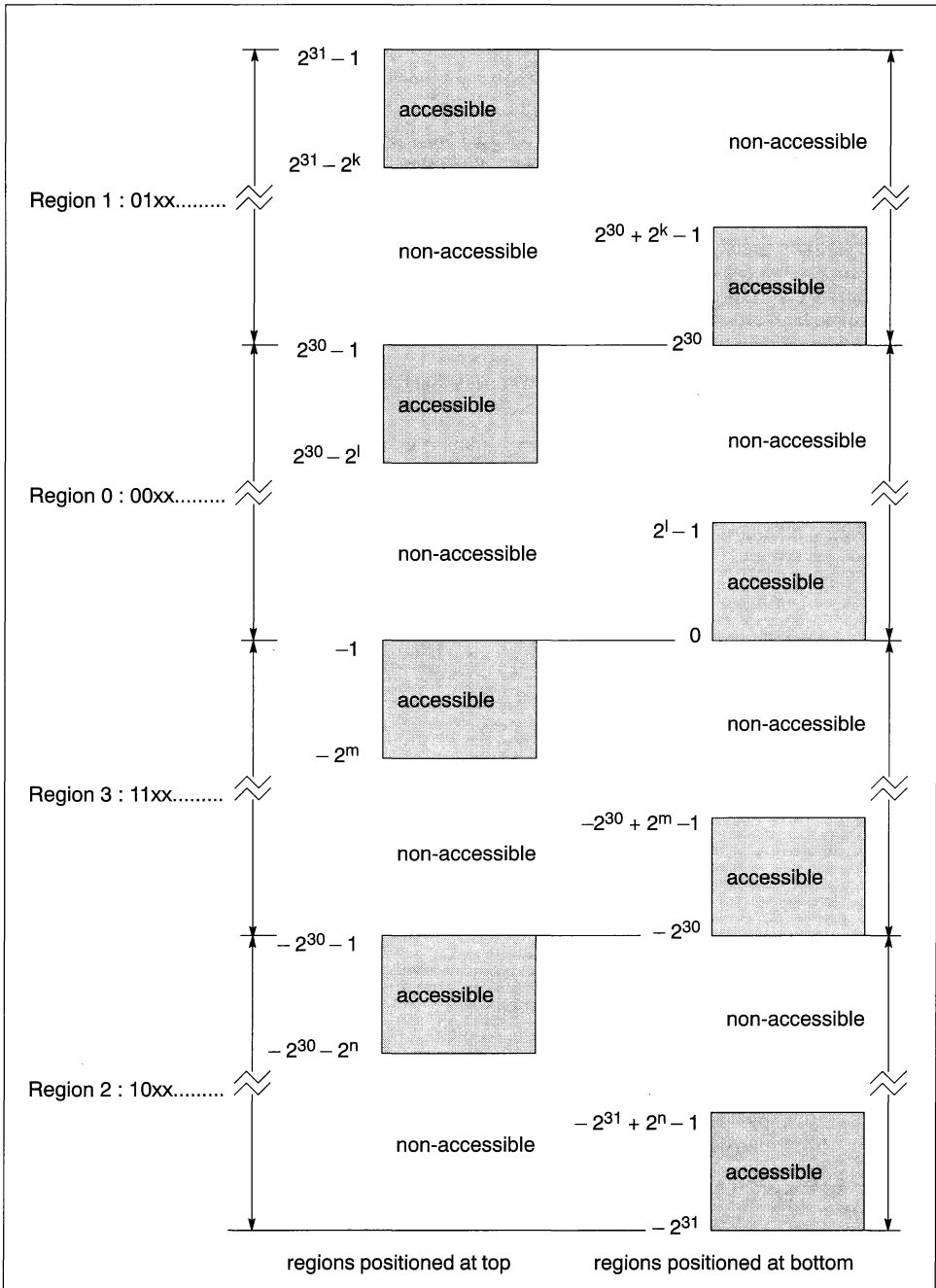


Figure 5.1 Position of region addresses in logical memory space

5.2.1 Region descriptors

A region descriptor defines the size of a region, the position of the logical region, the address translation, and the write and instruction fetch permissions (write-permit and execute-permit respectively) associated with that region.

A region descriptor is a single word. Bit 0 indicates whether writes may be made to the region (1 = write-permit). Bit 1 indicates whether instructions may be fetched from the region (1 = execute-permit). Bit 2 indicates the position of the logical region (1 = top, 0 = bottom). Bit 3 indicates the device requirement (1 = device-only, 0 = any).

The remaining bits specify the size of the region and the address of the physical region to which the logical region should be relocated. For a region of size 2^n bytes, bit $n-1$ is set to 1. All bits below bit $n-1$ are set to 0 (except for the write-permit, execute-permit, position and device requirement bits; bits 0, 1, 2 and 3). The remaining high-order bits, bits 31 through n , are used to replace the corresponding bits in the logical address which is being translated.

Note that the minimum region size of 256 bytes implies that bits 4 through 6 of the region descriptor **must** be set to 0.

A region can be set to have zero size by programming its region descriptor with the null descriptor, #8000000. A number of invalid region descriptors exist, and these should not be used.

An example of a logical to physical address translation which is positioned at the top of region 2 is shown in the following diagram. This region has execute permission and is read-only.

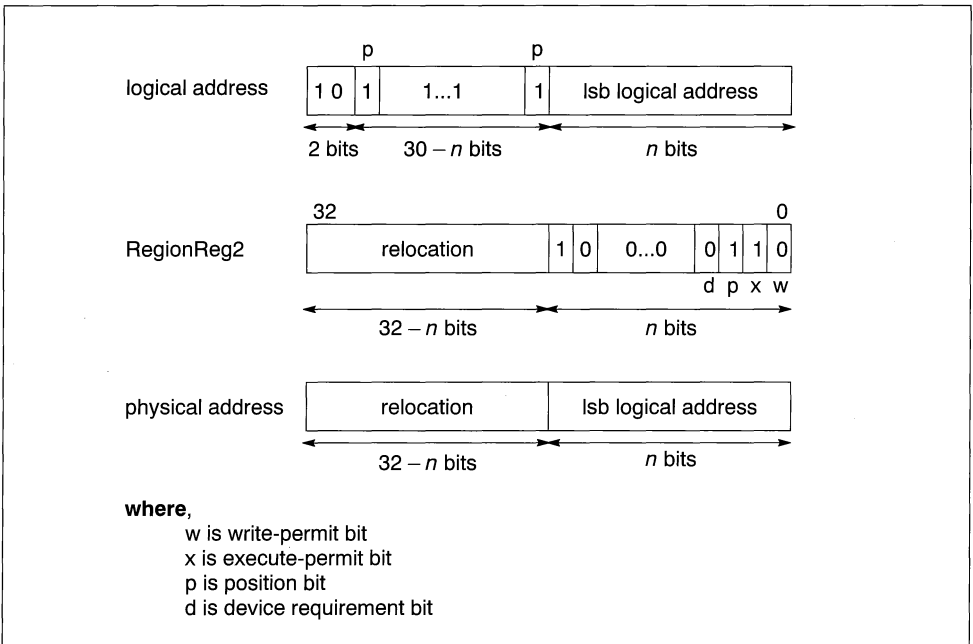


Figure 5.2 Logical to physical address translation

Note that for the logical address to be valid, bits n through to 29 must be 1's if the position bit (bit 2) in the region descriptor is set to 1, and must be 0's if the position bit is 0.

5.2.2 Non-overlapping regions

The translated regions must not overlap, that is no two distinct logical addresses may translate to the same physical address. If the regions do overlap the effect is undefined.

5.3 P-process machine registers

The register state of a P-process is similar to the register state of an L-process but also contains protection registers. The IMS T9000 transputer has the following registers related to the operation of memory management for P-processes. The operation of a P-process requires that the supervisor causes its state to be loaded and then causes the P-process to execute under protection until a trap occurs. When the trap occurs the P-process's state must be saved and the supervisor re-started.

The operations of loading and saving P-process state is shared between hardware mechanisms of the IMS T9000 and the supervisor program. The hardware mechanism provides for the loading and storing of state contained in Pstate vector, which consists of the **StatusReg**, watchpoint registers, **Wptr**, **lptrReg**, **Areg**, **Breg**, **Creg**, **Ereg** and **Xreg**, and the loading of the protection registers. Other registers, such as the floating point registers, are loaded and saved by the supervisor executing the appropriate instructions.

Register	Description
RegionReg0	Register descriptor for region 0
RegionReg1	Register descriptor for region 1
RegionReg2	Register descriptor for region 2
RegionReg3	Register descriptor for region 3
PstateReg	Pointer to the P-process state vector
WdescStubReg	Process descriptor of the supervisor

Table 5.2 Memory management registers

The **RegionReg0**, **RegionReg1**, **RegionReg2**, **RegionReg3** registers contain the region descriptors for region 0, region 1, region 2, region 3 respectively. As described above, the region descriptor defines the size, position, physical address and permissions of a region as a single 32-bit word.

The **PstateReg** register contains a pointer to a block of memory where the state of the executing P-process is to be saved when it traps to its supervisor process.

The **WdescStubReg** register contains the workspace descriptor of the supervisor process which is controlling the execution of the current P-process.

5.4 Debugging

The support provided for debugging a running P-process is an extension of that provided for L-processes, which is described in section 3.5. However, whenever the trap-handler would have been invoked for an L-process, for a P-process control is returned to its supervisor process. The supervisor process is responsible for taking any necessary action. Thus, the following debug operations will cause control to be returned to the supervisor process.

- a *j0* instruction acting as a breakpoint
- execution of a single instruction when single-stepping enabled
- a write access to the watchpoint region

6 Instruction set

This chapter provides information on the IMS T9000 instruction set. It contains tables listing all the instructions and where applicable provides details of the number of processor cycles taken by an instruction. For a more complete description of the instructions and their use refer to the *T9000 Transputer Instruction Set Manual*.

The transputer instruction set has been designed for simple and efficient compilation of high-level languages. All instructions have the same format, designed to give a compact representation of the operations occurring most frequently in programs.

Each instruction consists of a single byte divided into two 4-bit parts. The four most significant bits of the byte are a function code and the four least significant bits are a data value.

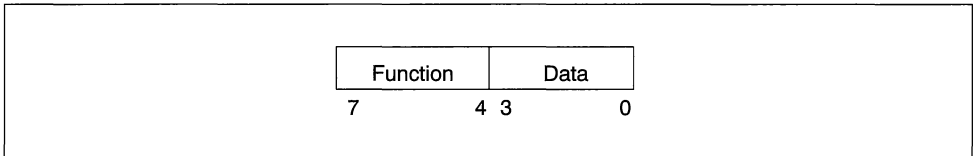


Figure 6.1 Instruction format

6.1 Efficiency of encoding

Measurements show that about 70% of executed instructions are encoded in a single byte; that is, without the use of prefix instructions. Many of these instructions, such as *ldl* (load local) and *add* require just one processor cycle or less with grouping.

The instruction representation gives a more compact representation of high-level language programs than more conventional instruction sets. Since a program requires less store to represent it, less of the memory bandwidth is taken up with fetching instructions. Furthermore, as memory is word accessed the processor will receive four instructions for every fetch.

Short instructions also improve the effectiveness of instruction pre-fetch, which in turn improves processor performance. There is a pre-fetch buffer which contains several words, so the processor rarely has to wait for an instruction fetch before proceeding. Since the buffer is transparent on jumps, there is little time penalty when a jump instruction causes the buffer contents to be discarded.

6.2 Interaction of the processor pipeline and the instruction set

The IMS T9000 has a pipelined processor with 5 pipeline stages. Each stage is dedicated to a particular operation, which in the main correspond to individual instructions, although even some of the simple instructions are operated on in more than one pipeline stage.

Stage	Operation	Function
0	Local	Push constants and locals onto the execution stack.
1	Address	Calculate addresses of non-local operands.
2	Read	Read non-local variables.
3	ALU	Stack-based ALU and FPU operations.
4	Conditional Jump/Store	Conditional jump or write results back to memory.

Table 6.1 Pipeline stages

The IMS T9000 treats commonly occurring sequences of instructions as if they were a single 'grouped' operation. The pipelined execution unit is able to execute several groups at the same time. Most groups

execute in one cycle, thus delivering an instruction rate well in excess of one instruction per cycle. An example of decoding is shown below:

Program	Mnemonic	Group
$x := 0$	<i>ldc 0; stl x</i>	1st group
$y := \#24$	<i>prefix 2; ldc 4; stl y</i>	2nd group
$w := x + y$	<i>ldl x; ldl y; add</i>	3rd group
	<i>stl w</i>	4th group
$z := w + (x + y)$	<i>ldl x; ldl y; add</i>	5th group
	<i>ldl w; add; stl z</i>	6th group
$e[0] := a[3] + b[4]$	<i>ldl a; ldnl 3; ldl b; ldnl 4; add</i>	7th group
	<i>ldl e; stnl 0</i>	8th group
$b[j] := a[i]$	<i>ldl i; ldl a; wsub; ldnl 0</i>	9th group
	<i>ldl j; ldl b; wsub; stnl 0</i>	10th group

Table 6.2 Expression evaluation

Evaluation of expressions sometimes requires use of temporary variables in the workspace, but the number of these can be minimized by careful choice of the evaluation order.

Groups commonly take one cycle at each stage in the pipeline, so that as groups are passed continuously down the pipeline one group is executed per cycle. However, a number of factors may cause a group to take more than one cycle at a given stage in the pipeline. These are enumerated below:

- 1 **Long ALU/FPU operations:** Most ALU/FPU operations take one cycle; those frequently used instructions which take longer are shown in the table below. The processor cycles column of the instruction set tables detail all instructions which take longer than one cycle.

Operation	Cycles	Notes
<i>prod</i>	2 – 5	
<i>mul</i>	2 – 5	
<i>div</i>	5 – 12	
<i>rem</i>	6 – 13	
<i>lmul</i>	3 – 6	
<i>ldiv</i>	15	
<i>lshr</i>	2	
<i>lshl</i>	2	
<i>crcbyte</i>	4	
<i>crcword</i>	16	
<i>fpadd</i>	2	1
<i>fpsub</i>	2	1
<i>fpmul</i> (single)	2	1
<i>fpmul</i> (double)	3	1
<i>fpdiv</i> (single)	8	1
<i>fpdiv</i> (double)	15	1

table continued overleaf

table continued from previous page

Operation	Cycles	Notes
<i>fprem</i> (single)	5 – 74	1
<i>fprem</i> (double)	5 – 529	1
<i>fprange</i> (single)	5 – 11	1
<i>fprange</i> (double)	5 – 18	1
<i>fpsqrt</i> (single)	8	1
<i>fpsqrt</i> (double)	15	1

Table 6.3 Speed of ALU/FPU operations

Notes:

- 1 These figures assume normalized values, there is a 2 cycle overhead for each denormalized operand or result (except there is no overhead for a denormalized result from *fprem*).
- 2 **Stack conflicts:** There are occasions when a group will produce a value on the integer or floating point evaluation stack which will then be used by the following group. If the following group requires it in an earlier pipeline stage than it is produced in, then the group will have to wait. This occurs mainly with the subscript instructions. Table 6.4 below shows the stages in which values are produced and consumed. If a value is produced and pushed onto the stack in stage n in a particular group, and is consumed in stage m in the following group, then $n - m$ extra cycles will have to be allowed for.

Instruction	Stage	
	Consumed	Produced
<i>ldc</i>		0
<i>ldl</i>		0
<i>ldlp</i>		0
<i>mint</i>		0
<i>ldnlp</i>	1	1
All subscript instructions	1	1
<i>ldnl</i>	1	2
<i>load16</i>	1	2
<i>lb</i>	1	2
All ALU and FPU instructions	3	3
<i>cj</i>	4	
All store instructions	4	

Table 6.4 Stages in which instructions operate

- 3 **Load/store conflicts:** Stores occur in later pipeline stages than loads, so if the load is to the same address as the store, the memory is not yet in the state that the group expects it to be in. When this happens, the second group proceeds until the operand that would have been loaded is actually used, at which point it waits until the data that is to be written has passed it. All writes generate their values at stage 4, which are then consumed in either stages 1 or 3. If it is in stage 3, then there will be no penalty, but there will be a 2 cycle penalty when the value is consumed in stage 1. The load may not occur in the immediately following cycle, but in the subsequent one, in which case any penalty is one cycle less.

- 4 **Jumps:** A jump causes a pipeline to be (partially) empty while the instruction at the destination address is fetched and decoded. The number of cycles added to the normal time for a group is given in the following table:

Instruction	Cycles	Notes
<i>j</i>	2	1
<i>cj</i> (taken)	4	1
<i>lend</i> (loop back)	2	1
<i>lend</i> (terminate)	5	1
<i>call</i>	3	1
<i>ret</i>	2	1

Table 6.5 Jumps

Notes:

- 1 These figures assume cache hits, if cache misses occur it may take longer, dependent on the PMI speed.

6.3 Instruction characteristics

Tables 6.8 to 6.37 give the complete set of instructions grouped by function.

The Primary Instructions table 6.8 gives the basic function code. Where the operand is less than 16, a single byte encodes the complete instruction. If the operand is greater than 15, one prefix instruction (*prefix*) is required for each additional four bits of the operand. If the operand is negative the first prefix instruction will be *nfix*. Examples of *prefix* and *nfix* coding are given in table 6.6.

For secondary instructions, which do not have an operand, the memory code is given in the instruction tables.

Mnemonic	Function code	Memory code
<i>ldc</i> #3	#4	#43
<i>ldc</i> #35		
is coded as		
<i>prefix</i> #3	#2	#23
<i>ldc</i> #5	#4	#45
<i>ldc</i> #987		
is coded as		
<i>prefix</i> #9	#2	#29
<i>prefix</i> #8	#2	#28
<i>ldc</i> #7	#4	#47
<i>ldc</i> -31 (<i>ldc</i> #FFFFFFE1)		
is coded as		
<i>nfix</i> #1	#6	#61
<i>ldc</i> #1	#4	#41

Table 6.6 *prefix* coding

The load device identity (*lddevice*) instruction (table 6.29) pushes the device type identity into the **Areg** register. Each product is allocated a unique group of numbers for use with the *lddevice* instruction.

Where applicable the instruction set tables contain a processor cycles column. This refers to the number of cycles taken by an instruction.

If a floating point instruction has one or more zero, infinity or NaN operands (except *fpaddbsn* with a single zero operand) it will execute in 1 cycle.

There are a number of errors that can be trapped. When this occurs, an error code is returned to the trap handler. Any instruction which is not in the instruction set tables is an invalid instruction and is flagged illegal, returning an error code to the trap handler.

The **Notes** column of the tables indicates the descheduling and error features of an instruction as described in table 6.7. It also indicates which instructions cannot be used in P-processes.

Ident	Feature
E	Error can be explicitly set
O	Integer overflow / divide by zero error
U	Unaligned memory access to word / half word
M	Invalid memory address for P-process
i	IEEE invalid operation exception
z	IEEE divide by zero exception
o	IEEE overflow exception
u	IEEE underflow exception
x	IEEE inexact exception
t	T800 FPU error exception
I	Interruptible instruction
B	Instruction can cause a breakpoint
T	Timesliceable instruction
P	Instruction not allowed in P-process
D	Instruction is a descheduling point
d	Each denormalized operand or result incurs an additional 2 processor cycles

Table 6.7 Instruction features

6.4 Instruction set tables

6.4.1 Primary instructions

Function Code	Mnemonic	Name	Notes
0	j n	jump	B,T,D
1	ldlp n	load local pointer	
2	pfix n	prefix	
3	ldnl n	load non-local	M,U
4	ldc n	load constant	
5	ldnlp n	load non-local pointer	
6	nfix n	negative prefix	
7	ldl n	load local	M
8	adc n	add constant	O
9	call n	call	M
A	cj n	conditional jump	
B	ajw n	adjust workspace	M
C	eqc n	equals constant	
D	stl n	store local	M
E	stnl n	store non-local	M,U
F	opr n	operate	

Table 6.8 IMS T9000 primary instructions

6.4.2 Secondary instructions

Sequential instructions

Memory Code	Mnemonic	Processor cycles	Name	Notes
24F6	and	1	and	
24FB	or	1	or	
23F3	xor	1	exclusive or	
23F2	not	1	bitwise not	
24F1	shl	1	shift left	
24F0	shr	1	shift right	
F5	add	1	add	O
FC	sub	1	subtract	O
25F3	mul	2 – 5	multiply	O
27F2	fmul	3 – 6	fractional multiply	O
22FC	div	5 – 12	divide	O
21FF	rem	6 – 13	remainder	O
F9	gt	1	greater than	
25F5	gtu	1	greater than unsigned	
F4	diff	1	difference	
25F2	sum	1	sum	
F8	prod	2 – 5	product	

Table 6.9 IMS T9000 arithmetic and logical instructions

Memory Code	Mnemonic	Processor cycles	Name	Notes
21F6	ladd	1	long add	O
23F8	lsub	1	long subtract	O
23F7	lsum	1	long sum	
24FF	ldiff	1	long diff	
23F1	lmul	3 – 6	long multiply	
21FA	ldiv	15	long divide	O
23F6	lshl	2	long shift left	
23F5	lshr	2	long shift right	
21F9	norm	2 – 3	normalize	

Table 6.10 IMS T9000 long arithmetic instructions

Memory Code	Mnemonic	Name	Notes
22F0	ret	return	M
21FB	ldpi	load pointer to instruction	
23FC	gajw	general adjust workspace	M,U
F6	gcall	general call	
22F1	lend	loop end	M,T,U,D

Table 6.11 IMS T9000 jump and call instructions

Memory Code	Mnemonic	Name	Notes
24FA	move	move message	M,I
25FB	move2dinit	initialize data for 2D block move	
25FC	move2dall	2D block copy	M,I
25FD	move2dnnonzero	2D block copy non-zero bytes	M,I
25FE	move2dzero	2D block copy zero bytes	M,I

Table 6.12 IMS T9000 block move instructions

Memory Code	Mnemonic	Name	Notes
F2	bsub	byte subscript	
FA	wsub	word subscript	
28F1	wsubdb	form double word subscript	
2CF1	ssub	sixteen subscript	
23F4	bcnt	byte count	
23FF	wcnt	word count	
F1	lb	load byte	M
23FB	sb	store byte	M
2CFA	ls	load sixteen	MU
2CF8	ss	store sixteen	MU
2BF9	lbox	load byte and sign extend	M
2FF9	lsx	load sixteen and sign extend	MU

Table 6.13 IMS T9000 indexing/array instructions

Memory Code	Mnemonic	Processor cycles	Name	Notes
2CF7	cir	2	check in range	E
2CFC	ciru	2	check in range unsigned	E
2BFA	cb	–	check byte	E
2BFB	cbu	–	check byte unsigned	E
2FFA	cs	–	check sixteen	E
2FFB	csu	–	check sixteen unsigned	E
25F6	cword	–	check word	E
24FC	csngl	–	check single	E
21F3	csub0	–	check subscript from 0	E
24FD	ccnt1	–	check count from 1	E
2FF8	xsword	–	sign extend sixteen to word	
2BF8	xbword	–	sign extend byte to word	
23FA	xword	–	extend to word	
21FD	xdouble	–	extend to double	

Table 6.14 IMS T9000 range checking and conversion instructions

Memory Code	Mnemonic	Name	Notes
2FF0	devlb	device load byte	M
2FF2	devls	device load sixteen	M,U
2FF4	devlw	device load word	M,U
62F4	devmove	device move	M,I
2FF1	devsb	device store byte	M
2FF3	devss	device store sixteen	M,U
2FF5	devsw	device store word	M,U

Table 6.15 IMS T9000 device access instructions

Memory Code	Mnemonic	Processor cycles	Name	Notes
27F4	crcword	16	calculate CRC on word	
27F5	crcbyte	4	calculate CRC on byte	
27F6	bitcnt	8	count bits set in word	
27F7	bitrevword	1	reverse bits in word	
27F8	bitrevnbits	1	reverse bottom n bits in word	

Table 6.16 IMS T9000 CRC and bit instructions

Memory Code	Mnemonic	Name	Notes
F0	rev	reverse	
25FA	dup	duplicate top of stack	
27F9	pop	pop processor stack	
63F0	nop	no operation	
24F2	mint	minimum integer	

Table 6.17 IMS T9000 general instructions

Memory Code	Mnemonic	Name	Notes
22F2	ldtimer	load timer	
25F4	sttimer	store timer	P
22FB	tin	timer input	P,D,I
24FE	talt	timer alt start	P
25F1	taltwt	timer alt wait	P,D,I
24F7	enbt	enable timer	P
22FE	dist	disable timer	P,I

Table 6.18 IMS T9000 timer handling instructions

Communication instructions

Memory Code	Mnemonic	Name	Notes
F7	in	input message	P,D,I,E,U
FB	out	output message	P,D,I,E,U
FF	outword	output word	P,D,I,E,U
FE	outbyte	output byte	P,D,I,E,U

Table 6.19 IMS T9000 input/output instructions

Memory Code	Mnemonic	Name	Notes
2CF0	ldcnt	load message byte count	P,E
61FC	vin	variable-length input message	P,I,E,U,D
61FD	vout	variable-length output message	P,I,E,U,D

Table 6.20 IMS T9000 variable length input/output instructions

Memory Code	Mnemonic	Name	Notes
2CF9	chantype	channel type	P,E,U
61F6	initvlcb	initialize VLCB	P,E,U
2CF3	ldchstatus	load channel status	P,E,U
21F2	resetch	reset channel	P,E,U
61F7	setchmode	set channel mode	P,E,U
61FE	stopch	stop virtual channel	P,E,U,D
61F8	sethdr	set virtual channel header	P,E,U
61F5	writ(hdr)	write virtual channel header	P,E,U,I
61F4	readhdr	read virtual channel header	P,E,U,I
2BFC	insphdr	inspect virtual channel header	P,E,U
61F9	swapbfr	swap buffer pointer in VLCB	P,E,U
2BFD	readbfr	read buffer pointer from VLCB	P,E,U

Table 6.21 IMS T9000 channel and virtual link instructions

Memory Code	Mnemonic	Name	Notes
61F1	grant	grant resource	P,U,D
61F2	enbg	enable grant	P,U
61F3	disg	disable grant	P,U
62F8	ldrespnr	load resource queue pointer	P,E,U
62F9	strespnr	store resource queue pointer	P,E,U
62FA	erdsq	empty resource data structure queue	P,U
62FB	irdsq	insert at front of RDS queue	P,U
62FC	mkrc	mark resource channel	P,E,U
62FD	unmkrc	unmark resource channel	P,E,U

Table 6.22 IMS T9000 resource channel instructions

Memory Code	Mnemonic	Name	Notes
60F5	wait	wait	P,U,O,D
60F4	signal	signal	P,U,O

Table 6.23 IMS T9000 semaphore instructions

Memory Code	Mnemonic	Name	Notes
24F3	alt	alt start	P
24F4	altwt	alt wait	P,D
24F5	altend	alt end	P
24F9	enbs	enable skip	P
23F0	diss	disable skip	P
24F8	enbc	enable channel	P,E,U
22FF	disc	disable channel	P,E,U
24FE	talt	timer alt start	P
25F1	taltwt	timer alt wait	P,D,I
24F7	enbt	enable timer	P
22FE	dist	disable timer	P,I
61F2	enbg	enable grant	P,U
61F3	disg	disable grant	P,U

Table 6.24 IMS T9000 alternative instructions

Process scheduling instructions

Memory Code	Mnemonic	Name	Notes
FD	startp	start process	P,U
F3	endp	end process	P,D,U
23F9	runp	run process	P
21F5	stopp	stop process	P,D
21FE	ldpri	load current priority	

Table 6.25 IMS T9000 scheduling instructions

Memory Code	Mnemonic	Name	Notes
60F0	swapqueue	swap scheduler queue	P
60F1	swaptimer	swap timer queue	P
60F2	insertqueue	insert at front of scheduler queue	P
2BF0	settimeslice	set timeslicing status	P
60F3	timeslice	timeslice	T,D

Table 6.26 IMS T9000 process queue manipulation and timeslicing instructions

Memory Code	Mnemonic	Name	Notes
2CF4	intdis	interrupt disable	P
2CF5	intenb	interrupt enable	P
60FE	fpldall	floating point load all	M,U
60FF	fpstall	floating point store all	M,U
61F0	stmoves2dinit	store move2dinit data	M,U
60FC	ldshadow	load shadow registers	P,U
60FD	stshadow	store shadow registers	P,U

Table 6.27 IMS T9000 interrupt instructions

Memory Code	Mnemonic	Name	Notes
2CF2	ldth	load trap handler	P
60F9	selth	select trap handler	P,D,U
2BF6	ldflags	load error flags	
2BF7	stflags	store error flags	
60FA	goprot	go protected	P,U
62FE	restart	restart	P,U
60FB	tret	trap return	P
60F8	syscall	system call	
62FF	causeerror	cause error	

Table 6.28 IMS T9000 trap handler instructions

Initialization and configuration instructions

Memory Code	Mnemonic	Name	Notes
22FA	testpranal	test processor analysing	P
25F4	sttimer	store timer	P
2127FC	lddevid	load device identity	
27FE	ldmemstartval	load value of memstart address	P
68FC	ldprodid	load product identity	

Table 6.29 IMS T9000 processor initialization instructions

Memory Code	Mnemonic	Name	Notes
2BFE	ldconf	load from configuration register	P,E
2BFF	stconf	store to configuration register	P,E

Table 6.30 IMS T9000 configuration instructions

Cache operation instructions

Memory Code	Mnemonic	Name	Notes
62F0	fdca	flush dirty cache address	M
62F2	fdcl	flush dirty cache line	P
62F1	ica	invalidate cache address	M
62F3	icl	invalidate cache line	P

Table 6.31 IMS T9000 cache instructions

Floating point instructions

Memory Code	Mnemonic	Processor cycles		Name	Notes
		REAL32	REAL64		
28F7	fpadd	2	2	floating point add	i,o,u,x,t,d
28F9	fpsub	2	2	floating point subtract	i,o,u,x,t,d
28FB	fpmul	2	3	floating point multiply	i,o,u,x,t,d
28FC	fpdiv	8	15	floating point divide	i,z,o,u,x,t,d
2DFB	fpabs	1	1	floating point absolute value	i,t
2DFA	fpexpinc32	2	2	floating point multiply by 2^{32}	i,o,u,x,t,d
2DF9	fpexpdec32	2	2	floating point divide by 2^{32}	i,u,x,t,d
2DF2	fpmulby2	2	2	floating point multiply by 2	i,o,u,x,t,d
2DF1	fpdivby2	2	2	floating point divide by 2	i,u,x,t,d
2CFF	fpem	5 – 74	5 – 529	floating point remainder	I,i,u,t,d
2DF3	fpsqrt	8	15	floating point square root	i,x,t,d
28FD	fprange	5 – 11	5 – 18	floating point range reduce	i,u,t,d
2DFD	fpaddbsn	2	2	floating point add double producing single	i,x,o,u,t,d

Table 6.32 IMS T9000 floating point arithmetic instructions

Memory Code	Mnemonic	Name	Notes
28FE	fpldnlsn	floating point load non-local single	M,U
28FA	fpldnldb	floating point load non-local double	M,U
28F6	fpldnlsni	floating point load non-local indexed single	M,U
28F2	fpldnldb	floating point load non-local indexed double	M,U
29FF	fpldzerosn	load zero single	
2AF0	fpldzerodb	load zero double	
2AFA	fpldnladdsn	floating point load non-local and add single	M,U,i,o,u,x,t,d
2AF6	fpldnladddb	floating point load non-local and add double	M,U,i,o,u,x,t,d
2AFC	fpldnlmulsn	floating point load non-local and multiply single	M,U,i,o,u,x,t,d
2AF8	fpldnlmuldb	floating point load non-local and multiply double	M,U,i,o,u,x,t,d
28F8	fpstnlsn	floating point store non-local single	M,U
28F4	fpstnldb	floating point store non-local double	M,U
29FE	fpstnli32	floating point store non-local int32	M,U

Table 6.33 IMS T9000 floating point load and store instructions

Memory Code	Mnemonic	Processor cycles		Name	Notes
		REAL32	REAL64		
29F4	fpgt	2	2	floating point greater than	i,t,d
29F5	fpeq	2	2	floating point equality	i,t,d
29F7	fpge	2	2	floating point greater than or equals	i,t,d
29FB	fplg	2	2	floating point less than or greater than	i,t,d
29F2	fporordered	1	1	floating point orderability	i,t
29F1	fpanan	1	1	floating point NaN	
29F3	fpointfinite	1	1	floating point not finite	
2DFE	fpchki32	2	2	check in range of int32	i,t
2DFF	fpchki64	2	2	check in range of int64	i,t

Table 6.34 IMS T9000 floating point comparison instructions

Memory Code	Mnemonic	Processor cycles	Name	Notes
2DF7	fpr32tor64	2	floating point real32 to real64	i,t,d
2DF8	fpr64tor32	2	floating point real64 to real32	i,o,u,x,t,d
29FD	fprtoi32	2 – 4	real to int32	i,x,t
29F6	fpi32tor32	2 – 4	int32 to real32	M,U,x
29F8	fpi32tor64	2	int32 to real64	M,U
29FA	fpb32tor64	2	bit32 to real64	M,U
2AF1	fpint	2	round to floating integer	i,x,t

Table 6.35 IMS T9000 floating point conversion instructions

Memory Code	Mnemonic	Name	Notes
2AF4	fprev	floating point reverse	
2AF3	fpdup	floating point duplicate	

Table 6.36 IMS T9000 floating point general instructions

Memory Code	Mnemonic	Name	Notes
2DF0	fprn	set rounding mode to round nearest	
2DF6	fprz	set rounding mode to round zero	
2DF4	fprp	set rounding mode to round plus	
2DF5	fprm	set rounding mode to round minus	

Note: These instructions take no time (0 cycles) if grouped with an operation.

Table 6.37 IMS T9000 floating point rounding instructions

7 Performance

The performance of the IMS T9000 is measured in terms of the number of (internal) processor cycles required to execute the program. The figures here relate to OCCAM programs. For the same function, other languages should achieve approximately the same performance as OCCAM.

The following tables are based on the time it takes to do ALU and FPU operations and it should be noted that many other instructions may be overlapped (see section 6.2).

7.1 Integer operations

These figures are estimates and give the minimum/maximum times for a particular operation.

Operation	Time (cycles)
Names	
variables	
in expressions	0
assigned to or input to	0 to 1
in PROC or FUNCTION call	0
channels	1
Array Variables (1-d)	
constant subscript	0
variable	0 to 1
plus subscript check	3
variable + constant subscript	0 to 1
plus subscript check	4
expression subscript	3
plus subscript check	1
Declarations	
CHAN OF <i>protocol</i>	2
[size] CHAN OF <i>protocol</i>	$3 + 5 * \text{size}$
PROC	0
Primitives	
assignment	0
input	15 or [5 + move]
output	16 or [5 + move]
STOP (call error handler)	7
SKIP	0
Arithmetic Operators	
+ -	1
*	2 to 5
/	5 to 12
REM	6 to 13
>> <<	1

table continued overleaf

table continued from previous page

Operation	Time (cycles)
Modulo Arithmetic Operators	
PLUS MINUS	1
TIMES	2 to 5
Boolean Operators	
OR	
first operand true	3 to 4
first operand false	0 to 1
AND	
first operand true	0
first operand false	3
NOT	0 to 1
Comparison Operators	
= <>	1
> <	1
>= <=	1
Bit Operators	
/\ \/ >< ~	1
Expressions	
constant	0
check if error	1
Timers	
timer input	1
timer AFTER	4 to ∞
ALT (timer)	20 to ∞
ALT guard	7 to ∞
Constructs	
SEQ	0
IF	0
IF guard	4
ALT (non timer)	11 to 17
ALT guard	7 to 16
PAR	$20 * \text{branches} - 6$
WHILE	$4 + 3 * \text{loops}$
Procedures and Function	
call and return	6 to 8
scalar parameter	0 to 1
array parameter	2

table continued overleaf

table continued from previous page

Operation	Time (cycles)
Replicators	
replicated SEQ	$(1 \text{ to } 3) + 3 * \textit{count}$
replicated IF	$(4 \text{ to } 6) + 3 * \textit{count}$
replicated ALT	$(13 \text{ to } 23) + (13 \text{ to } 22) * \textit{count}$
replicated timer ALT	$(2 \text{ to } \infty) + (13 \text{ to } \infty) * \textit{count}$
replicated PAR	$10 + 27 * \textit{count}$
range check on any of above	2

Table 7.1 Integer Performance

7.2 Floating point operations

All references to REAL32 or REAL64 operands within programs compiled for the IMS T9000 normally produce the following performance figures.

Operation	REAL32 Time (cycles)	REAL64 Time (cycles)	Notes
Names			
variables			
in expressions	0	0 to 1	
assigned to	0 to 1	1 to 2	
input to	1	1	
in PROC or FUNCTION call	0	0	
Arithmetic Operators			
+ -	2	2	1
*	2	3	1
/	8	15	1
SQRT	8	15	1
REM	5 to 74	5 to 529	1, 2
Comparison Operators			
= <>	2	2	
> <	2	2	
>= <=	2	2	
Conversions			
REAL32 to -		2	
REAL64 to -	2		
INT32 to -	2 to 4	2	
INT64 to -	10	10	
To INT32 from -	4	4	
To INT64 from -	11	11	

Table 7.2 Floating point performance

Notes:

- 1 These figures assume normalized values, there is a 2 cycle overhead for each denormalized operand or result (except there is no overhead for a denormalized result from *fprem*).
- 2 Typical value for REAL32 is 5 to 11; for REAL64 is 5 to 18, longer times are extremely rare.

7.3 Predefines

Operation	Time (cycles)
LONGADD	1
LONGSUM	1
LONGSUB	1
LONGDIFF	1
LONGPROD	3 to 6
LONGDIV	15
SHIFTRIGHT	2
SHIFLEFT	2
NORMALISE	2 to 3
ASHIFTRIGHT	3
ASHIFLEFT	4
ROTATERIGHT	3
ROTATELEFT	3
FRACMUL	3 to 6
BITCOUNT	8
CRCBYTE	4
CRCWORD	16
BITREVNBIT	1
BITREVWORD	1

Table 7.3 Predefines

8 Control system

This chapter describes the control system on the IMS T9000. The control system handles most of the general facilities necessary for the operation of the transputer.

8.1 Overview

The control system handles the following functions:

- initialization and configuration of the IMS T9000;
- loading and running a bootstrap program on the IMS T9000;
- resetting the IMS T9000;
- reporting errors and halting the IMS T9000.

An IMS T9000 is generally connected to a “controlling” processor, possibly another IMS T9000. A high-level protocol is defined for the controlling network to allow the controlling process to issue commands to, and receive responses from, devices in the control network. The controlling process sends command messages to the control system of the IMS T9000 which responds with handshake messages, see figure 8.1. The control system of the IMS T9000 can also send error messages to the controlling process, which responds with error handshakes. Commands are sent as packets with the first byte after the header containing a command code, which may be followed by additional data. Each command is terminated by an End of Message (EOM) token.

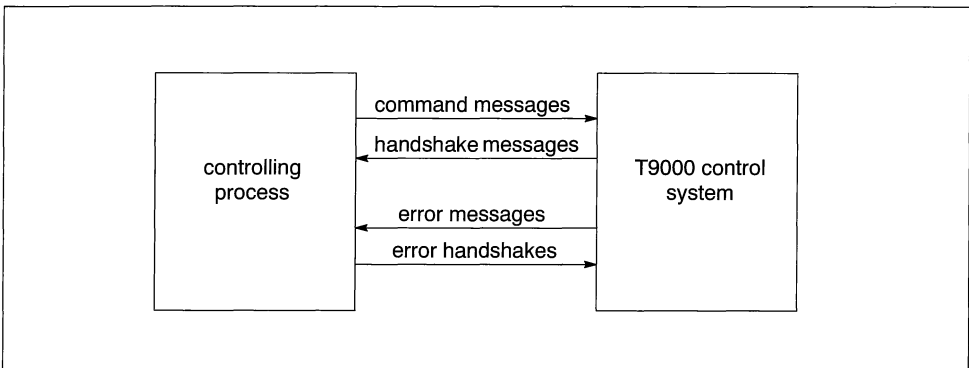


Figure 8.1 Communication between the controlling process and the IMS T9000 control system

The IMS T9000 has two bidirectional control links. They use the same electrical and packet level protocols as the four data links (see Data/Strobe links chapter 13), and a control link network is generally connected to one of the data links of a controlling IMS T9000.

One control link (**CLink0**) receives control and programming information and returns status information, the other (**CLink1**) enables all the devices in a system to be daisy-chained. This allows a simple physical connectivity to be used for the controlling network, as shown in figure 8.2. The routing information for each link of each device is programmed, via the control link, from the controlling processor.

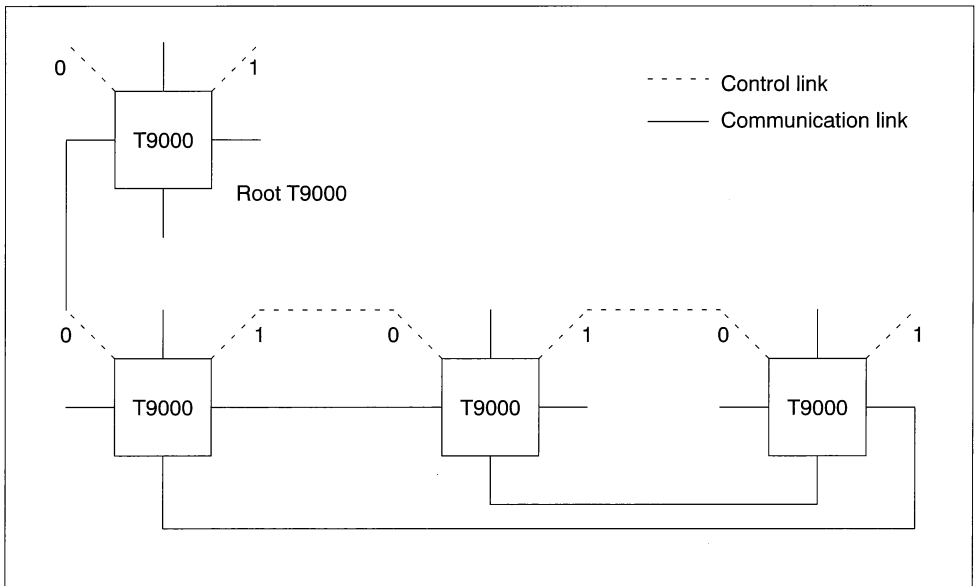


Figure 8.2 A daisy-chained control link network

The control links on all IMS T9000 transputer family products allow a separate control network to be used to assist in error handling and configuring, booting, resetting and analyzing processors and other components connected in a system, even in the presence of errors on the data communications links in the network.

For large systems IMS C104 routing devices can be used to connect the controlling network as a physical tree. Whatever the physical topology, the controlling network provides an independent virtual link between each device and the controlling processor.

8.1.1 Tiers of handshaking

There are three tiers of handshaking in the IMS T9000 control system.

- **Flow control**

Flow control tokens provide a low level of handshaking at the DS-Link level. Initially, tokens cannot be sent until a flow control token has been received. Thereafter, tokens are allowed to be transmitted so long as there is room in the input buffer of the receiving link to receive the data.

- **Packet acknowledge**

Each message packet received over a link is acknowledged by an acknowledge packet.

- **Command handshake**

Each command message received is acknowledged by a handshake message before the controlling process can send another command message to the same device.

The exceptions to this are *Reset* and *RecoverError* command messages which can be sent in violation of the normal protocol. The *Reset* command can be sent before a handshake message is acknowledged. *RecoverError* may be sent to any node at any time to allow the controlling process to handle error conditions in the network.

The strict exchange of a handshake for every message ensures that deadlock cannot occur at this level even if an error message is sent concurrently with a command message, without demanding any parallelism in the controlling process.

8.1.2 Autonomous operation

The IMS T9000 can operate without a control link network. This is referred to as *stand alone mode*. The IMS T9000 must be set to stand alone mode if it is not connected to a control link network. Alternatively, the IMS T9000 may be set to stand alone mode if errors are to be handled independently of a connected control link network.

There are two independent flags which determine autonomous behavior.

- The stand alone mode bit in the **ModeStatus** configuration register.
This can be set either by a configuration access by the controlling process or by the CPU (possibly from the boot code). When the stand alone mode bit is set errors are handled in a distinct way. When an error is detected the control system performs an auto-reset of the chip.
- The **StartFromROM** pin.
Setting the **StartFromROM** pin causes the initial boot code to be loaded from ROM, as opposed to loading code down **CLink0**. The **StartFromROM** pin is only effective after a hard reset (reset 0, see section 8.7).

In stand alone mode software running on the IMS T9000 can perform many of the functions otherwise performed by the control links.

8.2 Control system interconnections

The control system interfaces to other subsystems of the IMS T9000.

A number of subsystems of the IMS T9000 are controlled through a separate address space, referred to as the configuration space. The control system is connected via the configuration bus to registers in the configuration space. The configuration space is accessed either by the CPU using *ldconf* and *stconf* instructions, or by the control system using *CPeek* and *CPoke* command messages received along **CLink0**. The control system has control of the internal configuration bus whenever the CPU is not running.

The control system handles memory accesses via the scheduler using peek, poke commands. Figure 8.3 shows how the control system interacts with some of the other sub-systems of the IMS T9000.

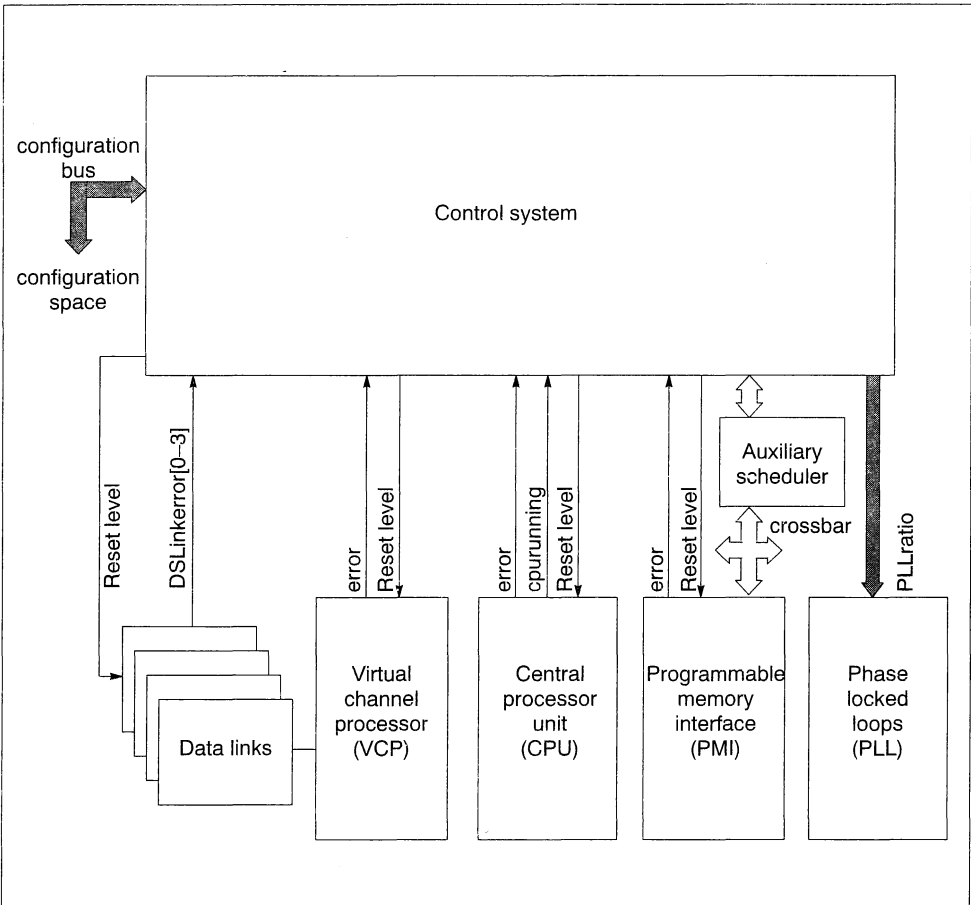


Figure 8.3 IMS T9000 control system connectivity

8.3 Control system functional description

The control system functional block diagram is shown in figure 8.4.

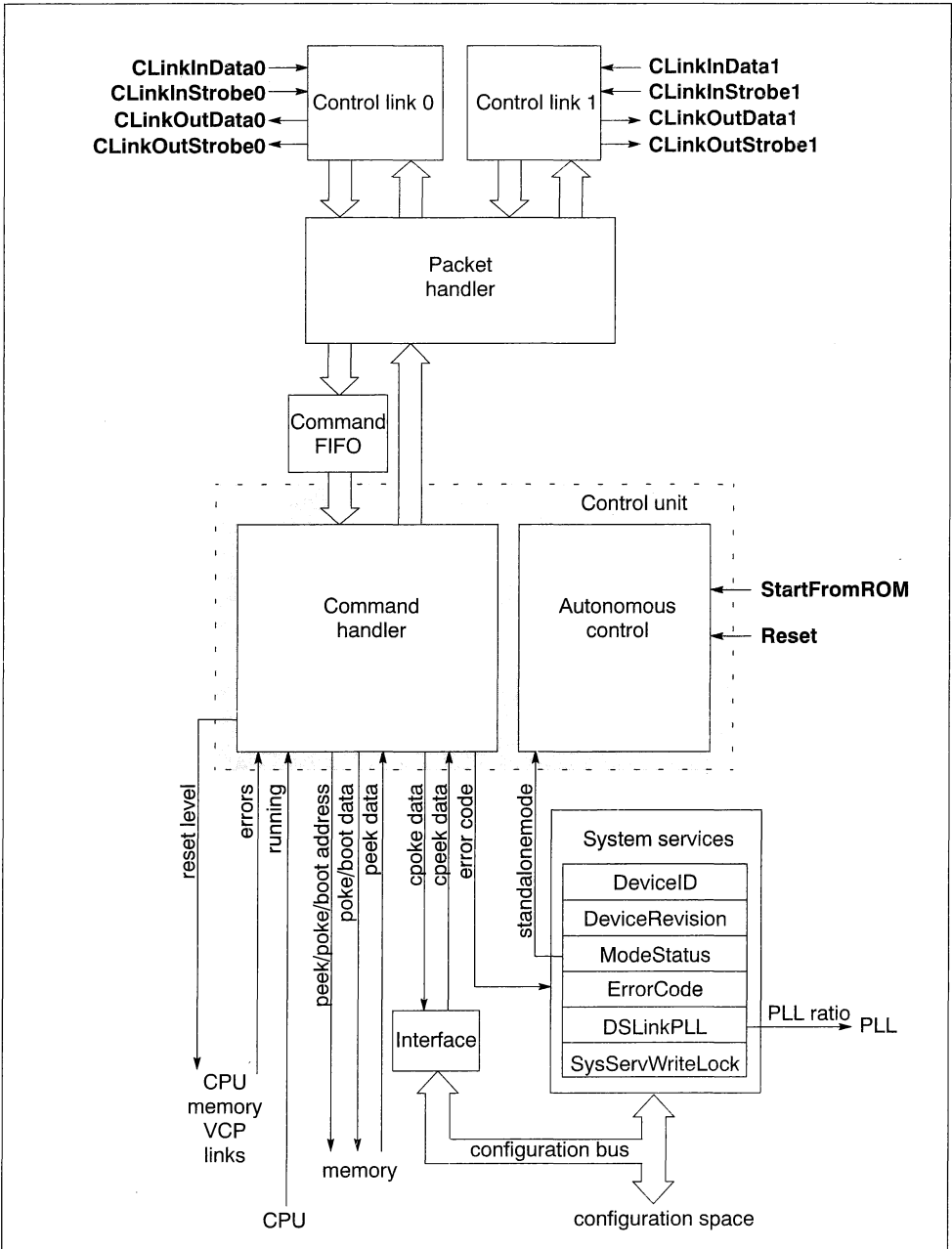


Figure 8.4 Control system functional block diagram

The control system on the IMS T9000 consists of: a pair of control links; a packet handler; a control unit and system services. A command packet enters the IMS T9000 via the control link and is forwarded to the packet handler which handles the initial processing of the packet stream. It is then passed to the command handler, via the command FIFO, which responds to the command and signifies receipt of the command to the controlling processor by returning a command handshake. System services is a subset of the configuration registers. The set of configuration registers which make up system services contain control information.

The functionality within each unit of the control system is described in more detail below.

8.3.1 Control links

The IMS T9000 has two bidirectional control links; **CLink0** and **CLink1**. They use the same electrical and packet level protocols as the four data links (see Data/Strobe links chapter 13). All communications with the controlling processor are via **CLink0**. **CLink1** is provided to allow IMS T9000 product family components to be connected in a daisy-chain.

The IMS T9000 control system includes a simple through-routing function and the controlling processor (possibly another IMS T9000) appears to have a direct connection to every device in the system. The control link traffic consists of messages and acknowledgements, whose headers describe which device the packet is for. **CLink0** receives commands from the controlling process and sends back status information.

When the network is initialized the first communication with each device programs the label and return addresses to establish the virtual channels between the control process and that device. The first two bytes of the first packet received on **CLink0** are recorded as the 'label' of the device. Thereafter the device accepts only packets with the same header; all others are routed out of **CLink1** (or destroyed, causing a protocol error, if **CLink1** is inactive). Note that this label can only be set after hard reset (i.e. assertion of the **Reset** pin). The return header address is programmed by a command message called *Start*, which includes a two byte return header. This header is prepended to all packets sent by the device. The return header can be changed by sending subsequent *Start* command messages. If the first packet received on **CLink0** is not a *Start* command message, this is an error, which will be reported as soon as a return header is programmed by a *Start* command message.

CLink0 is started automatically on receipt of the first token. **CLink1** must be started by setting the **Start-Link** bit of the **CLink1** command register (**CLink1Command**) using a *CPoke* command.

Figure 8.5 shows an example of daisy-chaining the control links in a network. The header values given in this example are fictitious and do not attempt to show how a network should be labelled. The *Start* command sent from T9000₀ to T9000₁ has header value 30, which becomes the label of T9000₁, and carries the return header value 11. The subsequent *Start* command arriving on **CLink0** of T9000₁ has header value 35 and is forwarded down **CLink1** to T9000₂. It carries return header value 12. The first packet (*Start* command) received by T9000₃ has header value 40 and carries return header value 13.

The link module hardware in each control link is identical to that in each data D/S link. Associated with each control link is a set of configuration registers, which are equivalent to those described in section 13.6 in the Data/Strobe links chapter.

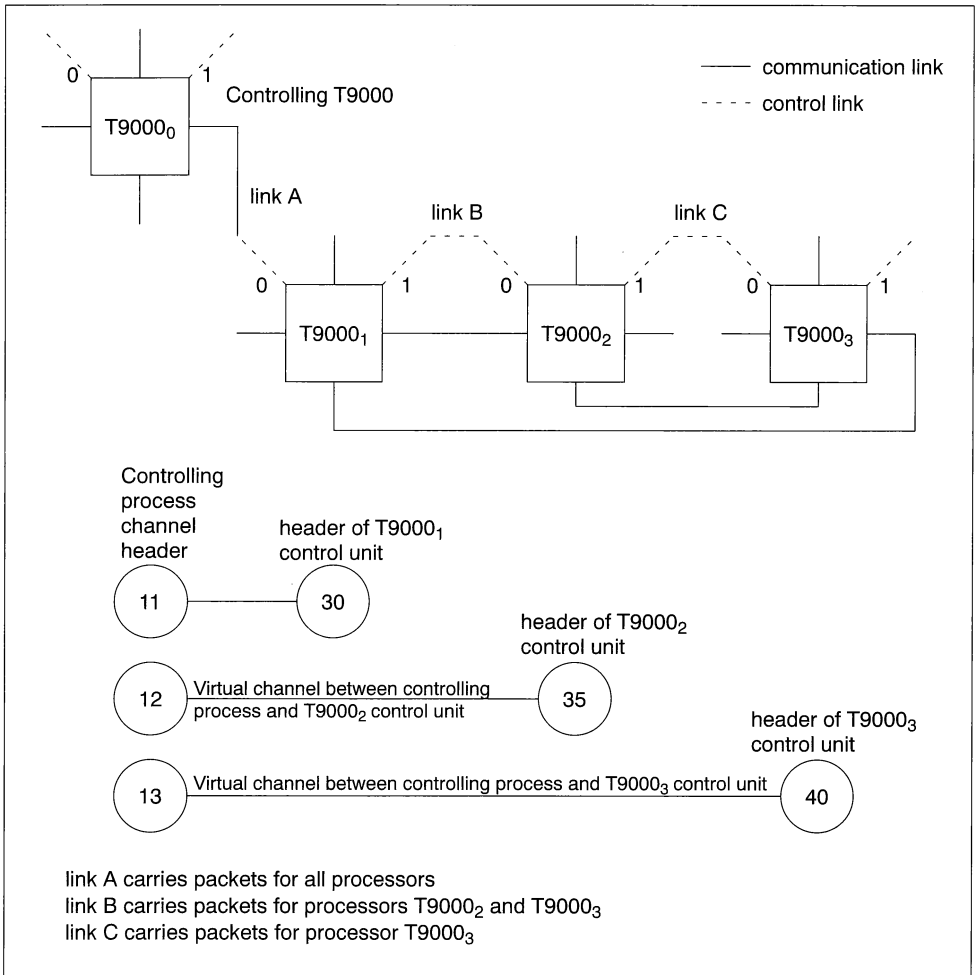


Figure 8.5 Daisy-chain operation of control messages over the control links

8.3.2 Packet handler

The packet handler handles the initial front end processing of the packet stream and performs the following tasks.

- Records the first header seen on **CLink0** after power up.
- Checks headers on incoming **CLink0** packets to determine whether they are for this device. If the packets are not for this device they are forwarded down **CLink1**.
- Programs the return header to establish the virtual channels between the controlling process and this device, by means of a *Start* command.
- Adds the return header to outgoing **CLink0** packets.
- Forwards incoming **CLink1** packets onto **CLink0**.

- Detects and handles acknowledge packets incoming on **CLink0**.
- Validates command length and termination type.
- Detects asynchronous commands *Reset*, *RecoverError* and *ErrorHandshake*.
- Rejects commands if another command is already in progress.

Note that only one command at a time can be in progress. If another command arrives it is discarded, and an error will be signalled to the control unit, which handles error transmission. The exceptions to this are the *Reset*, *RecoverError* and *ErrorHandshake* command messages which can be sent in violation of the normal protocol.

8.3.3 Control unit

The control unit consists of:

- a command handler which responds to commands and generates handshakes and error messages;
- an autonomous control block which controls the behavior of the T9000 when it is operating independently of a control network.

Command handler

The command handler generates command responses and handles errors in the system. It performs the following functions.

- Captures errors from error inputs, encodes them and forwards them to the controlling process via **CLink0**.
- Responds to errors with appropriate stop/halt to sub-systems.
- Arbitrates between command responses and errors, and forwards them on **CLink0** to the controlling process.
- Filters illegal commands as errors. (Illegal commands are commands with invalid codes.)
- Filters inappropriate commands received from the packet handler as errors.
- Responds to *Reset* commands with appropriate stop/halt to sub-systems.
- Responds to *Peek*, *Poke* and *Boot* commands and accesses the memory system.
- Responds to *CPeek*, *CPoke* commands and accesses the configuration bus.

Autonomous control

The autonomous control block handles errors when the IMS T9000 is operating independently of the control link network. If an error occurs the control unit performs an auto-reset and boot from ROM.

8.3.4 System services

The subsystems of the IMS T9000 are controlled by a set of configuration registers in the configuration space (see chapter 15). System services consists of a block of 8 configuration registers. This block of system services configuration registers contain control information and general information which is not integral to any of the other functional subsystems on the IMS T9000. For example, it is an area in which the device identification code resides, enabling either the control unit or the CPU to identify the device.

The functionality to be controlled by the system services configuration registers, and the associated bit fields are described below.

Note, all undefined bits of a configuration register must always be written with 0's.

DeviceID

The **DeviceID** register contains a 16 bit device identification code unique to the device. The device identification code can also be read using the *Identify* command. It can also be returned by instructions *lddevld* (load device identity) and *ldprodid* (load product identity). This register is read only.

DeviceID		#1001	Read only
Bit	Bit field	Function	
15:0	DeviceID	Device identification code.	
31:16		Undefined	

Table 8.1 Bit fields in the **DeviceID** register

DeviceRevision

The **DeviceRevision** register contains the revision of the device. It is a 16 bit read-only register.

DeviceRevision		#1002	Read only
Bit	Bit field	Function	
15:0	DeviceRev	Device revision.	
31:16		Undefined	

Table 8.2 Bit fields in the **DeviceRevision** register

ModeStatus

An IMS T9000 can be set to operate in stand alone mode by setting the **StandAloneMode** bit in the **ModeStatus** configuration register. In stand alone mode the IMS T9000 transputer is booted from ROM.

When in stand alone mode if an unmasked/untrapped error occurs, the control unit resets all the subsystems of the IMS T9000 and then causes the processor to boot from ROM. The **ErrorSinceReset** flag in the **ModeStatus** register is set to indicate to the ROM code that an error has occurred.

ModeStatus		#1003	Read/Write
Bit	Bit field	Function	
0	ErrorSinceReset	When set to 1 signifies to ROM code that an error has occurred since power-on reset.	
1	StandAloneMode	When set to 1 sets the IMS T9000 in stand alone mode.	
31:2		Undefined	

Table 8.3 Bit fields in the **ModeStatus** register

ErrorCode

The **ErrorCode** register is an 8 bit register used for debugging after a crash. This register is read only.

ErrorCode		#1004	Read only
Bit	Bit field	Function	
7:0	ErrorCode	Contains an error code which can be used for debugging after a crash. Refer to table 8.9, page 121 for the error code definitions.	
31:8		Undefined	

Table 8.4 Bit fields in the **ErrorCode** register

DSLlinkPLL

The **DSLlinkPLL** register contains the **SpeedMultiply** bit field and is used to program the DS-Link speeds. This takes the 10 MHz clock and multiplies it by a programmable value to provide the root clock for all the DS-Links. Refer to section 13.4 in the Data/Strobe links chapter for further details.

DSLlinkPLL		#1005	Read/Write
Bit	Bit field	Function	
5:0	SpeedMultiply	Sets link master clock to required value (see Data/Strobe links chapter).	
31:6		Undefined	

Table 8.5 Bit fields in the **DSLlinkPLL** register

SysServWriteLock

The **SysServWriteLock** register protects the system services subsystem from writes by the CPU. The system services registers require protection so that the control system is guaranteed to function regardless of program behavior.

Reset clears this register.

SysServWriteLock		#1006	Read/Write
Bit	Bit field	Function	
0	SysServWriteLock	When set to 1 it inhibits modification of the system services registers by the CPU.	

Table 8.6 Bit fields in the **SysServWriteLock** register

8.4 Control commands

The following section details the command messages which can be sent from a controlling process to the IMS T9000.

Each command is handshaken by the IMS T9000. The handshake message can contain the result of a *Peek*, *CPeek* or *Identify* command, or it may be simply a handshake code corresponding to the command message. Command response codes are the same as the command codes except with the top bit inverted. Many handshake messages include a status byte which indicates whether the received command was valid as defined below.

- Status byte has value **0** if command is valid.
- Status byte has value **1** if command is invalid or has failed for some reason.

Note that the following definitions are of the *messages* as sent and received by the controlling process. At the packet level there are headers and End of Message (EOM) indicators and also acknowledge packets which are not described here. They are however shown in figures 8.6 and 8.7 which show the command packets received by the IMS T9000 and the handshake packets returned to the controlling process respectively.

Start

This command programs or re-programs the return header of the IMS T9000. The return header is 2 bytes long, with byte 0 being the first byte transmitted following the command code. Note that if this command is used to re-program the return header, the acknowledge for the command message packet will be sent with the *old* header, whilst the handshake will be sent with the *new* header.

The *Start* command must be the first command received following power on reset. If an error occurs before the first *Start* command is received, the start handshake will be returned before the error message is sent.

Reset

The *Reset* command message causes some or all of the subsystems of the IMS T9000 to be reset. The level of reset is encoded in the 'level' byte of the command message. There are three different levels of reset to which the IMS T9000 responds. Reset 1 is equivalent to a hard reset except that the control system is not affected; reset 2 resets all subsystems of the IMS T9000 except the control system, and leaves the configuration and the PMI activity unchanged; reset 3 simply halts the processor. See section 8.7 for more information on reset levels. A *Reset* command with an invalid level is handshaken with a failure status.

Note that a *Reset* command may cause a handshake for a previously transmitted command to be: terminated prematurely (with an end-of-message token); completed with a failure status; or suppressed entirely.

Note that any level of reset may abort the command which was executing when the *Reset* command was applied. An illegal level of *Reset* will also result in a handshake with a failure status being returned.

Identify

The *Identify* command message causes the IMS T9000 to respond with a handshake containing a unique identifier. This can be used to check the contents of a network. The lower 16 bits of the identifier are the same as the contents of the **DeviceID** register (see section 8.3.4); the upper 16 bits are zero. Note, the *Identify* command code is identical for all IMS T9000 family devices.

Stop

The *Stop* command message stops the processor 'cleanly' so that register values are preserved for debugging. It acts like the **Analyse** pin on the T8 transputer. Refer to section 8.5.4 for further information on the use of this command.

RecoverError

This command is used in error recovery on the control links (see section 8.5.3). It restores the protocol after a link error in the control link system. Note that if there is an unhandshaken error, the *RecoverError* handshake will be returned before the error message is sent.

CPeek

The *CPeek* command includes a 2 byte address which points to a register in the configuration address space. The handshake message returns the value stored at the given address. If the address is invalid the handshake message returns an invalid status.

CPoke

The *CPoke* command includes a 2 byte address and 4 bytes of data. It writes the data to the configuration space register at the given address. If the address contained in the command message was invalid the status byte of the handshake message indicates failure.

Note, some registers do not have a value, but writing to them causes some action to occur.

Peek

The *Peek* command includes a 4 byte address which points to a memory location in the normal address space. The address location must be word-aligned. The handshake message returns the value of the location, unless the status byte indicates that the command failed because the address given in the peek command message was invalid.

Poke

The *Poke* command writes data to a memory location. The 4 byte address of the location (which must be word-aligned) and the value to be written (4 bytes of data) are included in the command message. If the address contained in the command message was invalid, the status byte of the handshake message indicates failure.

Boot

The *Boot* command starts a 'booting' sequence. The booting sequence makes the loading of code via the control link more efficient than performing a series of pokes. The *Boot* command message contains the length in bytes (which must be a multiple of 4) of the boot code to be loaded, and the address (which must be word-aligned) of the memory into which the boot code is to be written.

BootData

The *BootData* command loads code as part of a booting sequence. Each *BootData* command message contains 16 bytes of code. These code blocks are loaded into consecutive contiguous blocks of four words, starting from the pointer given in the *Boot* command message, until as many bytes are loaded as specified by the length field of the *Boot* command message. Any remaining bytes are discarded.

Run

The *Run* command causes the processor to start executing, with a workspace pointer (**Wptr**) (which must be word-aligned) and instruction pointer (**Iptr**) contained in the *Run* command message.

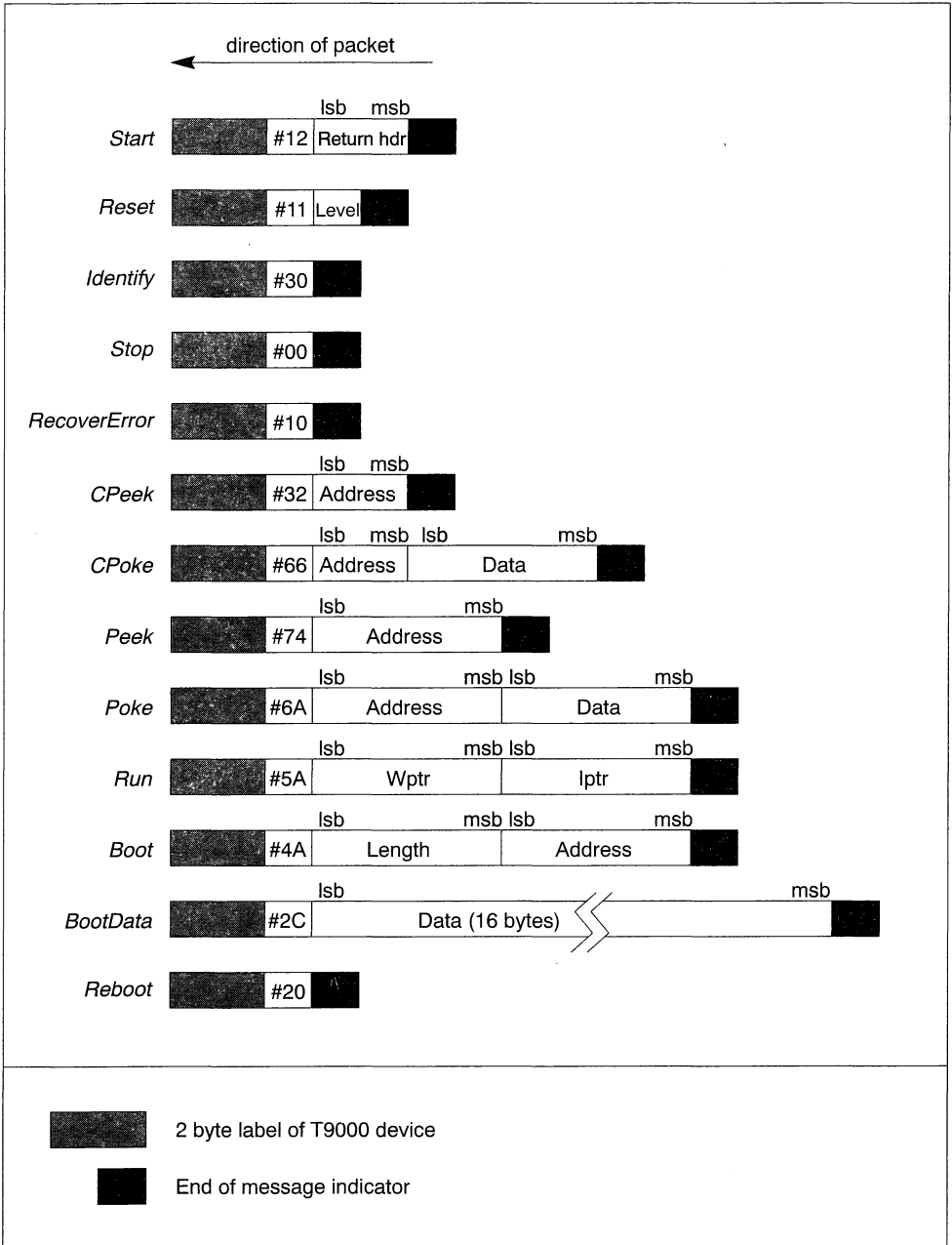
Reboot

The *Reboot* command causes the processor to reboot from ROM. It starts executing, with a workspace pointer (**Wptr**) (which must be word-aligned) and instruction pointer (**Iptr**) read from memory locations at the top of the address space.

The IMS T9000 reads a **Wptr** and an **Iptr** from two fixed locations at the top of memory:

Boot from ROM **Iptr** address is #7FFFFFF8

Boot from ROM **Wptr** address is #7FFFFFFC



Note: the ordering of the length and address bytes of the *Boot* command has changed from that stated in *The T9000 Transputer Products Overview Manual*.

Figure 8.6 Command messages from controlling process to the IMS T9000

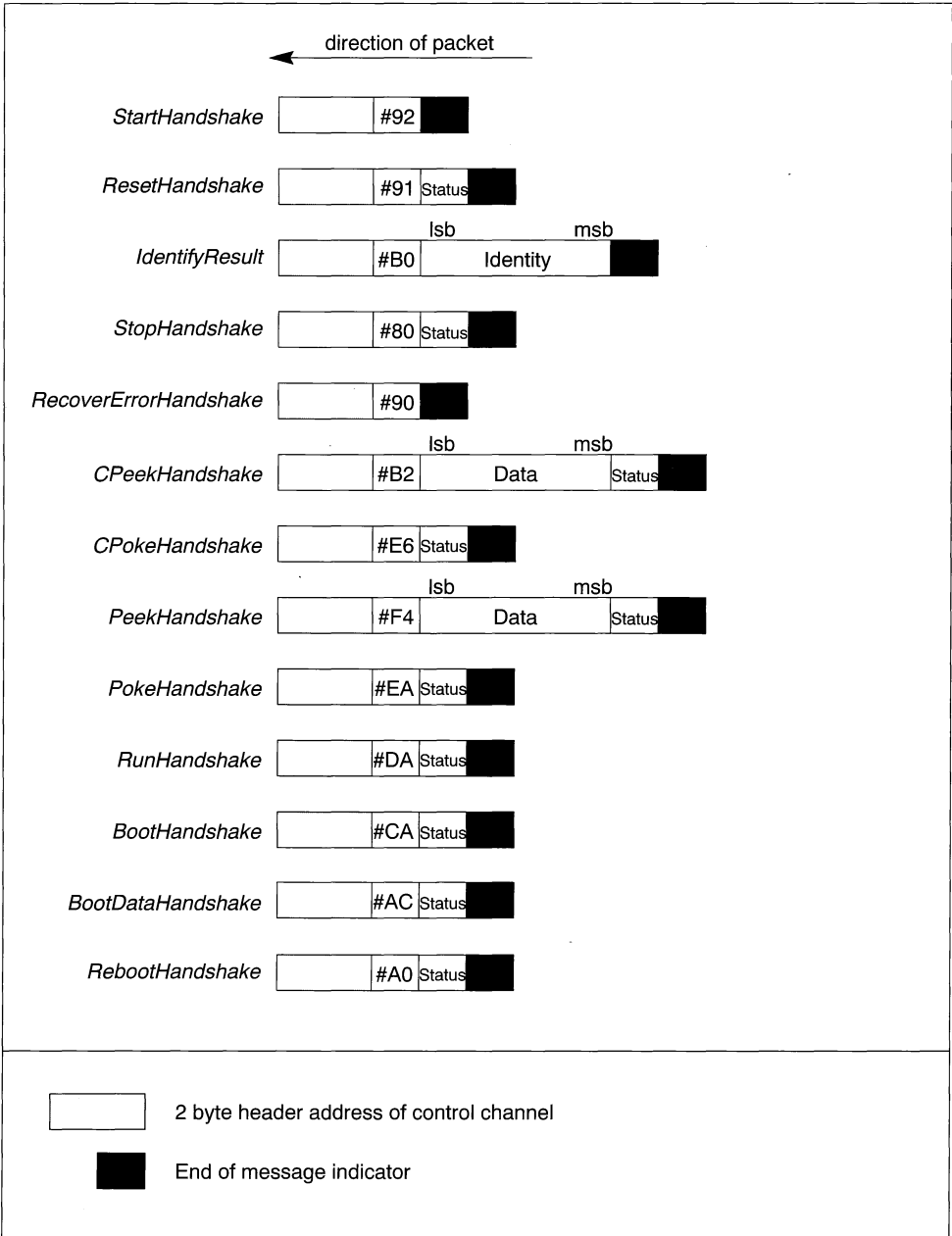


Figure 8.7 Handshake messages from the IMS T9000 to the controlling process

Error message

The IMS T9000 control system can send an *Error* message to the controlling process to indicate that an error has occurred. The *Error* message contains an error code which determines the type of error, as given in section 8.5.1. The controlling process returns an *ErrorHandshake* message.

When there is an error record outstanding an error message is sent whenever:

- all previously sent error messages have been handshaken;
- all previously sent messages, since the last hard reset or *RecoverError*, have been acknowledged.

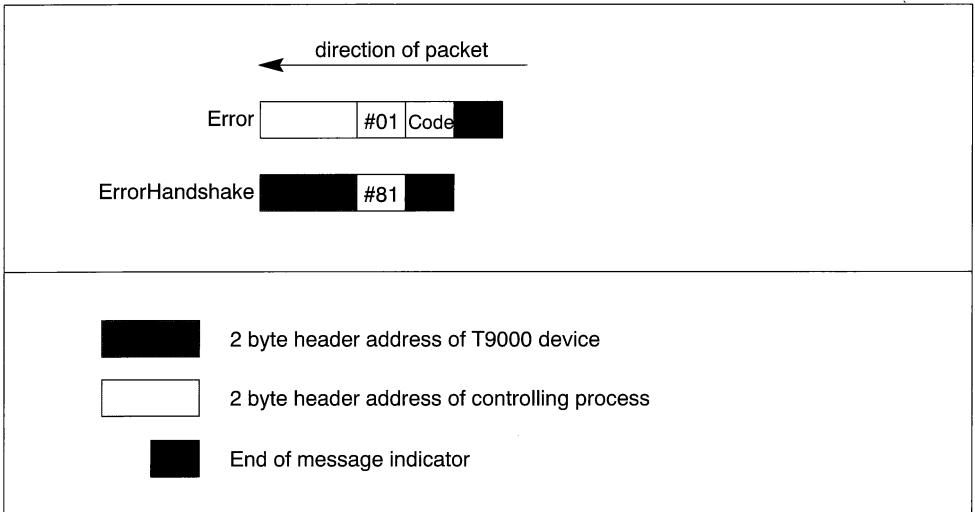


Figure 8.8 Error message

8.4.1 IMS T9000 gross state and validity of commands

The overall state of the IMS T9000 determines which commands are valid. Certain commands change this state, as does the occurrence of an error. This is detailed in the table 8.7 below. In this table, 'stand alone error' is an error which occurs when the IMS T9000 is operating in stand alone mode (i.e. the **StandAloneMode** flag in the system services **ModeStatus** register is set). This causes a reboot using an **lptr** and a **Wptr** read from the top of memory, and so returns to the Running state via a transitory state which is not listed here. 'Last boot data' indicates a *BootData* command when the remaining number of bytes to complete the boot sequence is between 1 and 16 inclusive. This table does not explicitly deal with booting from ROM, which causes a 'doubling' of some states since Running and Error can both occur without a *Start* having been received.

If the command is not valid for the state of the IMS T9000 then the command will be handshaken with a failure status.

State	Valid commands	Commands or events which change the state	Resulting state
Initial	<i>Start</i>	<i>Start</i>	Ready
Ready	<i>Start, Identify, RecoverError, Reset, CPeek, CPoke, Peek, Poke, Boot, Reboot, Run</i>	<i>Boot</i> <i>Reboot</i> chip error <i>Run</i> stand alone error	Booting Running Error Running Running
Booting	<i>Start, Identify, RecoverError, Reset, CPeek, CPoke, Peek, Poke, BootData, Reboot</i>	<i>Reset</i> <i>Reboot</i> last boot data chip error stand alone error	Ready Running Ready Error Running
Running	<i>Start, Identify, RecoverError, Reset, Stop</i>	<i>Reset</i> <i>Stop</i> chip error	Ready Stopping Error
Stopping	<i>Start, Identify, RecoverError, Reset, Stop</i>	<i>Reset</i> chip error stand alone error	Ready Error Running
Error	<i>Start, Identify, RecoverError, Reset</i>	<i>Reset</i>	Ready

Table 8.7 Gross state and validity of commands

Note that, once a *Run* command has been received, further *Peek, Poke, CPeek, CPoke, Boot, BootData* and *Run* commands are invalid until the processor is halted by either a *Reset* command or a system error (see section 8.5).

8.5 Errors

The IMS T9000 control system handles three distinct classes of errors, as listed below.

- 1 Errors on the control links, which include:
 - parity/disconnect on **CLink1** (down link);
 - unexpected acknowledge;
 - message of the wrong length or invalid command code;
 - handshake protocol error, when a command is received (other than *Reset* or *RecoverError* commands) before the handshake for the previous command has been received.
- 2 IMS T9000 system errors – errors from one of the subsystems when stand alone mode is not set.
- 3 Stand alone mode errors – subsystem error when stand alone mode is set.

The error may result in stopping the CPU and/or sending an error message to the host via **CLink0**, see table 8.8 below.

Error class	Result of error		
	Stops the CPU	ErrorSinceReset flag set	Error message sent on CLink0
Control link error	No	No	Yes
System error	Yes	Yes	Yes
Stand alone mode error	Yes	Yes	No

Table 8.8 Error effect

8.5.1 Recording of Errors

The IMS T9000 control system can send *Error* messages to the controlling process to indicate that an error has occurred. The *Error* message contains an error code which determines the type of error, as given in table 8.9 below. The control unit can record the occurrence of *one* error corresponding to each error value.

Code	Error type	Priority
#C0	Control link 0 command error	Highest
#C1	Control link 0 protocol error	
#C2	Control link 1 parity or disconnect error	
#80	Link 0 error	
#81	Link 1 error	
#82	Link 2 error	
#83	Link 3 error	
#01	CPU error	
#02	VCP error	
#03	PMI error	

Table 8.9 Error codes

The receipt of an error handshake causes the record of the last sent error to be removed.

The system services **ErrorCode** register always contains the value of the most recent error to occur.

A hard reset, reset 1 or reset 2 causes the record of untransmitted errors to be cleared.

8.5.2 Stand alone mode errors

When an IMS T9000 is set to operate in stand alone mode, errors are handled in a distinct way. If an unmasked/untrapped error occurs, the control system resets all the subsystems of the IMS T9000 and then causes the processor to boot from ROM. The **ErrorSinceReset** flag in the **ModeStatus** register is set to indicate to the ROM code that an error has occurred.

The **ErrorSinceReset** flag is set by any unmasked/untrapped error and is set independent of the **StandAloneMode** flag.

The **ErrorSinceReset** flag can be accessed by the *testpranal* instruction. This flag is cleared by a hard reset, a reset to level 1 or by a configuration write to the register.

8.5.3 Errors on control links

The control link network is assumed to be designed and connected by the user to achieve very high reliability. The control links should be operated at a low enough speed to ensure this.

If a parity or disconnect error occurs on **CLink1** then an error message is sent to the controlling process along **CLink0**. If a parity or disconnect error occurs on **CLink0** then an error message cannot be sent to the controlling process. However, the output of **CLink0** is halted, and this will be detected by the adjacent device, which will report the error to the controlling process. In this manner all errors on the control link system are reported to the controlling process.

The *RecoverError* command allows the control link to recover from an error in the control links, enabling the control network to avoid deadlock if for example an acknowledge packet gets corrupted. The *RecoverError* mechanism restarts the communication between nodes after there has been an error in the control network. To achieve this, the *RecoverError* command is permitted to be sent in violation of the normal protocol.

A link error may, or may not, cause one or more packets to be lost, thus the state of the remote end of the virtual link (the end which is implemented by the control system hardware) becomes uncertain. In this instance the *RecoverError* command can be used to reset the state of the remote end of the virtual link causing any un-handshaken error message (which may have been lost) to be re-sent.

Note, the *RecoverError* command may result in non-deterministic behavior if the error message was in fact not destroyed. If the error message was not destroyed the following can occur.

- The *Error* message will arrive at the controlling process concurrently with the controlling process's sending of the *RecoverError* command.
- The controlling process will perform an input, expecting a handshake to the *RecoverError*, and will receive the *Error* message instead, concurrently with the IMS T9000 receiving the *RecoverError* command.
- This will cause an acknowledge packet to be sent to the IMS T9000, concurrently with the IMS T9000 transmitting a *RecoverError* handshake.
- The IMS T9000 will receive the acknowledge, assume that this is for the handshake, and proceed to re-transmit the error.
- It is now non-deterministic whether the next input by the controlling process will receive the handshake message or the error message (if this arrives and overwrites it first).

The final non-determinism is the consequence of the original non-deterministic loss of packets. However the same thing could happen even in the absence of link errors if the *Reset* command also reset the state of the remote end of the virtual link, so for this reason it does not.

8.5.4 Post-mortem debugging of IMS T9000 systems

The following section details debugging of an IMS T9000 system following an error.

An IMS T9000 system is brought to a quiescent state for analysis by sending the *Stop* command to each IMS T9000.

The *Stop* command stops the processor cleanly so that register values are preserved and debugging can be performed. The CPU continues execution until the process deschedules or reaches a timeslicing point. The VCP deactivates all channels queued for output or addressed by incoming packets. The PMI and links are not affected. The timers continue to operate until a reset is received, but no processes are placed on the run queue after a *Stop* has been received. Sending *Stop* to a processor which is already stopped or which has had an error has no effect.

Following the *Stop* command, the controlling process must send a *Reset3* command to each IMS T9000 which halts the CPU. A delay of at least 64 ms from the receipt of the *StopHandshake* must be allowed for the system to become quiescent, before the *Reset3* command is sent.

Once the IMS T9000s have been reset the system is guaranteed to be static. The configuration can then be checked using *CPeek* commands and if necessary 'repaired' using *CPoke* commands. *Peek* commands can be used to save the contents of memory locations to be used by a debugging kernel. A kernel can be loaded using a *BootData* sequence. A *Run* command can then be sent with the appropriate **lptr** and **Wptr** for the debugger process. The debugger process is run as a high priority process with the old register values as its parameters.

Each debugger process should perform the following.

- Swap the run and timer queues and save the shadow registers.
- Reset the VCP.
- Re-initialize the system virtual links.
- Inspect the virtual channels and memory to determine what the problem was.

System analysis can then be carried out in the usual way, without further reference to the control network.

State delivered to the boot program

When the processor is rebooted, either by a control link *Run* or *Reboot* command or by rebooting from memory in stand alone mode, the previous state of the processor is written into memory at word offsets relative to the new **Wptr** as follows:

Offset from Wptr	Value	Description
0	WdescReg	process descriptor of the currently executing process
1	lptrReg	pointer to next instruction to be executed
2	StatusReg	status and control information for the current process
3	ThReg	pointer to the current trap handler data structure
4	ReasonReg	trap reason code, see <i>Instruction Set Manual</i> table 10.5
5	error type	type of error, see <i>Instruction Set Manual</i> table 10.10
6	EptrReg	pointer to instruction that caused the error

8.6 Configuration

The IMS T9000 transputer has several sub-systems which need to be initialized on bootstrap. This initialization is performed via the configuration bus.

The IMS T9000 transputer can be bootstrapped from a ROM device, such as an EPROM or a Flash EPROM. Refer to the Programmable Memory Interface chapter 10 for details on booting from ROM. The IMS T9000 transputer can also be setup and bootstrapped using the control links and the four high speed data links, enabling the transputer to be used without any external memory or external I/O.

The setting of the **StartFromROM** pin determines whether the IMS T9000 transputer is bootstrapped from a ROM device or down control link (**CLink0**) from a controlling processor. When the **StartFromROM** pin is held low the IMS T9000 will be bootstrapped down **CLink0**, when set high the IMS T9000 will be bootstrapped from ROM.

For a system of IMS T9000's, the controlling IMS T9000 can be booted from ROM and usually the other IMS T9000s in the system will be booted down a link.

8.6.1 Booting from link

An example IMS T9000 system configuration is shown in figure 8.9. In this system, the IMS T9000 is reset by the controlling system and then initialized. The controlling system can be another IMS T9000 transputer which has been bootstrapped from ROM, or a computer with a link adaptor. The **StartFromROM** pin is held low thus the IMS T9000 will bootstrap using the control link (**CLink0**). Instructions sent from the controlling processor to the transputer via this link can be used to configure the sub-systems on the device; for example, the configuration space of the Virtual Channel Processor (VCP) needs to be initialized ready for the receipt of program code. The cache is initialized to behave as 16 Kbytes of on-chip memory. Setting up the device in this way can be accomplished using the relevant software tools.

This system design is very small and is useful for the design of transputer networks in which each node requires less than 16 Kbytes of memory, and therefore no external memory is required.

The controlling processor can configure systems using *CPoke* commands, and can load code and data into the IMS T9000's memory using *Poke* and *BootData* commands.

Once the transputer has been setup, the program data can be passed to the device using the data links (**Link0-3**), and once loaded, the application can be executed. Errors are reported to the controlling processor via **CLink0**, which can also be used to debug the system.

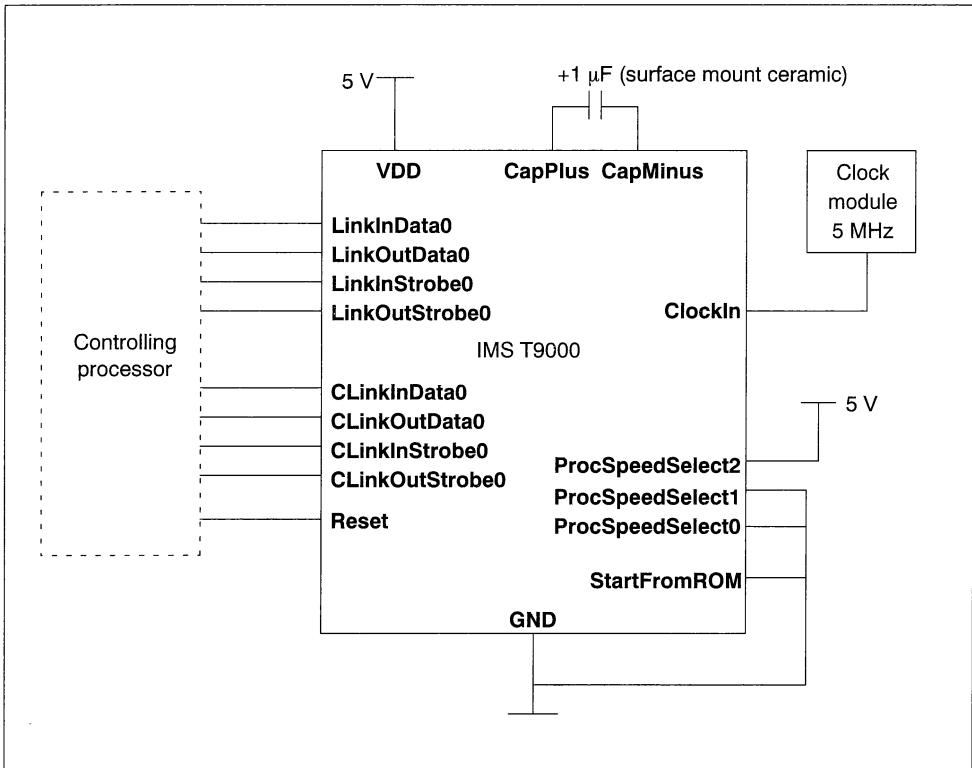


Figure 8.9 Basic IMS T9000 system booting from link

8.6.2 Boot from ROM then link

An IMS T9000 may be booted from ROM, in order to perform local configuration and bootstrapping, and then hand control to a controlling process connected via **CLink0**.

This is achieved if the ROM code terminates by setting error, which (in the absence of a trap handler) causes the IMS T9000 to halt, and an *Error* message to be transmitted from **CLink0**. Receipt of this message informs the controlling process that it has mastership of the IMS T9000's memory and configuration bus. Note that the ROM code can communicate data to the controlling process by storing it at a known location before setting error. The controlling process can then access this data by means of *Peek* commands.

8.7 Reset levels

The term *configuration* is used to refer to the sequence of operations required to take an IMS T9000 transputer network from its power-on state to having an application, or operating system, running. In doing so the state of the network must be taken through a sequence of defined levels or *reset levels*. These are shown in figure 8.10.

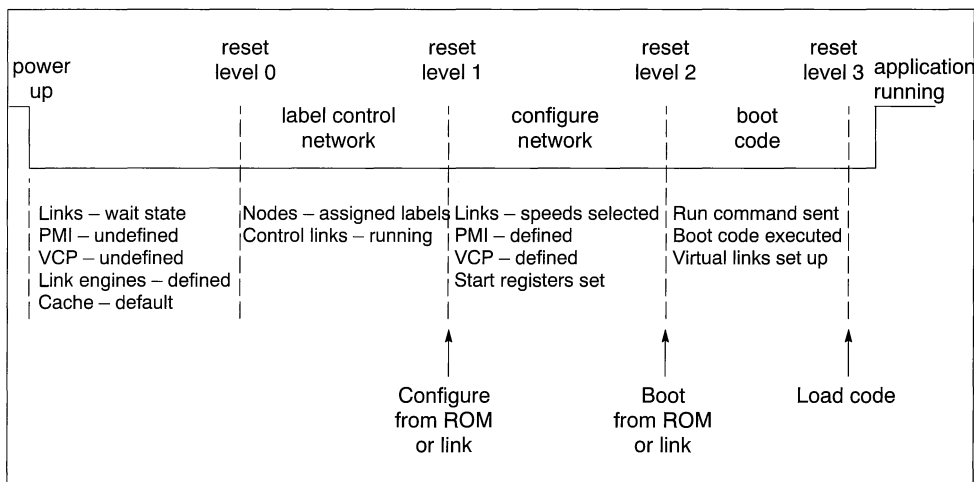


Figure 8.10 Stages of configuration and reset levels

Hardware reset, soft resets, and error responses are communicated between the control system and the various other sub-systems. These levels of reset may either be initiated via commands along **CLink0** or by an autonomous response by the control system.

8.7.1 Level 0 – hardware reset

After a hardware reset (assertion of the **Reset** pin) each IMS T9000 is in the following state.

The processor is stopped, **Wdesc** is *NotProcess.p* (refer to Appendix A for this value) and the scheduler queues are empty.

The state of the PMI and VCP is not defined, and both are inactive.

All the (data and control) links are in Wait state with a default speed of 10 MHz. The label and return headers for the control links are undefined.

The cache is initialized to act as 16 Kbytes of on-chip RAM.

The network can be returned to level 0 by taking all the **Reset** pins in the network high, it cannot be achieved by software. At this point the **StartFromROM** pin is sampled.

8.7.2 Level 1 – labelled control network

The labelling phase moves from level 0 to level 1. In it the label and return headers are set by a *Start* command message being received on **CLink0**. Level 1 for the network has all label and return headers configured and all connected control links operational.

In a small system, such as a single IMS T9000 operating in stand alone mode (refer to section 8.5.2), the label and return headers remain undefined. Any error occurring which would normally output an error message on **CLink0** will result in the PMI fifth bank being re-enabled and the ROM code being restarted. Level 1 in this case is considered to have the label and return headers configured as undefined.

The network can be reset to level 1 by sending a *Reset1* command message to each IMS T9000. After this reset message the label and return headers are still valid. All other registers in the configuration space are reset to their level 0 values.

8.7.3 Level 2 – configured network

The configuration phase moves from level 1 to level 2. The state (resident in the configuration space) required to make all subsystems of the IMS T9000 operational, is programmed. If the **StartFromROM** pin was sampled high at the end of the hardware reset then a process will be executed from ROM. This will use the *stconf* instruction to program the configuration space registers. If the **StartFromROM** pin was sampled low then the configuration space will be programmed by *CPoke* command messages received down **CLink0** or by *stconf* instructions executed by code loaded and run via **CLink0**.

The network can be reset to level 2 by sending a *Reset2* command message to each IMS T9000. At this level of reset the application program is stopped (possibly in order to reload and run another one that is configuration compatible) whilst the hardware configuration is unchanged. This level of reset leaves the PMI still active and the values in the configuration space of the PMI unaltered.

8.7.4 Level 3 – booted network

The booting phase moves from level 2 to level 3. This phase is responsible for setting up the virtual links for the network using the instructions described in the communications chapter 11. This is always performed by running code, but this code can either be executed from ROM, or be loaded down the control link using the *Boot* and *BootData* command message.

The network can be reset to level 3 by sending a *Reset3* command message to each IMS T9000. At this level of reset the handshake state is cleared.

8.7.5 Loading code

The network is now connected and code can be loaded via the communication links, or executed from ROM.

8.7.6 Levels of reset effect

The effect of different levels of reset on various aspects of the IMS T9000 state is summarized in the following table.

When the handshake/acknowledge state is cleared any outstanding handshakes/acknowledges will be ignored.

State	Reset level				Recover Error
	0	1	2	3	
DS-Links 0-3	Re-configured	Re-configured	Cleared	no effect	no effect
PMI	Re-configured	Re-configured	Cleared	no effect	no effect
VCP	Re-configured	Re-configured	Cleared	no effect	no effect
CPU	Cleared	Cleared	Cleared	Cleared	no effect
Configuration CPU locks	Cleared	Cleared	Cleared	no effect	no effect
Error state	Cleared	Cleared	Cleared	no effect	no effect
Handshake state	Cleared	Cleared	Cleared	Cleared	Cleared
Acknowledge state	no effect	no effect	no effect	no effect	Cleared
Label and return header	Cleared	no effect	no effect	no effect	no effect

Table 8.10 Effect of the different levels of reset on various aspects of the T9000 state

9 Instruction and data cache

There are two separate high speed caches on the IMS T9000, a general purpose unified (instruction and data) cache and a small workspace cache for local variables. This chapter describes the instruction and data cache on the IMS T9000 and explains how it works.

The cache provides fast multi-ported access to the most commonly used data, reducing the number of accesses to external memory. Thus, enabling lower cost, slower devices to be used as external memory without degradation of performance. The majority of external memory accesses will be cache refills (i.e. multiple word reads and writes) enabling fast access memory methods such as page mode to be used.

The IMS T9000 has a 16 Kbyte associative unified write-back (also known as copy-back) cache. That is, memory writes update the cache (if applicable) without necessarily updating memory immediately. Updating of memory occurs when the line that has been changed is discarded from the cache, thus main memory changes on a miss not on a write. This minimizes external bus accesses and reduces the latency of processor write accesses. At power-on the cache behaves as 16 Kbytes of internal memory, so that the IMS T9000 may be used with no external memory. During configuration the cache may be programmed to behave as 16 Kbytes of cache, 16 Kbytes of internal RAM, or 8 Kbytes of cache and 8 Kbytes of internal RAM.

To summarize, the cache system has the following properties:

- Bandwidth of 800 Mbytes/s from a 50 MHz T9000
- 16 Kbytes of memory arranged in four banks with 16 bytes/line
- Each cache bank is 256 way set associative
- Write-back (copy-back)
- Random replacement strategy
- Physical address tags
- Allocate cache line on a read or write miss

9.1 Cache overview

The cache is accessed, via a crossbar switch, by a number of functional units within the IMS T9000, see figure 9.1. The IMS T9000 memory system has been designed to provide fast and efficient access to data and instructions.

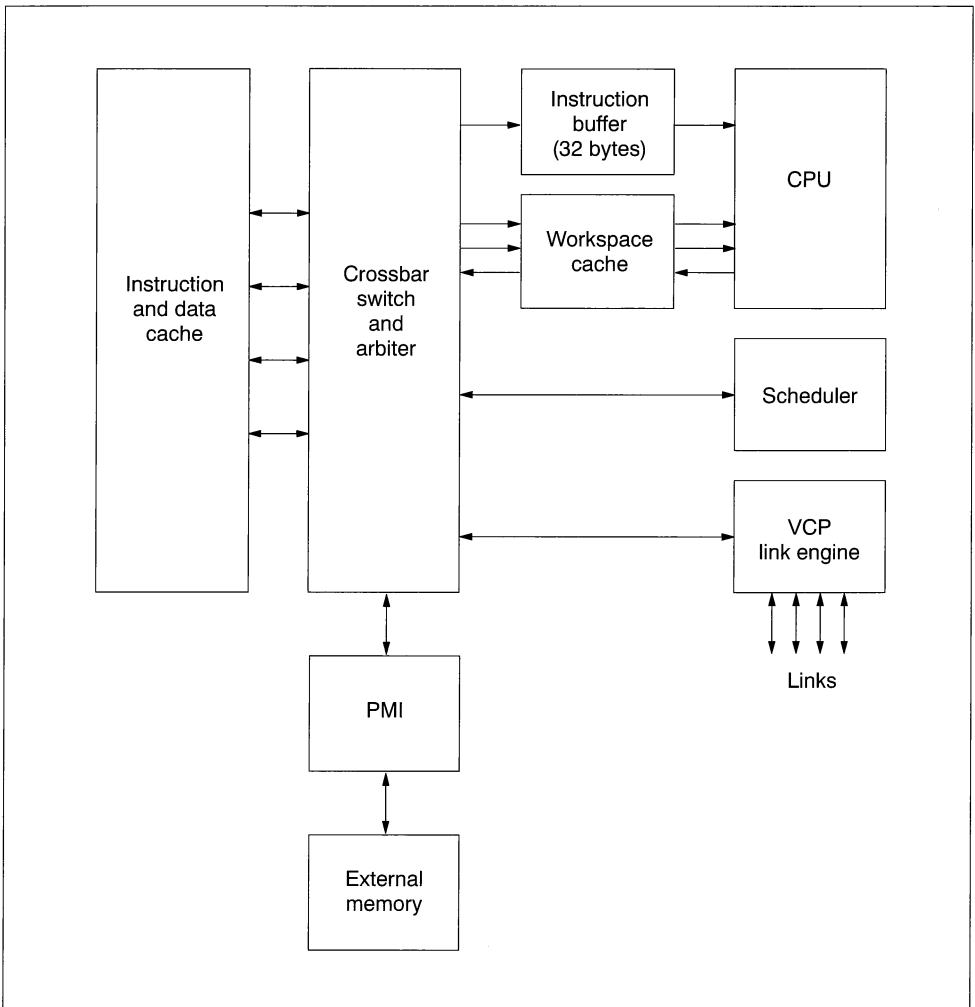


Figure 9.1 IMS T9000 memory system overview

9.1.1 Cache organization

The IMS T9000 cache system is organized as four independent 4 Kbyte, fully associative, cache banks. Each bank (not to be confused with the four banks of the IMS T9000 Programmable Memory Interface (PMI)) caches one quarter of the address space and each has its own address bus and data bus. This allows up to four separate accesses to be made in a single cycle. The directory search covers all lines in the cache bank selected. Each bank has 256 lines, with four 32 bit words per line and each line having its own fully-associative tag, see figure 9.2.

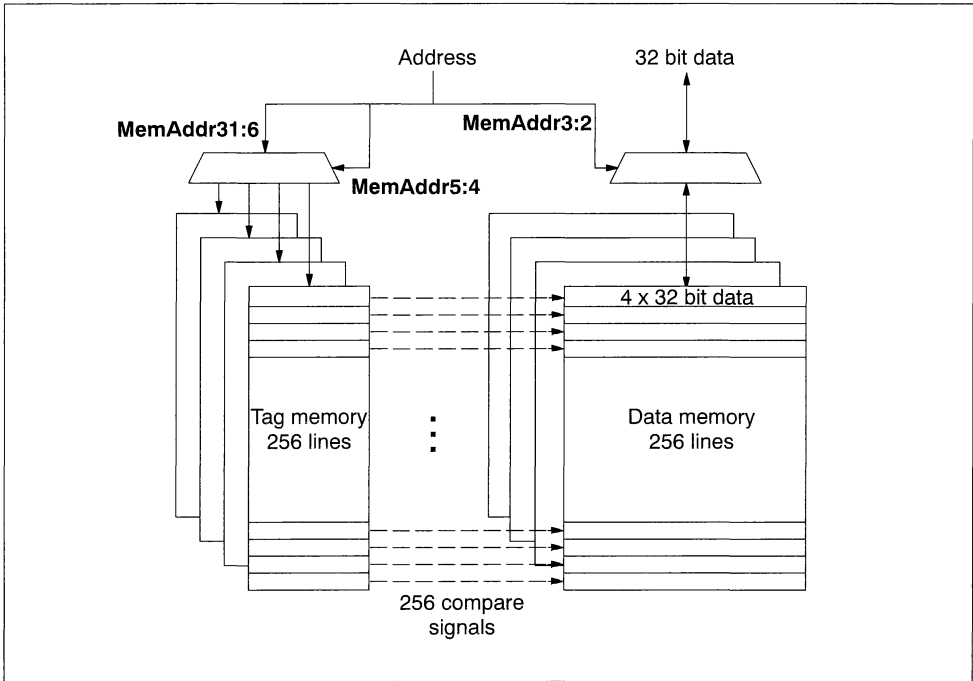


Figure 9.2 Cache organization

All internal addresses are 30 bits wide (**MemAddr2-31**) and are split into three fields, see table 9.1 below.

Address fields	Function
MemAddr31:6	Cache line tag
MemAddr5:4	Selects the cache bank 00 bank 0 01 bank 1 10 bank 2 11 bank 3
MemAddr3:2	Selects the word in the cache line 00 word 0 01 word 1 10 word 2 11 word 3

Table 9.1 Address bit functions

The tag, stored in content addressable memory (CAM), records the physical address in external memory which is being cached in the line. The tag is matched against incoming addresses. Each cache line has a tag comparator which compares the address presented with the stored tag. The cache line is selected when the address presented matches the stored tag.

The CAM also contains a valid bit which is set to 1 to indicate the line contains valid data, i.e. the address and data are meaningful. The valid bit must be set for a match to be made. Lines without their valid bit set are not included in the cache search. To write valid bits and tags the CAM is addressed using a conventional row decoder. This is controlled automatically by the cache controller and refill engine.

The data memory is stored in four banks. Each bank consists of;

- 256 lines of data.
- four 32 bit data words per line.
- a dirty bit per data word – this indicates whether a write has taken place to the word since it was loaded from main memory.
- a dirty bit per line – this indicates whether any of the four words in the line are dirty.

The dirty bits allow the refill engine to write back to external memory only those words which have been written to.

Four internal byte enables are provided to write bytes and part words.

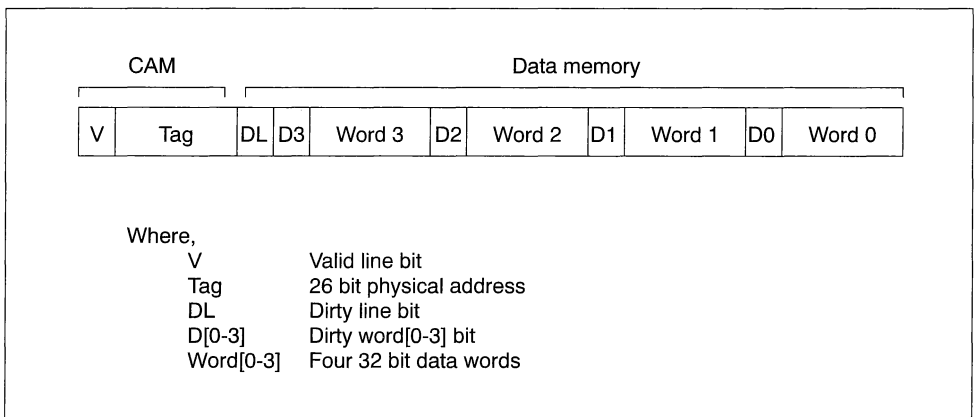


Figure 9.3 Cache line structure

9.2 Cache functional description

Figure 9.4 shows the major functional units of the cache.

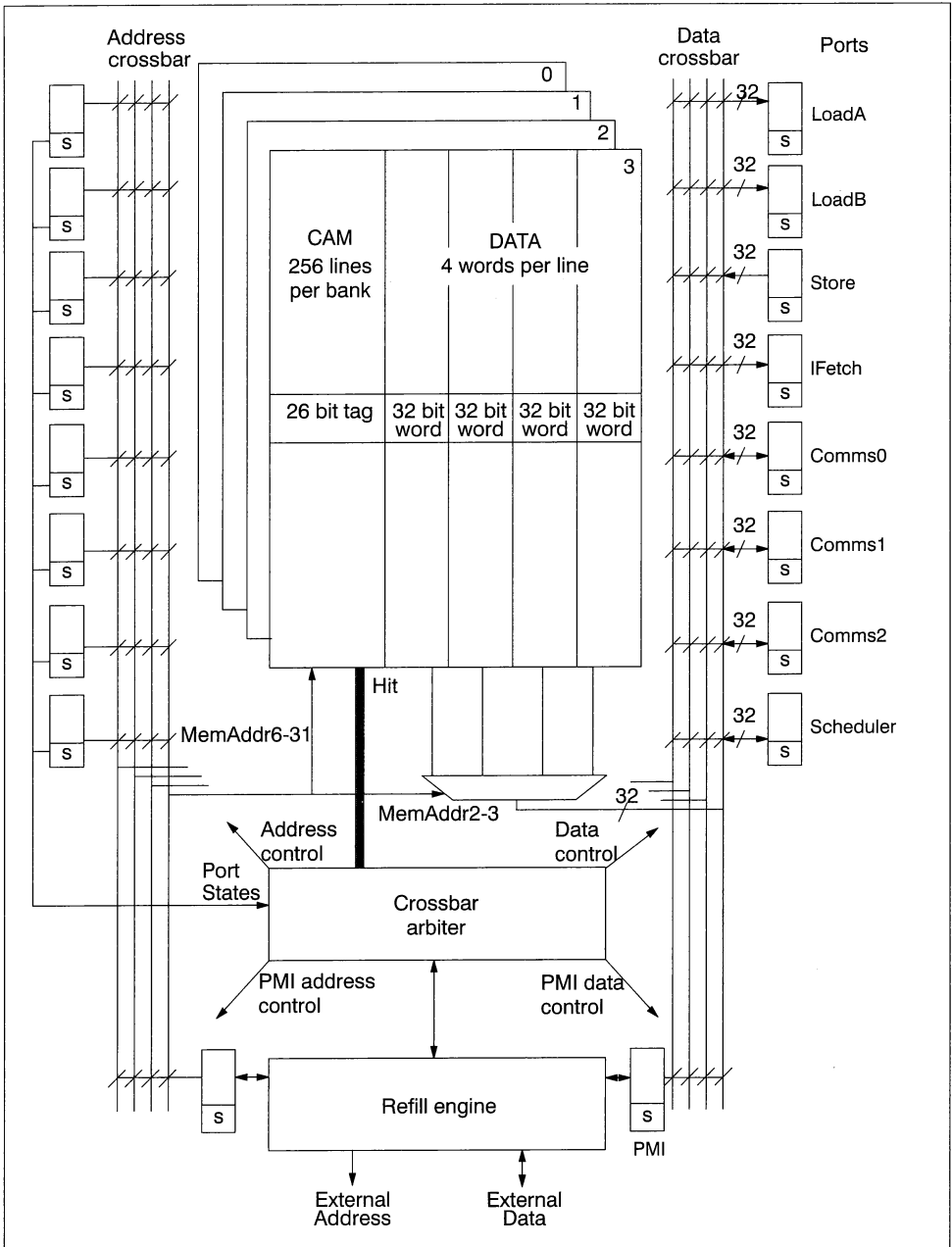


Figure 9.4 Cache functional diagram

9.2.1 Port crossbar switch and arbiter

There are four subsystems which use the cache, with a total of nine sources of address, they are:

- CPU Pipeline (4 sources)
 - CPU load A
 - CPU load B
 - CPU store
 - Instruction fetcher
- Virtual channel processor (3 sources)
 - Comms0
 - Comms1
 - Comms2
- Scheduler and control unit
- Programmable memory interface (PMI) / Refill engine

Each of the nine requesting sources has its own crossbar port which handles memory requests, and ensures the requesting source is connected to the required cache bank. The nine ports are connected to the four cache banks using a crossbar switch. Figure 9.5 is a block diagram of the crossbar switch and controller interconnections. An arbiter decides which functional unit of the IMS T9000 gains access to each of the four cache banks on each cycle.

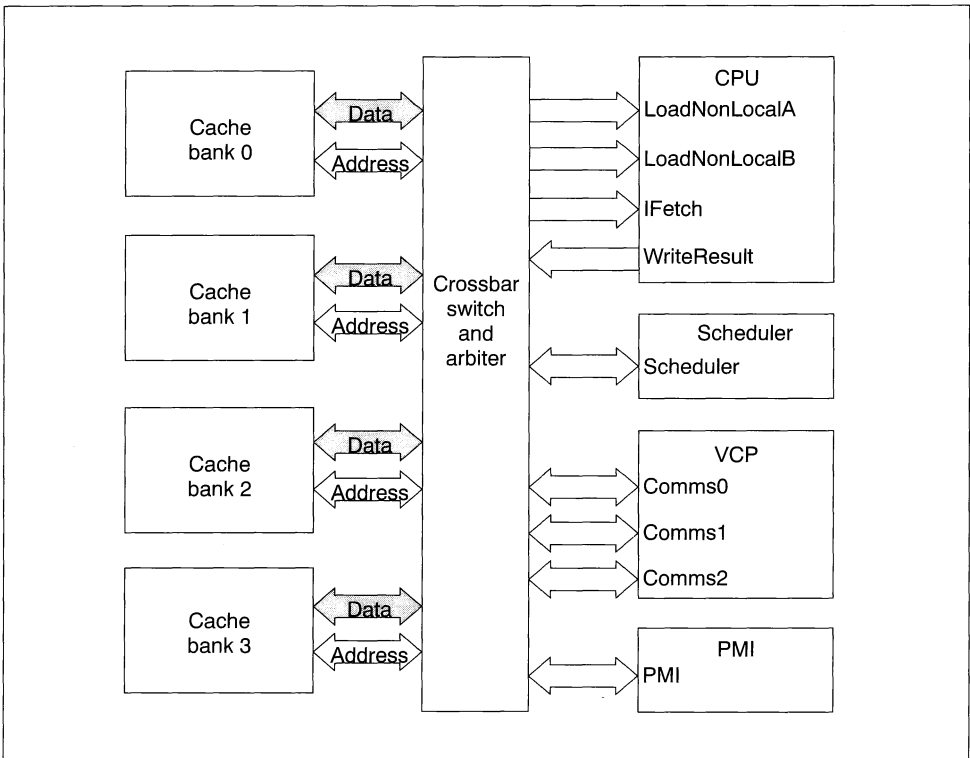


Figure 9.5 Crossbar switch and arbiter interconnections

A cache port has two operational stages, an arbitration stage and a cache access stage. Up to four ports can access the cache (one per bank) at any one time. A port can make one access per cycle. Thus, up

to 4 ports can make an access per cycle. An arbiter decides which functional unit of the IMS T9000 gains access to each of the cache banks on each cycle. Each port has its own state machine which makes requests to the arbitration unit. The port forwards the address to the PMI if a cache miss occurs.

9.2.2 Refill engine

If a physical address requested is missing from the cache, the address is passed to the cache refill engine. The refill engine arbitrates between this and earlier accesses which have misses outstanding. If the missing address is cacheable, the refill engine generates all the addresses needed to refill the associated cache line and generates requests to the PMI. The PMI fetches the line requested by the refill engine from external memory and requests it is written to the cache. The refill engine writes-back another line (selected using a random policy) in order to ensure there is a clean line ready for the next miss. If the address is marked non-cacheable only the missed address is fetched and returned to the requesting subsystem without being written to the cache.

Write-backs (dirty data written back to main memory) can be performed before refills (memory lines written to the cache) have been completed. A cache line is selected by random choice for write back by the status of the valid and dirty bits. The first word to be written back is read whilst the refill is in progress.

9.2.3 Replace pointer

Each cache bank has its own random replace pointer. The line to be replaced is selected using a pseudo-random number generator and conventional row decoder. The pseudo-random number generator generates 8 bit random numbers, sufficient to cover the 256 cache lines per cache bank.

9.3 Cache operation

9.3.1 Cache request

The sequence of events in servicing a memory request are as follows:

- The subsystem presents a request to the cache port. The port acknowledges the request once it has finished servicing any previous requests.
- The port decodes address bits **MemAddr4-5** to determine which cache bank the address is in and sends a cache bank request to the arbitration unit. On receipt of a grant signal from the arbiter the port can access the cache bank for one cycle (see section 9.3.2 for details of the arbitration algorithm).
- If the access is a cache 'hit' (i.e. the address matches a valid cache line) the port either; returns the read data, or handshakes the write data with the requesting subsystem. The port is now ready to receive the next request.
- If the access is a cache 'miss' (i.e. if the physical address requested is missing from the cache) and the PMI is ready, the PMI arbiter grants one of the current misses.
- If a miss occurs and the PMI is unable to service the miss immediately, the port goes into a retry state. It waits in this state until the PMI becomes available. Once the PMI is available the port must retry the cache to check whether the requested address has been brought into the cache during the waiting period. If the retry is a miss, the port again attempts to access the PMI. If the PMI is not granted the port must again go into a retry state and repeat the procedure.
- Once the port has gained access to the PMI, the external memory access is granted and, if necessary, the refill engine is started. On completion of the first read or write access, the port returns the read data or handshakes the write data with the requesting subsystem, and the port is ready to receive the next request.

Points to note about the operation of the port are:

- All ports (including the PMI/refill engine) are granted access to the cache banks on a cycle basis. Thus, during refill the cache bank is available for use by other subsystems.
- A port does not know whether an attempted access resulted in a cache hit or a miss. The time taken for an access to be serviced is dependent on priority of arbitration, cache contents, state of refill engine and external memory access time.
- Accesses can only be made to the PMI at the end of a cache access. Therefore it may take several cache misses before the requested access is granted use of the PMI.

The cache and PMI are pipelined into the following three stages:

- 1 cache access
- 2 PMI analyzer
- 3 external memory pads

The cache and analyzer operate in a single cycle, the PMI pads take two or more cycles.

The possible states for a cache port are shown in figure 9.6.

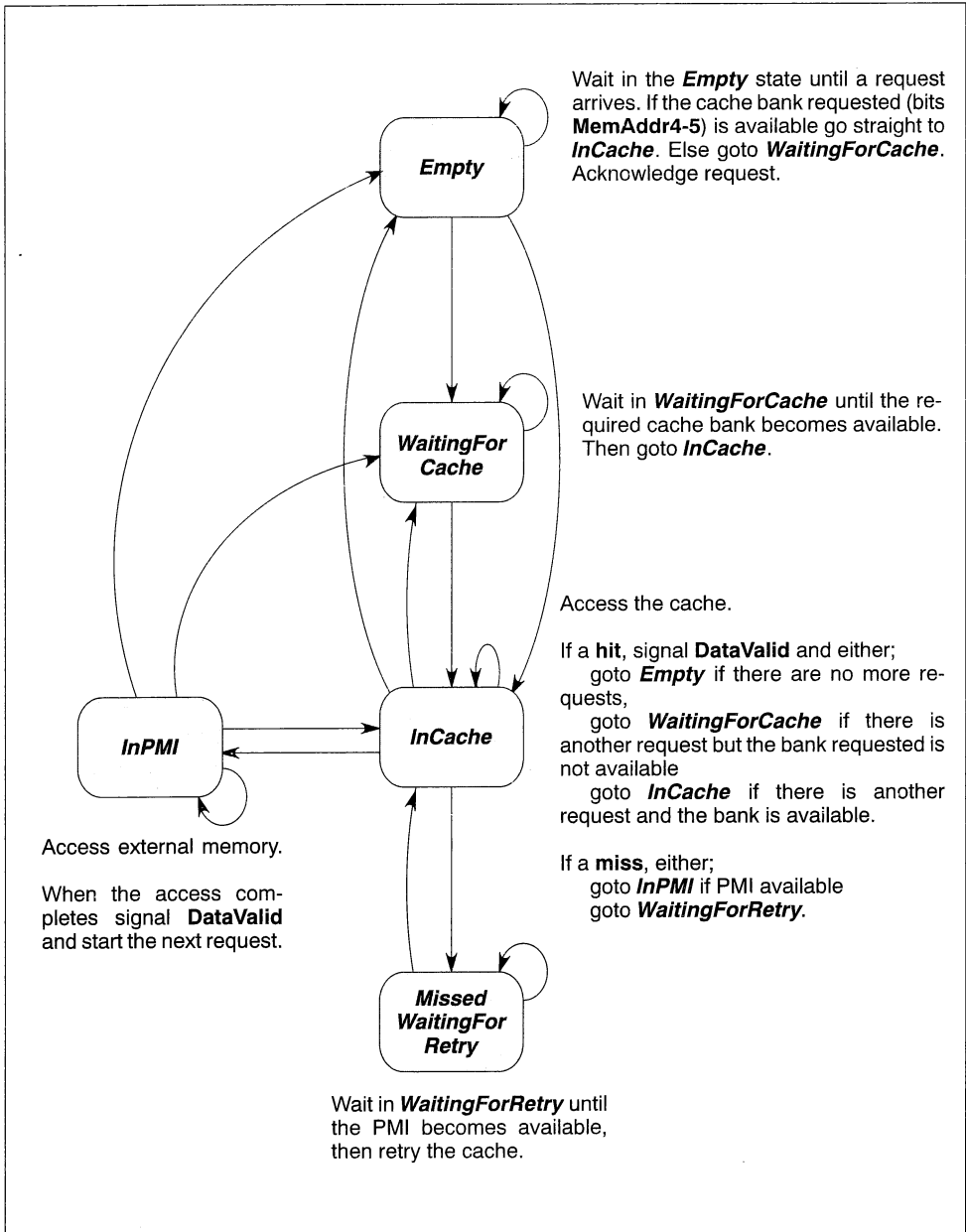


Figure 9.6 Port state diagram

9.3.2 Arbitration

There are nine requesting ports (PMI, scheduler, 3 VCP ports, 4 CPU ports), and five available resources (PMI, 4 cache banks). Each resource has its own arbitration logic. This makes sure the most efficient use is made of the resource and that the resource is shared fairly between all requesting subsystems.

The order in which the requests are processed is governed by the order of priorities. The PMI is given the highest priority and has priority over all other requesting sources. The other three co-processors (CPU, VCP and scheduler) share equal priority under a dynamic priority allocation scheme. Within the VCP and CPU, each port has a fixed order of priority as listed below. Priority 0 is the highest priority.

- Priority 0= PMI
- Priority 1= Scheduler
- Priority 1= VCP
 - 1 Comms0
 - 2 Comms1
 - 3 Comms2
- Priority 1= CPU
 - 1 IFetch (nearly empty)
 - 2 WriteResult
 - 3 LoadNonLocalB
 - 4 LoadNonLocalA
- Priority 2= IFetch

Note that the IFetch port has two priorities, low priority when filling the buffer and high priority when the instruction grouper is stalled waiting for instructions. This appears as two ports in the arbitration logic, but only one port in the crossbar switch.

Once the PMI/refill engine requests have been accounted for, the arbitration logic guarantees the CPU, VCP and scheduler each a third of the remaining cache bandwidth. Each subsystem is also guaranteed at least a third of the available external memory bandwidth.

Queueing to ensure fairness

The arbitration logic dynamically queues the three equal priority sources (CPU, VCP and scheduler) as high, medium and low preference. If more than one source requests a single resource the source with highest preference is serviced first. Each time a request is granted it is put on the back of the preference queue.

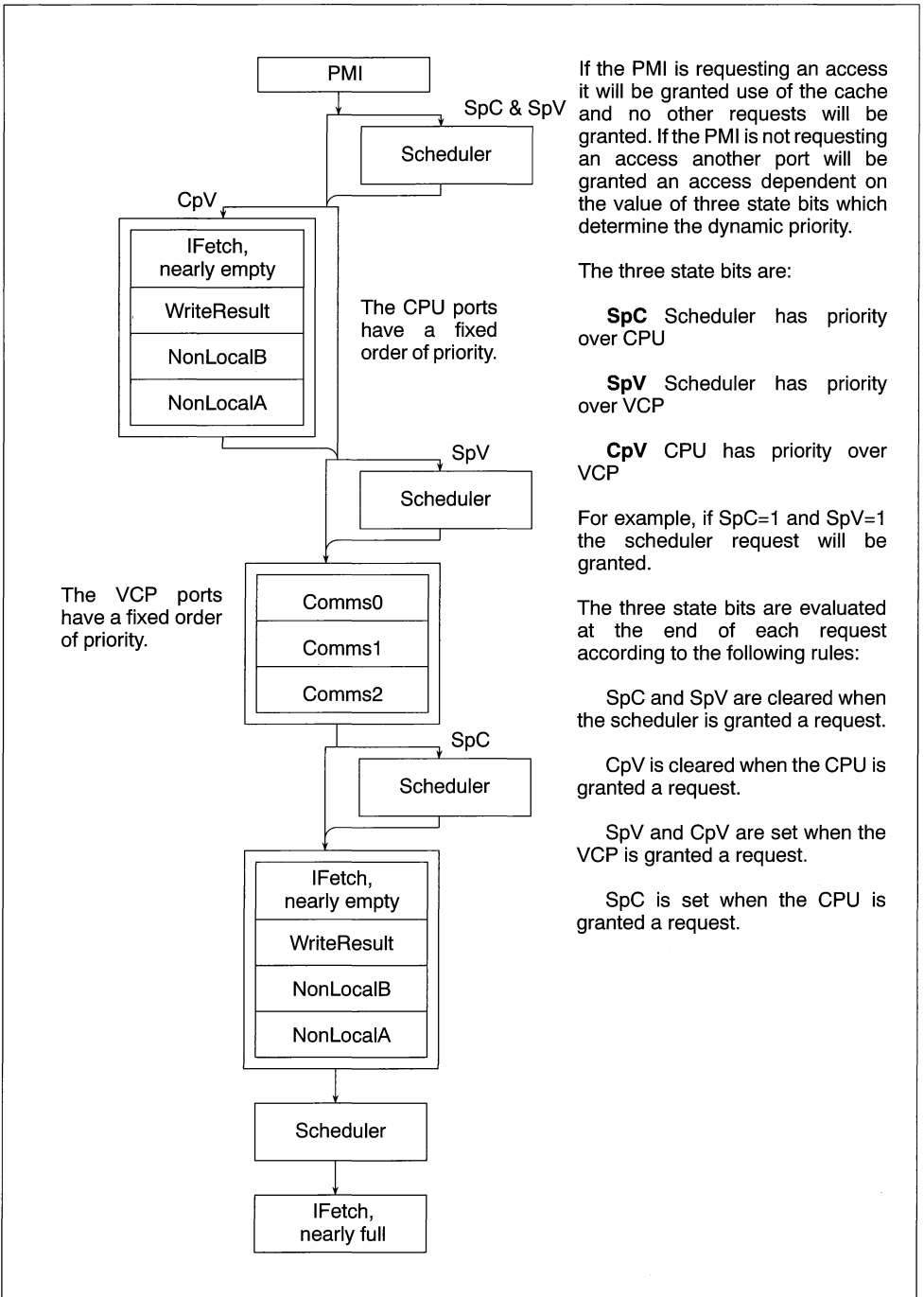


Figure 9.7 Arbitration – dynamic priorities

9.3.3 Cacheable and non-cacheable accesses

The IMS T9000 Programmable Memory Interface (PMI) can support up to four memory banks (not to be confused with the cache banks) of external memory. The PMI can mark the memory banks as cacheable or non-cacheable. This can be used for peripheral devices (for example VRAM) and shared memory systems where cache coherency problems could occur if the memory was cached. Data from external memory banks marked as non-cacheable will never be copied into the cache.

The cacheability status of addresses in the external memory banks are specified for each bank in the PMI format control registers (**FormatControl0**, **FormatControl1**, **FormatControl2**, **FormatControl3**). The **CacheMode** bit field of the register defines if the bank is occupied by devices whose contents can be transferred to the IMS T9000 cache. If the **CacheMode** bit is set to 1 the bank addresses are cacheable. Note, any bank which is programmed to have 64 bit memory is defined as a cacheable area. Refer to the Programmable Memory Interface chapter for further details on the PMI and its configuration registers.

Both cacheable and non-cacheable accesses will check the cache to see if the address wanted is in it; only if they miss the cache and are labelled cacheable will they be allocated a line in the cache.

For write accesses that hit, data is written into the cache and the dirty bits are set for the affected entries. An external access is not made immediately. The dirty cache data is only written to external memory when:

- the line is replaced when another line is brought into the cache following a miss.
- the line is flushed using the cache flush instructions.

Non-cacheable accesses

The sequence of events for a non-cacheable access that misses the cache are:

- The cache port presents the address to the cache. Note that the port does not know whether an address is marked cacheable or non-cacheable. The physical address requested is missing from the cache and the address is passed to both the PMI address analyzer and the cache refill engine. For a write access the write data is also passed to the refill engine.
- The PMI address analyzer determines that the address is non-cacheable and does not start the refill engine.
- The PMI performs the external access and, for a read, returns the data to the requesting port.

A non-cacheable access that misses does not allocate a cache line.

The subsystem requesting the memory access never knows whether the address came from the cache, generated a cache line refill, or was labelled as non-cacheable.

Note: When the cache is configured to behave as 16 Kbytes of internal RAM, the refill engine is disabled and all external RAM behaves as non-cacheable.

Cacheable accesses

Each cache bank maintains an empty cache line ready to be refilled whenever a cache miss occurs at a cacheable address.

The sequence of events for a cacheable access that misses are:

- The cache port presents the address to the cache. If a miss occurs the address is passed to both the PMI address analyzer and the cache refill engine. For a write, the write data is also passed to the refill engine.
- The PMI address analyzer determines that the address is cacheable and starts the refill engine. The cache system operates in a write-back mode, thus, data which is changed is not always written out to external memory immediately (updating of memory occurs when the line that has been changed is discarded from the cache). For a write, the PMI converts the write to an external read from the miss address.

- The PMI performs the first external access and, for a read, returns the data to the requesting port and to the refill engine which writes the data to the cache. For a read miss the line and word are marked clean. For a write miss, in particular a part-word write miss, the refill engine merges bytes from the write miss data with bytes read from the external memory to mimic the effect of a part-word write before writing the merged word to the cache and marking it dirty.
- The refill engine generates the rest of the memory addresses needed to refill the cache line. When the last word is written to the cache, the cache line is marked valid. For a 32 bit external memory system three more external reads are performed. For a 64 bit external memory system only one more external read is performed after the miss address. The order of external reads is given in section 9.3.4 below which details cache refill cycles.
- The refill engine randomly chooses a cache line to write back to the external memory. It reads the tag, valid bit and dirty line bit along with the first data word and dirty word bit for that line. If the line is valid and dirty the write back machine is started. The cache line is marked as invalid signifying that the line is empty and available for the next refill operation.
- The four write back words are passed to the PMI which writes them to external memory if dirty.

For cacheable read and write accesses that hit, the port returns the read data or handshakes the write data with the requesting subsystem.

9.3.4 Cache refill cycles

Any read from external memory can become a cache-refill cycle. The refill engine examines the value of the **CacheMode** bit in the PMI **FormatControl0-3** register for the selected external memory bank (refer to the Programmable Memory Interface chapter for further details on the PMI registers). If the access is a read or write to a cacheable location the refill engine will initiate a cache-line refill cycle for all four words of the cache line. The refill engine generates the subsequent addresses of the cache line. The refill engine is defined to present the miss address first and then 'wraparound' the other addresses as defined in table 9.2 below.

64 bit port sizes are defined to be cacheable and any reads and writes always transfer full 64 bit data from external memory to the internal cache. No byte transfers take place, therefore no byte writes are needed. During 64 bit reads the **notMemWrB0-3** strobes are all inactive. During 64 bit writes all of the **notMemWrB0-3** strobes are active. For transfers from a 64 bit interface the full cache line is transferred in two external reads and **MemAddr2** is not used. The first external read is the miss address, see figure 9.8.

Address bits **MemAddr2-3** select the word in the cache line. The first **MemAddr2-3** pair defines the miss address presented by the refill engine and transferred to the external address pins. Table 9.2 defines the low order address bit activity during a cache line refill cycle from a non-64 bit interface. This is also illustrated in figure 9.8 which shows the order of external reads for cache refill cycles from 64 bit and non-64 bit interfaces.

First MemAddr3:2 (miss address)	Second MemAddr3:2	Third MemAddr3:2	Fourth MemAddr3:2
00	01	10	11
01	00	11	10
10	11	00	01
11	10	01	00

Table 9.2 Low order address bit activity for a non-64 bit cache line refill cycle

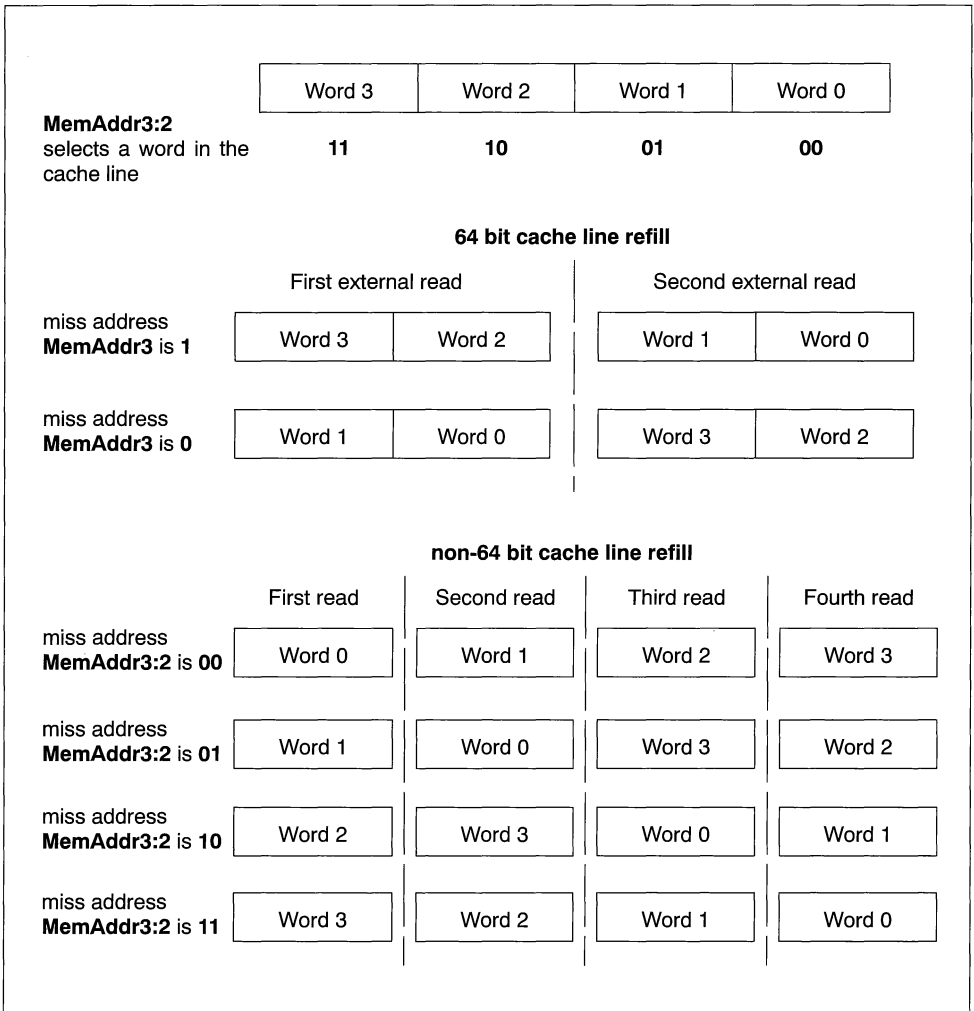


Figure 9.8 Order of external reads for cache refill cycles from 64 bit and non-64 bit interfaces

Cache refills from 8/16 bit ports

For port sizes of 16 or 8 bits the PMI generates the low order address bits (**A0-A1**) on the **notMemWrB2-3** strobes, as defined in table 9.3 below, to assemble full 32 bit operands. This is repeated for each of the four words which make up the required cache line.

	External port size	
	16 bit	8 bit
notMemWrB3	becomes A1	becomes A1
notMemWrB2	undefined	becomes A0

Table 9.3 Allocation of **A0-A1** address bits

Write-back cycles

The cache is a write-back cache, thus main memory is only updated when a line from the cache is selected for displacement and written out to main memory. Each 32 bit word in the cache has a dirty bit associated with it. This dirty bit is set whenever any of the IMS T9000 internal functional units writes to an address location which is currently cached. When write-back cycles are performed, only those words in the line which have their associated dirty bit set, are written to main memory.

For 64 bit write-back cycles, if one or more of the two 64 bit words in the line has a dirty bit set a write back will occur. This is illustrated in table 9.4. Refer back to figure 9.3 showing the cache line structure to see how the dirty bits map onto the cache line words.

Dirty bits				Write activity
D3	D2	D1	D0	
0	0	0	0	No writes
1	X	0	0	Writes 32 bit words 2 and 3 (64 bits)
X	1	0	0	
0	0	1	X	Writes 32 bit words 0 and 1 (64 bits)
0	0	X	1	
1	X	1	X	Writes 32 bit words 0, 1, 2 and 3 (two 64 bit writes)
1	X	X	1	
X	1	1	X	
X	1	X	1	

where,

D is the dirty word bit
 1 indicates the word is dirty
 X indicates 0 or 1

Table 9.4 64 bit write-back cycles

Table 9.5 below defines **MemAddr2-3** activity for a non-64 bit write-back cycle, assuming all of the words in the line are dirty. In this case the addresses follow the binary sequence defined below. If any of the elements are not dirty then the word is not written out. In which case the next address is the next dirty element of the line whose address follows the sequence.

First MemAddr3:2	Second MemAddr3:2	Third MemAddr3:2	Fourth MemAddr3:2
00	01	10	11

Table 9.5 Low order address bit activity for a non-64 bit write-back cycle

DMA and cache-refill cycles

External DMA requests are sampled at the end of the complete cache-refill write-back operation. Thus, the operation is not interrupted with DMA activity.

9.4 Cache instructions

The IMS T9000 provides four instructions to support interfacing the cache to external hardware systems. The instructions are provided to support coherency in shared memory systems when the internal memory is configured as cache.

The cache instructions reference cache lines 0 to 1023, with the lines spread over the four cache banks. The line structuring of the four cache banks is shown in figure 9.9.

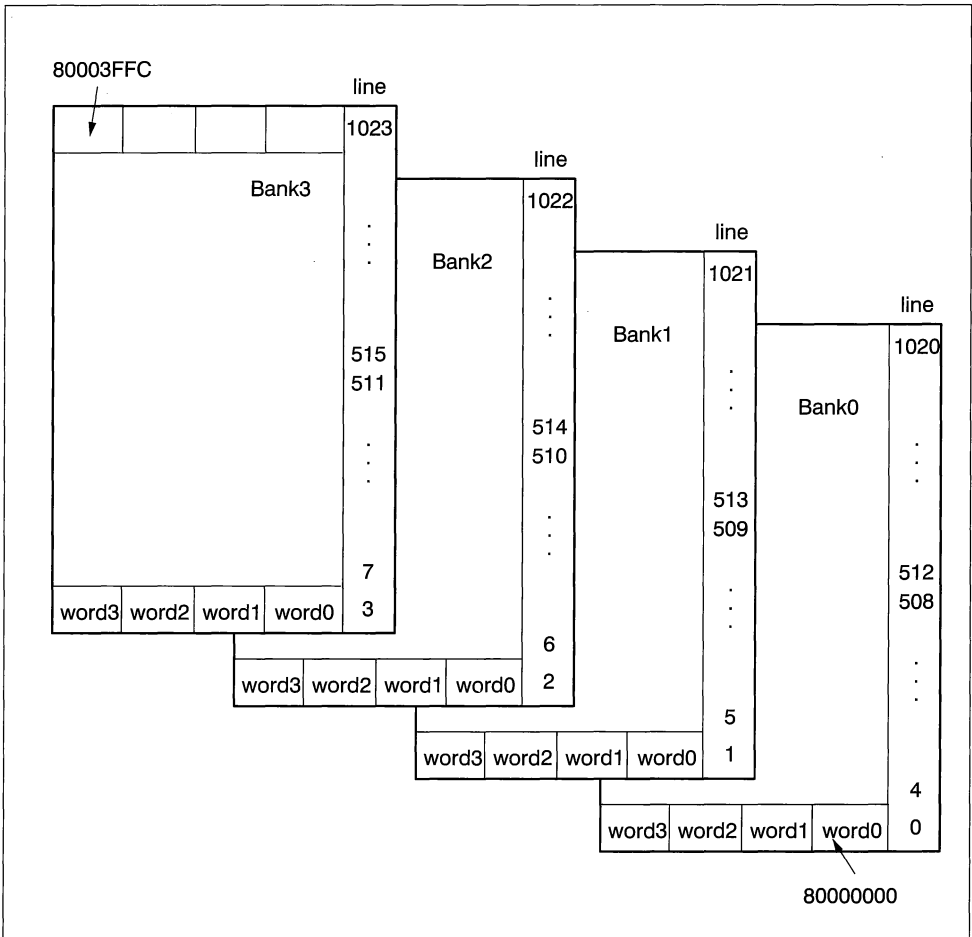


Figure 9.9 Line structuring of the cache banks

The allocation of the cache lines for different configurations are shown in table 9.6.

	TSRAM	Cache	Cache lines
All cache	0 Kbytes	16 Kbytes	0 to 1023
Half cache, half TSRAM	8 Kbytes	8 Kbytes	512 to 1023
All TSRAM	16 Kbytes	0 Kbytes	—

Table 9.6 Cache/TSRAM allocation

9.4.1 Flushing data from the cache

To maintain coherency between the data in the cache and external memory the IMS T9000 provides two instructions to flush data from the cache in order to update external memory. The user can specify an address to be flushed, if the address is present in the cache the entire line containing that address is flushed, and the line is labelled clean. For large blocks of data the user can specify a line number and an address range. If a line contains an address within the specified range, the line is flushed.

Flush dirty cache address (*fdca*)

If the byte address specified by **AReg** is present in the cache, and the line is dirty, the *fdca* instruction causes the line to be written back to external memory. If the address is not in the cache, no action is taken. **AReg** is incremented by the number of bytes per line (16). The line in the cache remains valid and is marked clean.

Flush dirty cache line (*fdcl*)

If the byte address (stored in the cache line tag) specified by **AReg** is within the address range specified by **BReg** and **CReg** inclusive, and the line is dirty, the *fdcl* instruction causes the line to be written back to external memory. **AReg** is incremented by 1. The line in the cache remains valid and is marked clean. The *fdcl* instruction will normally be repeated for all cache lines (512 or 1024 depending on cache allocation).

9.4.2 Invalidate cache block

There are two instructions on the IMS T9000 which can be used to invalidate specified cache lines. When access is made to an invalid cache line the data is retrieved from external memory.

Invalidate cache address (*ica*)

If the byte address specified in **AReg** is present in the cache the line is marked as invalid. If the address is not present in the cache, no action is taken. **AReg** is incremented by the number of bytes per line (16). Note, if the line specified contains dirty data, it is not written back to memory, thus it is necessary to ensure that other processes are not using the same cache line.

Invalidate cache line (*icl*)

If the address (stored in the cache line tag) specified by **AReg** is within the range specified by **BReg** and **CReg** inclusive, the cache line is invalidated. **AReg** is incremented by 1. Note, any dirty data which was in that cache line is not written back to memory and is lost. The *icl* instruction will normally be repeated for all cache lines (512 or 1024 depending on cache allocation).

9.4.3 Cache instruction performance

The number of cycles required to perform an operation depends on whether the address is present in the cache. For a flush operation it also depends on the number of dirty words which are flushed back, this is specified in table 9.7 below.

Instruction	Number of cycles	Condition	Notes
<i>fdca</i>	5	Address not in cache	1
	6	Address in cache, line clean	
	14	Address in cache, all 4 words in cache line dirty	
<i>fdcl</i>	5	Address not in range	1
	8	Address in range, line clean	
	17	Address in range, all 4 words in cache line dirty	
<i>ica</i>	5	Address not in cache	
	8	Address in cache	
<i>icl</i>	5	Address not in range	
	8	Address in range	

Notes

- 1 Assuming three cycle, 32 bit external memory.

Table 9.7 Number of cycles taken by an instruction

9.5 Cache configuration registers

The cache (in common with a number of other sub-systems of the IMS T9000) is controlled via registers in a configuration space. The registers are accessed via the *ldconf* and *stconf* instructions, or via *CPeek* and *CPoke* command messages received along control link **CLink0**. This section describes the functionality of the cache to be controlled by the associated configuration registers.

Note: All undefined and INMOS reserved bits in the configuration registers must be written with 0 unless otherwise stated.

9.5.1 RamSize and DoRamSize registers

The size of the cache is determined by the setting of the cache configuration registers. The **RamSize** register defines the amount of RAM which is allocated to be internal RAM. It can be programmed to be 0, 8 or 16 Kbytes.

RamSize		#2001	Read/Write
Bit	Bit field	Function	
1:0	RAMsize	Determines how much of the cache is allocated as internal RAM. 00 16 Kbytes of internal RAM 01 8 Kbytes of internal RAM 11 0 Kbytes of internal RAM	
31:2		Undefined	

Table 9.8 Bits in the **RamSize** register

Writing a '1' to the **DoRamSize** register changes the size of the cache to that defined by the **RamSize** register and forces a write-back so that there is a clean line ready for the next miss.

DoRamSize		#2002	Write only
Bit	Bit field	Function	
0	DoRAMsize	Sets the size of the cache following the setting of the RAMSize register.	

Table 9.9 Bit in the **DoRamSize** register

9.5.2 RamLineNumber, RamAddress and DoAllocate registers

Internal RAM is implemented by locking lines into the cache. The **RamLineNumber** and **RamAddress** registers allow the addresses of the locked cache lines to be configured by the user. This enables the RAM to be located anywhere in the processors physical address range. The cache controller manages the allocation of cache line addresses to ensure coherency.

Writing a '1' to the **DoAllocate** register allocates the cache line specified by the **RamLineNumber** register with the RAM address specified in the **RamAddress** register. The previous content of the line are written back, if necessary, and data at the RAM address is fetched from external memory.

RamLineNumber #2003		Read/Write
Bit	Bit field	Function
7:0	Line number	Specifies line number of the locked line.
31:8		Undefined

Table 9.10 Bits in the **RamLineNumber** register

RamAddress #2004		Read/Write
Bit	Bit field	Function
31:4	RAM address	Address of locked cache line.
3:0		Undefined

Table 9.11 Bits in the **RamAddress** register

DoAllocate #2005		Write only
Bit	Bit field	Function
0	DoAllocate	Locks cache lines following the setting of the RamLineNumber and RamAddress registers.

Table 9.12 Bit in the **DoAllocate** register

9.6 Initialization of the cache

9.6.1 Reset state

Following a hard reset or a soft reset to level 1, the contents of the cache are discarded and the cache is configured as the first 16 Kbytes of memory (byte addresses 80000000 to 80003FFF). The contents of any external memory at these addresses are masked, and the previous contents of the internal memory are lost. The cache is configured to behave like internal memory and the refill engine is turned off. All lines are marked as clean. Following reset the hex addresses contained in each line are as shown in figure 9.13.

Line	Bank 3	Bank 2	Bank 1	Bank 0
255	80003FFF–80003FF0	80003FEF–80003FE0	80003FDF–80003FD0	80003FCF–80003FC0
...				
128	8000203F–80002030	8000202F–80002020	8000201F–80002010	8000200F–80002000
127	80001FFF–80001FF0	80001FEF–80001FE0	80001FDF–80001FD0	80001FCF–80001FC0
...				
1	8000007F–80000070	8000006F–80000060	8000005F–80000050	8000004F–80000040
0	8000003F–80000030	8000002F–80000020	8000001F–80000010	8000000F–80000000

Table 9.13 Addresses contained in each line after reset

9.6.2 Starting the cache

After power up and reset the PMI is disabled. The IMS T9000 transputer must be bootstrapped either from a ROM device (such as an EPROM or a Flash EPROM), or down control link (**CLink0**) from a host transputer (or a computer with a link adaptor) with the bootstrap code executed in internal memory. The PMI configuration registers are set up by the bootstrap code to match the external hardware, and the PMI is enabled.

The cache size can be configured using the **RamSize** and **DoRamSize** registers (see section 9.5.1). The refill engine starts up and randomly chooses a cache line to write back so that there is space to store the first miss address. The refill engine continues replacing lines at random when cache misses occur.

Following reset the address space for different cache/TSRAM allocations is shown in figure 9.10.

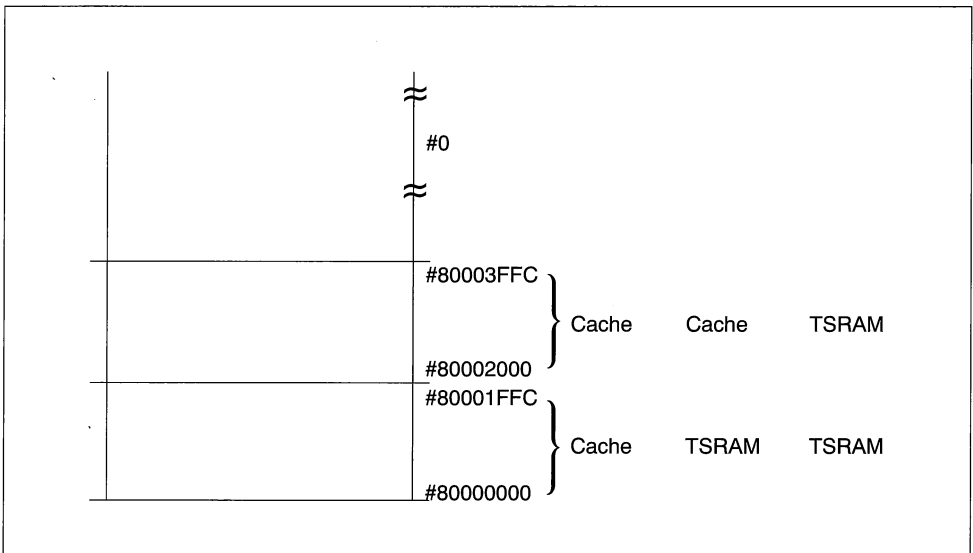


Figure 9.10 Address space for different cache/TSRAM allocations

All 64 bit port sizes are defined as cacheable and always transfer a full 64 bit operand to and from external memory to the internal cache. Thus, internal memory must be configured as 8 or 16 Kbytes of cache before 64 bit external memory accesses can be made. On start-up internal memory is configured as 16 Kbytes of internal RAM and no cache. Therefore, in order to ensure correct operation of the cache system the following sequence of events must be followed.

- 1 Enable the PMI and configure the external memory bank as 64 bit cacheable memory.
- 2 Configure internal memory as 8 or 16 Kbytes of cache.

Note: 64 bit external memory should not be used before the cache has been turned on.

10 Programmable memory interface

The IMS T9000 programmable memory interface (PMI) can access a 4 Gbyte physical address space, and provides a peak bandwidth of 200 Mbytes/sec. It is designed to support memory subsystems with minimal external support logic. The interface has internal logic to provide decode and timing control functions and can be programmed through the configuration registers as described in section 10.3 below.

The external address space is partitioned into four banks (not to be confused with the four cache banks). This allows the implementation of mixed memory systems, with support for DRAM, SRAM, EPROM, VRAM and I/O (see figure 10.1). The timing of each of the four memory banks can be programmed separately, with a different device type being placed in each bank with no external hardware support. The PMI has a 64 bit data bus, and each bank of memory can be configured to be 8, 16, 32 or 64 bits wide. The PMI directly supports: 8, 16, 32 and 64 bit SRAM; 32 and 64 bit DRAM. All banks programmed to be 64 bit wide memory are defined as cacheable and always transfer a full 64 bit operand to and from external memory to the internal cache, providing fast cache refill. The full performance of the IMS T9000 transputer can be exploited using relatively low-cost DRAM, and up to 8 Mbytes of DRAM can be connected with no external components.

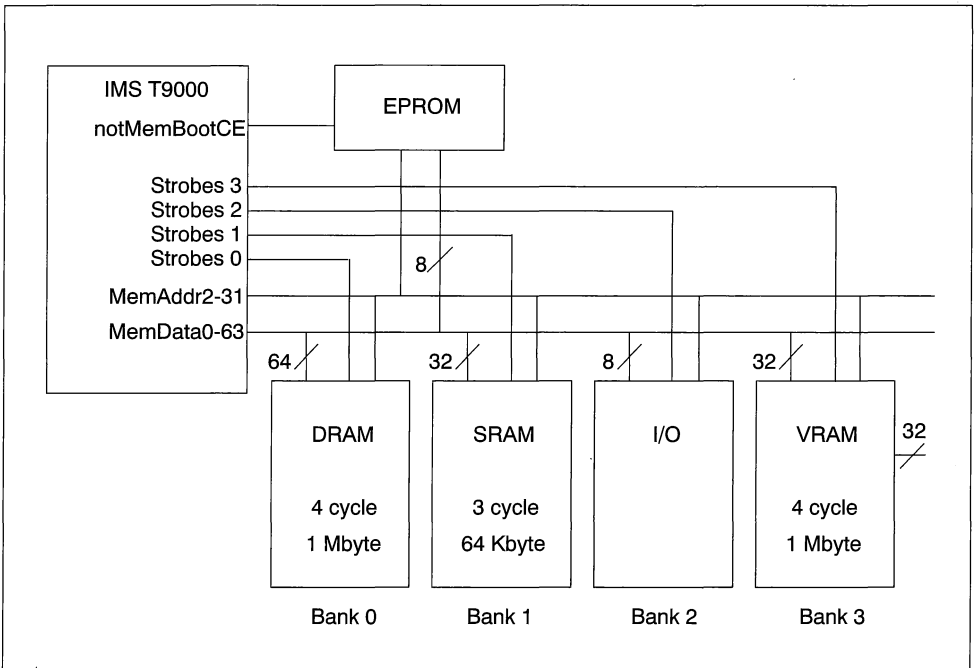


Figure 10.1 Example of a mixed memory system using a T9000 transputer

In this chapter a *cycle* is one processor clock cycle and a *phase* is one quarter of the duration of one processor clock cycle.

10.1 Pin functions

10.1.1 MemData0-63

The data bus transfers 64, 32, 16 or 8 bit data items depending on the bus width configuration. For 64-bit data items the most significant bit is carried on **MemData63**. For 32 bit data items the most significant bit is **MemData31**. **MemData0-15** transfers 16 bit data items, and **MemData0-7** transfers 8 bit data items. The least significant bit of the data bus is always **MemData0**.

10.1.2 MemAddr2-31

The address bus may be operated in both multiplexed and non-multiplexed modes. When a bank is configured to contain DRAM, or other multiplexed memory, then the internally generated 32 bit address is multiplexed as row and column addresses through the external address bus. The multiplexing is controlled for each bank by the format control registers (**FormatControl0**, **FormatControl1**, **FormatControl2**, **FormatControl3**), see page 165.

10.1.3 notMemWrB0-3

The transputer uses word addressing and four byte-write strobes are provided.

64 bit wide memory is defined as an array of 8 byte words with **MemAddr3-31** selecting an 8 byte word. **MemAddr2** is 0 for 64 bit addressing. No further addressing is performed for 64 bit memory. 32 bit wide memory is defined as an array of 4 byte words with **MemAddr2-31** selecting a 4 byte word. Each byte of this array is addressable with the byte enable pins **notMemWrB0-3** selecting a byte within a word. 16 bit wide memory is defined as an array of 2 byte words with 31 address bits selecting a 2 byte word and **notMemWrB0-1** selecting a byte within the word. 8 bit wide memory is defined as an array of 1 byte words with 32 address bits selecting a word. For 16 bit and 8 bit wide memory, the lower order address bits (**A0** and **A1**) are multiplexed onto the unused byte-write pins to give an address bus 31 or 32 bits wide respectively.

notMemWrB0 addresses the least significant byte of a word. All four strobes have the same timing and are only active during write cycles. The timing is controlled by the write strobe registers (**WriteStrobe0**, **WriteStrobe1**, **WriteStrobe2**, **WriteStrobe3**), see page 169.

The function of the byte enables **notMemWrB0-3** for different bank size configurations is given in table 10.1 below. Note that other bus masters must not drive the same data pins during a write.

	External port size			
	64 bit	32 bit	16 bit	8 bit
notMemWrB3	set active (0)	enables MemData24-31	becomes A1	becomes A1
notMemWrB2	set active (0)	enables MemData16-23	undefined	becomes A0
notMemWrB1	set active (0)	enables MemData8-15	enables MemData8-15	undefined
notMemWrB0	set active (0)	enables MemData0-7	enables MemData0-7	enables MemData0-7

Table 10.1 **notMemWrB0-3** pins

10.1.4 notMemRAS0-3

The four programmable RAS strobes are controlled by the timing control registers (**TimingControl0**, **TimingControl1**, **TimingControl2**, **TimingControl3**) and RAS strobe registers (**RASStrobe0**, **RASStrobe1**, **RASStrobe2**, **RASStrobe3**), see section 10.3.2. One strobe is allocated to each of the four

banks which are decoded on chip. If a bank is programmed to contain DRAM, or other multiplexed memory, then the associated **notMemRAS** pin acts as its RAS strobe by default. For banks which do not contain DRAM the **notMemRAS** pin is available as a general purpose programmable strobe.

10.1.5 notMemCAS0-3

The four programmable CAS strobes are controlled by the CAS strobe registers, see page 169. One strobe is allocated to each of the four banks which are decoded on chip. If a bank is programmed to contain DRAM, or other multiplexed memory, then the associated **notMemCAS** pin acts as its CAS strobe by default. For banks which do not contain DRAM the **notMemCAS** pin is available as a general purpose programmable strobe.

10.1.6 notMemPS0-3

These four additional programmable strobes are controlled by the programmable strobe registers, see page 169. One strobe is allocated to each of the four banks which are decoded on chip.

10.1.7 MemWait

Wait states can be selected by taking **MemWait** high. **MemWait** is sampled during **RASTime** and **CAS-Time**. **MemWait** retains the state of any strobe during the cycle in which **MemWait** was asserted. **MemWait** suspends the cycle counter and the strobe generation logic until deasserted. When **MemWait** is deasserted cycles continue as programmed by the configuration registers.

10.1.8 MemReqIn, MemGranted

Direct memory access (DMA) can be requested at any time by driving the asynchronous **MemReqIn** signal high. The address and data buses are tristated after the current memory access or refresh cycle terminates. If the current memory cycle is part of a cache line write back or fill then the four words of the line are transferred before the buses are tristated.

Strobes are left inactive during the DMA transfer. If a DMA is active for longer than one programmed refresh interval then external logic is responsible for providing refresh.

MemGranted follows the timing of the bus being tristated and can be used to signal to the device requesting the DMA that it has control of the bus.

Table 10.2 below lists the processor pin state while **MemGranted** is asserted.

MemGranted asserted	
Pin name	Pin state
MemAddr2-31	floating
MemData0-63	floating
notMemWrB0-3	inactive
notMemRAS0-3	inactive
notMemCAS0-3	inactive
notMemPS0-3	inactive
notMemRf	inactive
MemReqOut	active
notMemBootCE	inactive

Table 10.2 Pin states while **MemGranted** is asserted

10.1.9 MemReqOut

The **MemReqOut** pin indicates to external logic that IMS T9000 external bus cycles are pending and execution will stall if a DMA transfer is initiated, or has stalled if a DMA transfer is in progress.

Once a DMA transfer has been granted the IMS T9000 processor can continue to execute out of the internal cache until an access to external memory is required. The **MemReqOut** pin will be taken high and external logic can use this information to interrupt the DMA transfer in progress. The external logic should deassert **MemReqIn** when the memory buses are available for the processor to use.

10.1.10 notMemBootCE

The IMS T9000 has a dedicated area of external memory address space of fixed size and timing. This functions as a fifth bank with fixed decode and timing parameters. This is to provide slow access to configuration/ bootstrap code stored in ROM. **notMemBootCE** is used to access external memory placed in this dedicated address range. This address space can also be used to access code/data which is not bootstrap code if required. Refer to the booting section 10.6 for more information.

10.1.11 notMemRf

The IMS T9000 can be operated with memory refresh enabled or disabled. The selection is made during memory configuration, when the refresh signal is also determined.

notMemRf indicates that the current cycle is a refresh cycle. It is asserted low at the beginning of the refresh cycle and deasserted high at the end of the refresh cycle.

10.1.12 notMemStrobe

Reference strobe for external bus cycles.

10.2 External bus cycles

The IMS T9000 programmable memory interface is designed to provide efficient support for dynamic memory without compromising support for other devices, such as static memory and I/O devices. This flexibility is provided by allowing the required waveforms to be programmed via the configuration registers described in section 10.3.

Interaction of the PMI with the on-chip cache is highly optimized. In order to support specialized memory types, addresses within 8, 16 or 32 bit memory banks can be specified to be cacheable or non-cacheable. Note, 64 bit memory is always defined as cacheable. In addition, each bank can be specified to contain: 8, 16, 32 or 64 bit wide SRAM; or 32 or 64 bit wide DRAM memory. The PMI synthesizes the required number of cycles to assemble full words before transferring them to or from the internal cache.

Transputer memory is byte addressed, with words aligned on eight-byte boundaries for 64 bit devices, four-byte boundaries for 32 bit devices and on two-byte boundaries for 16 bit devices.

During read cycles byte level addressing is performed internally by the IMS T9000. The PMI can read bytes, half-words, words or dual-words. It always reads the size of the bank.

During write cycles the IMS T9000 uses the **notMemWrB0-3** strobes to perform addressing of bytes. If a particular byte is not to be written then the corresponding data outputs are tristated. Writes can be less than the size of the bank.

The internally generated address is indicated on pins **MemAddr2-31**, however the three low order address bits **A0**, **A1** and **A2** have different functions depending on the size of the external data bus. The least significant bit of the data bus is always **MemData0**. The most significant bit can be adjusted dynamically to suit the required external bus size.

Note that pins which are not used during an access are tristated. For example; for an 8 bit bus, pins **MemData8-63** are tristated. In addition, when partial writes are performed pins not used are tristated in order to avoid bus contention. For example; for a 32-bit write to a 16 bit external bus with the internal byte writes having the values 0110, **MemData16-63** would be tristated throughout the two accesses because they are not used, while during the first access **MemData0-7** would be tristated, and during the second access **MemData8-15** would be tristated, to avoid bus contention.

The two low order address bits **A0** and **A1** are encoded onto two of the byte enable pins (**notMemWrB2-3**) as defined in table 10.1. The function of the byte write pins when writing 32 bit operands to 32, 16 or 8 bit port sizes are illustrated in figure 10.2.

Table 10.3 represents the byte writes and lower order address bits activity, whilst transferring operands to arbitrary port sizes. For example; to transfer a write with internal byte enable values of 0110 to a 16 bit port, two accesses are required as shown in table 10.3.

Internal byte control	Transfer size (bytes)	External port size						
		32-bit		16-bit		8-bit		
		notMemWrB3:0 – sequence	notMemWrB1:0 A1 sequence	notMemWrB0 A1:A0 sequence				
		T _n	T _n	T _{n+1}	T _n	T _{n+1}	T _{n+2}	T _{n+3}
0001	1	1110	10 0	– –	0 00	– –	– –	– –
0010	1	1101	01 0	– –	0 01	– –	– –	– –
0011	2	1100	00 0	– –	0 00	0 01	– –	– –
0100	1	1011	10 1	– –	0 10	– –	– –	– –
0101	2	1010	10 0	10 1	0 00	0 10	– –	– –
0110	2	1001	01 0	10 1	0 01	0 10	– –	– –
0111	3	1000	00 0	10 1	0 00	0 01	0 10	– –
1000	1	0111	01 1	– –	0 11	– –	– –	– –
1001	2	0110	10 0	01 1	0 00	0 11	– –	– –
1011	3	0100	00 0	01 1	0 00	0 01	0 11	– –
1100	2	0011	00 1	– –	0 10	0 11	– –	– –
1101	3	0010	10 0	00 1	0 00	0 10	0 11	– –
1110	3	0001	01 0	00 1	0 01	0 10	0 11	– –
1111	4	0000	00 0	00 1	0 00	0 01	0 10	0 11

where,

T_n = external bus cycle.

T_{n+1}, T_{n+2}, T_{n+3} refer to subsequent bus cycles

Table 10.3 Low order address activity

Table 10.4 shows the mapping of the data bits onto the external data pins. It should be read in conjunction with figure 10.2 to show the function of the byte write pins when writing operands to the different port sizes.

Port size	Data pins				
	MemData63:32	MemData31:24	MemData23:16	MemData15:8	MemData7:0
64 bit	d63:d32	d31:d24	d23:d16	d15:d8	d7:d0
32 bit	tri-stated	d31:d24	d23:d16	d15:d8	d7:d0
16 bit	tri-stated	tri-stated	tri-stated	d15:d8	d7:d0
	tri-stated	tri-stated	tri-stated	d31:d24	d23:d16
8 bit	tri-stated	tri-stated	tri-stated	tri-stated	d7:d0
	tri-stated	tri-stated	tri-stated	tri-stated	d15:d8
	tri-stated	tri-stated	tri-stated	tri-stated	d23:d16
	tri-stated	tri-stated	tri-stated	tri-stated	d31:d24

Table 10.4 Mapping of internal data bits (d63:d0) onto external data pins (**MemData63:0**)

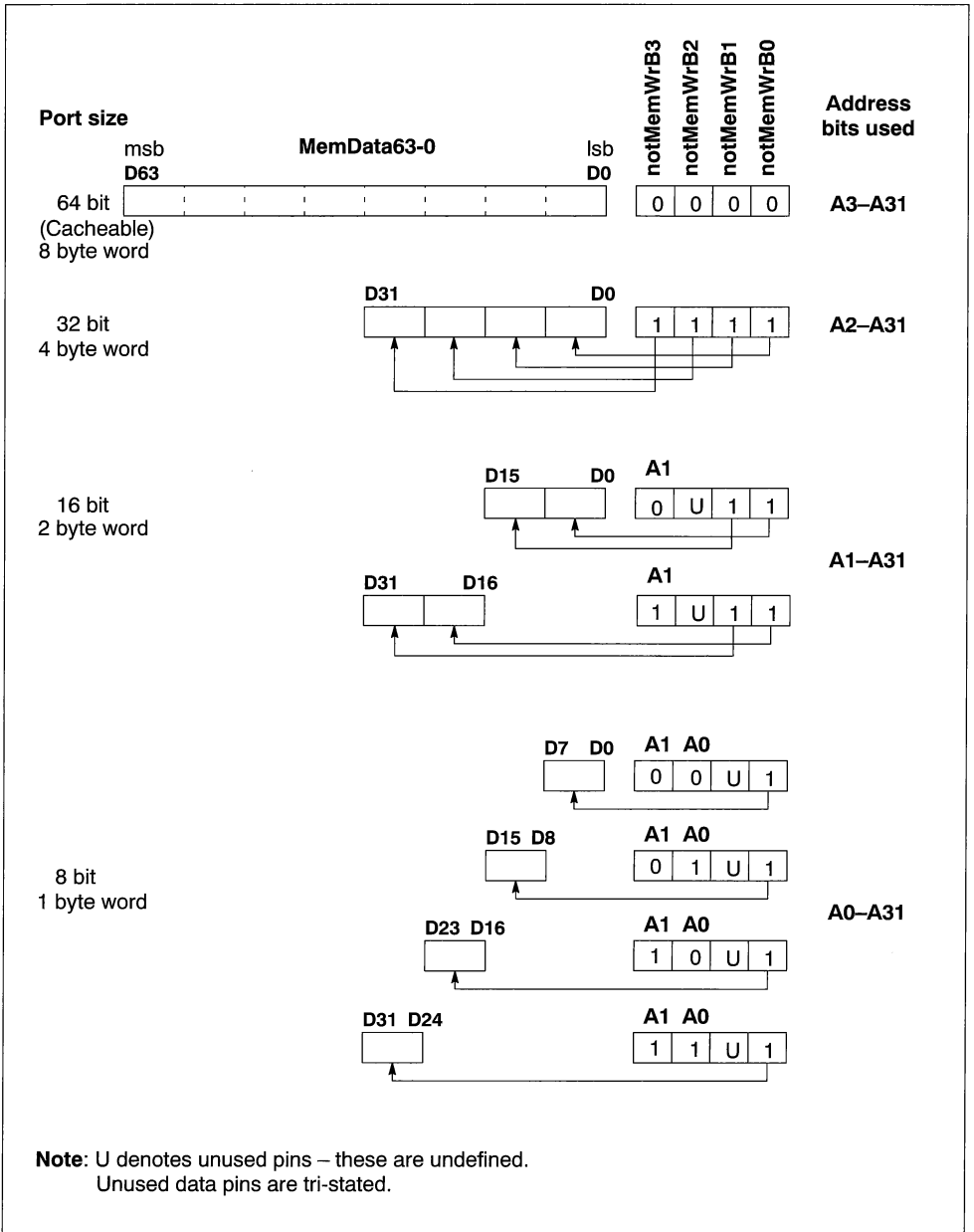


Figure 10.2 Setting of the byte write pins when writing 32 bit operands to 32, 16 or 8 bit port sizes

A generic memory interface cycle consists of a number of defined periods, or times, as shown in figure 10.3. This generic memory cycle uses DRAM terminology to clarify the use of the interface in the most complex situations, but can be programmed to provide waveforms for a wide range of other device types. The timing of each of the four memory banks can be programmed separately, with a different device type being placed in each bank with no external hardware support.

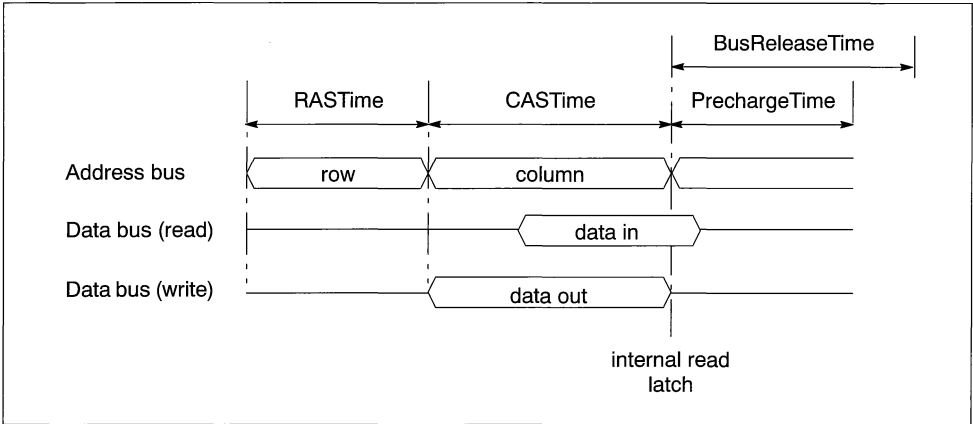


Figure 10.3 Generic memory cycle

The **RASTime** and **CASTime** are consecutive. The **CASTime** can be followed by concurrent **Precharge** and **BusRelease** times. Thus, for DRAM, the times are used for RAS, CAS, and precharge respectively. For non-multiplexed addressed memory the **RASTime** can be programmed to be zero.

If the **RASTime** is programmed to be non-zero, and page-mode memory is programmed in a bank, the **RASTime** will only occur if consecutive accesses are not in the same page. The **RASTime** will not commence until the **PrechargeTime** for a previous access to the same bank has completed. During this time the address is multiplexed by the amount specified in the format control register for the bank so as to output the row address on the address bus. During the **RASTime** a transition can be programmed on the RAS strobe, but not on any other strobe.

During the **CASTime** the programmable strobes and byte-write strobes are active. The address is output on the address bus without being shifted. Write data is valid during **CASTime**. Read data is latched into the interface during the last clock cycle of the **CASTime**.

The **PrechargeTime** and **BusReleaseTime** commence concurrently at the end of the **CASTime**. A **PrechargeTime** will occur to the current bank if:

- the next access is to the same bank but to a different row address.
- the next cycle is to a different bank.

The **BusReleaseTime** runs concurrently with the **PrechargeTime** and will occur if:

- the current cycle is a read and the next cycle is a write.
- the current cycle is a read and the next cycle is a read to a different bank.

The **BusReleaseTime** is provided to allow slow devices to float to a high impedance state.

10.2.1 External DRAM cycles

The IMS T9000 interface has logic to utilize page-mode DRAM. The internal logic determines if page-mode accesses are appropriate and constructs the required waveforms as defined by the timing control register for the bank. For random accesses to dynamic memory the interface will execute a **RASTime**,

followed by a **CAS**Time, followed by a **Precharge**Time. Figure 10.4 shows a random access to dynamic memory in bank 0.

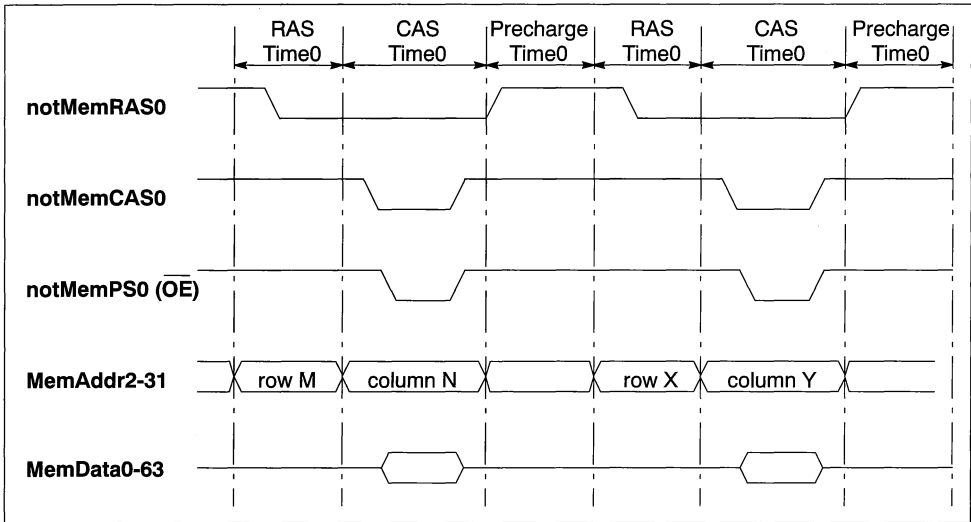


Figure 10.4 Random read access to DRAM from bank 0

For consecutive accesses within the same page in a single bank the row address remains constant and only subsequent column addresses change. To perform a cache line transfer 4 consecutive addresses are transferred, and a **RAS**Time sub-cycle is only required for the first transfer across the external data bus. This may be omitted if the previous access to the bank was in the same page. To read a cache line from a 32 bit wide bank of DRAM in bank 2 the PMI will execute a single **RAS**Time, followed by four **CAS**Times, followed by a **Precharge**Time, as shown in figure 10.5.

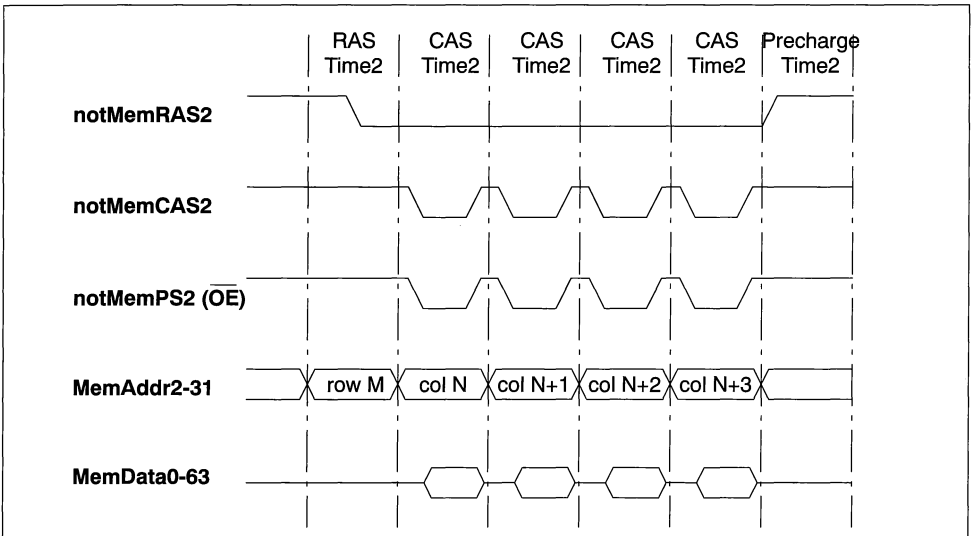


Figure 10.5 Page-mode access to DRAM - 32 bit interface cache refill from bank 2

For a 64 bit wide bank of DRAM the PMI will execute a single **RAS_{Time3}**, followed by two **CAS_{Time3}** followed by a **Precharge_{Time3}**, as shown in figure 10.6.

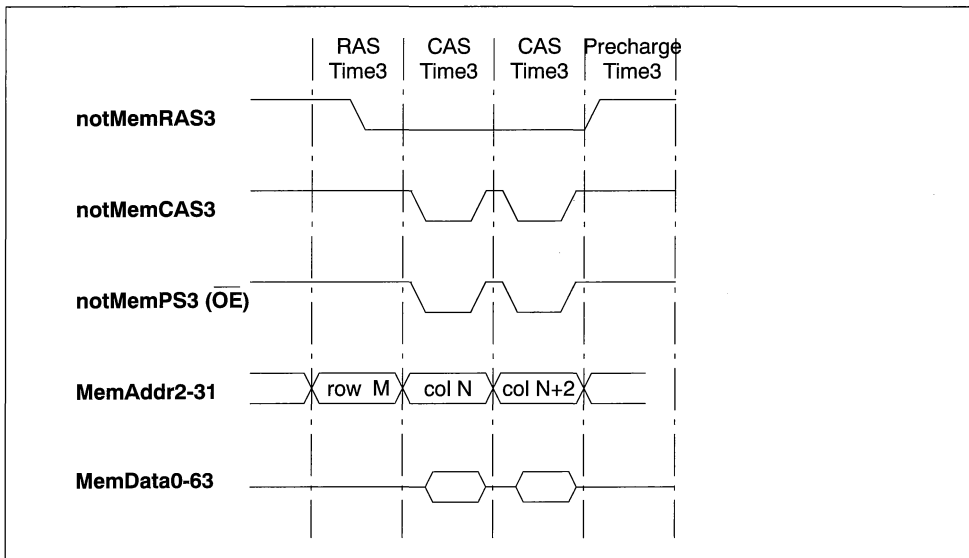


Figure 10.6 Page-mode access to DRAM – 64 bit interface cache refill from bank 3

The IMS T9000 is not limited to performing only cache-line refills in page-mode. As long as the row address remains constant, then the PMI will continually operate in page-mode. Memory contents can be fetched from any consecutive or non-consecutive column locations so long as the row address remains constant. Figure 10.7 shows an extended page mode cycle from DRAM.

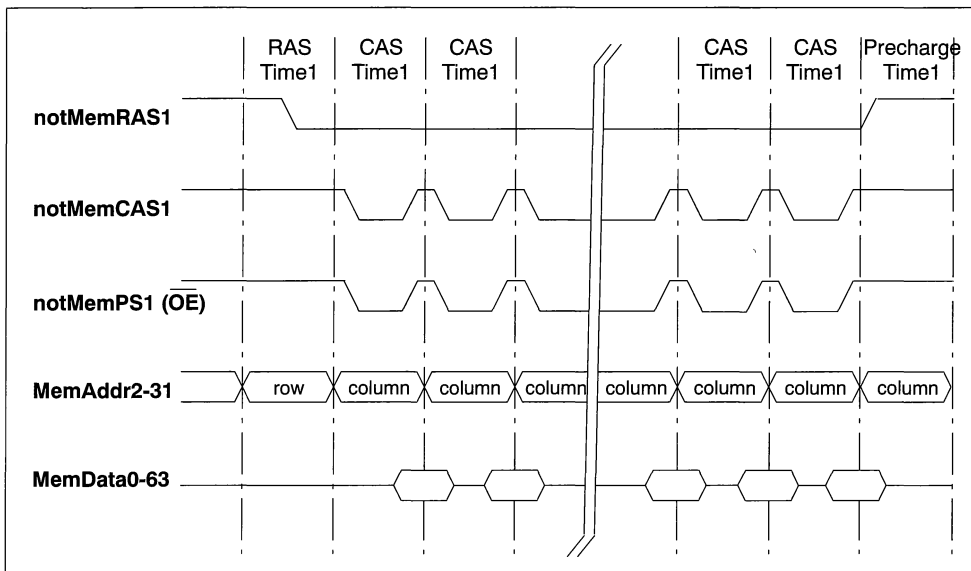


Figure 10.7 Extended page mode access from DRAM in bank 1

10.2.2 External non-DRAM cycles

The IMS T9000 interface does not explicitly distinguish between a bank which is programmed as dynamic memory and a bank which is not dynamic memory. This is to allow complete flexibility in the use of the strobes and the various timing parameters. The correct mode of access is determined by proper programming of the **TimingControl0-3** register parameters. Some of these parameters are inapplicable to a static memory bank and should be programmed to zero. Static memory cycles can be adequately defined by the **CAS_{Time0}** parameter. For a cache line read from static memory the **RAS_{Time}** is programmed to be zero and no **RAS** sub-cycle occurs. The PMI will execute four **CAS_{Time}** cycles for a cache line refill from a 32 bit wide bank of non-DRAM, as shown in figure 10.8.

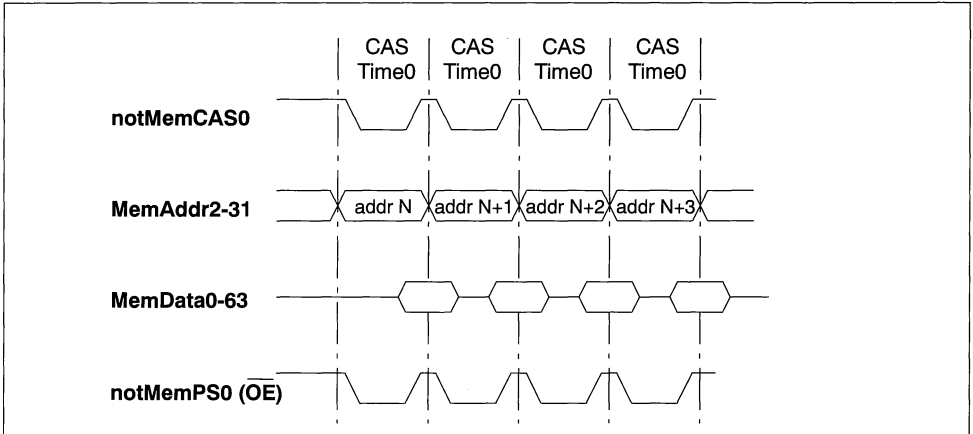


Figure 10.8 32 bit non-DRAM bank 0 cache refill

10.2.3 Bank switching

Precharge Time and **Bus Release Time** allow consecutive cycles to access different banks without the need for any external controlling logic. Figure 10.9 shows switching between SRAM in bank 0 and SRAM in bank 1. A **Bus Release Time** is inserted between the two accesses. The CAS, PS and Write strobes are inactive during this time, the RAS strobe is unaffected.

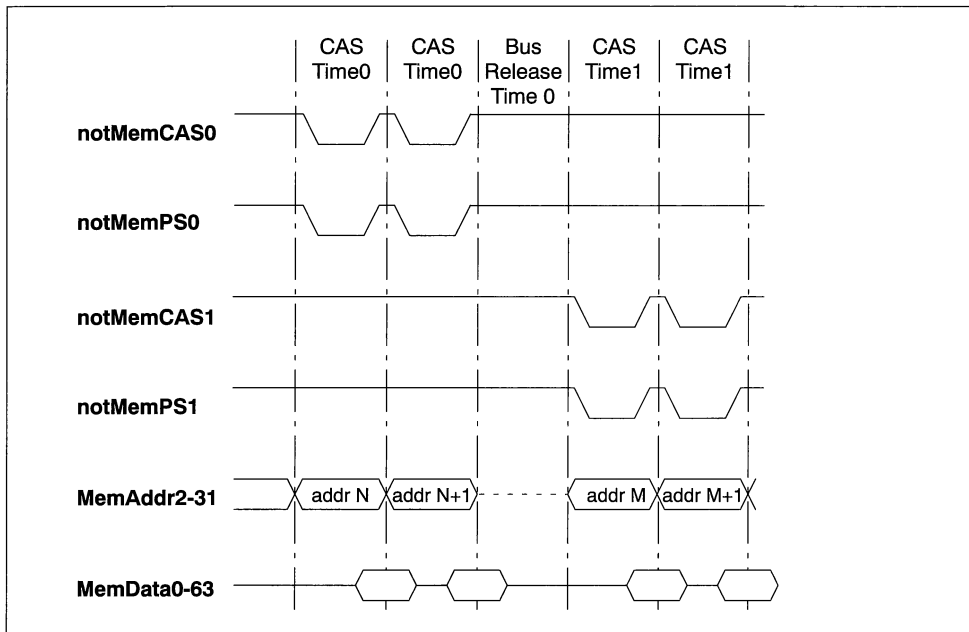


Figure 10.9 SRAM bank 0 to SRAM bank 1 with bus release time

Figure 10.10 shows switching between DRAM in banks 0 and 1. During **PrechargeTime0** the strobes for bank 0 are inactive and the strobes for bank 1 operate as defined by their configuration registers. Access is made to bank 1 whilst bank 0 is precharging. The example shown has the **PrechargeTime** programmed as 2 cycles.

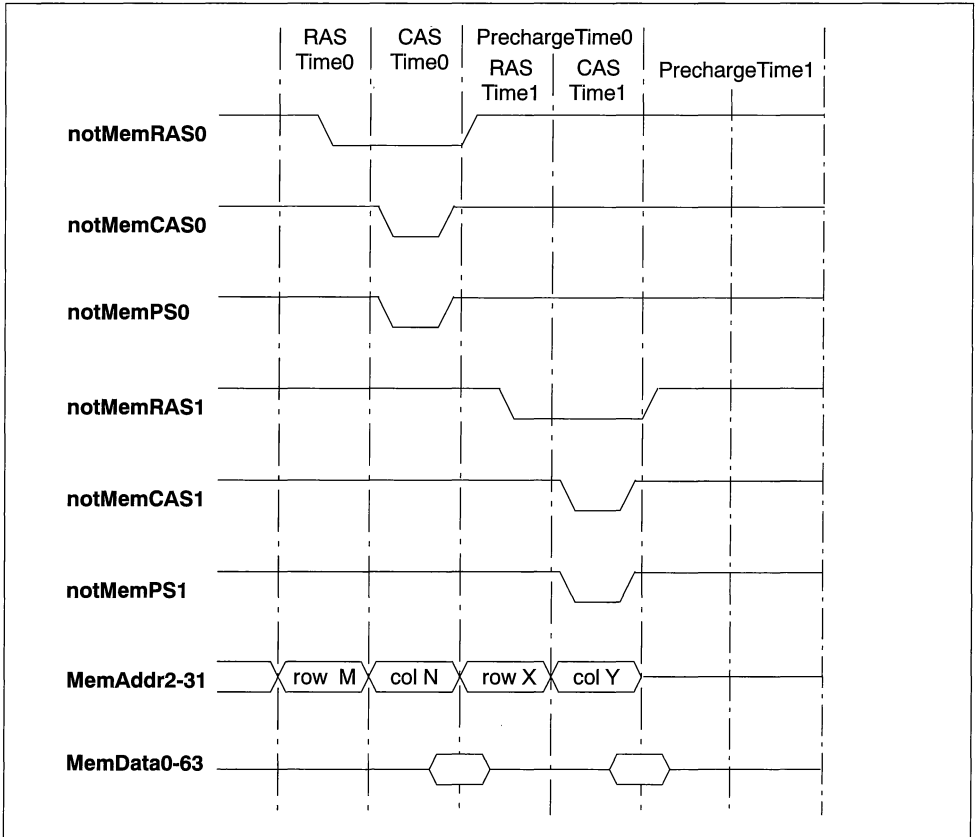


Figure 10.10 DRAM bank 0 to DRAM bank 1 switching, no bus release time

Figure 10.11 shows switching between DRAM in bank 1 and SRAM in bank 2. The example shown has the **PrechargeTime** for bank 1 programmed as 1 cycle and the bank 2 **CAS**Time as 2 cycles.

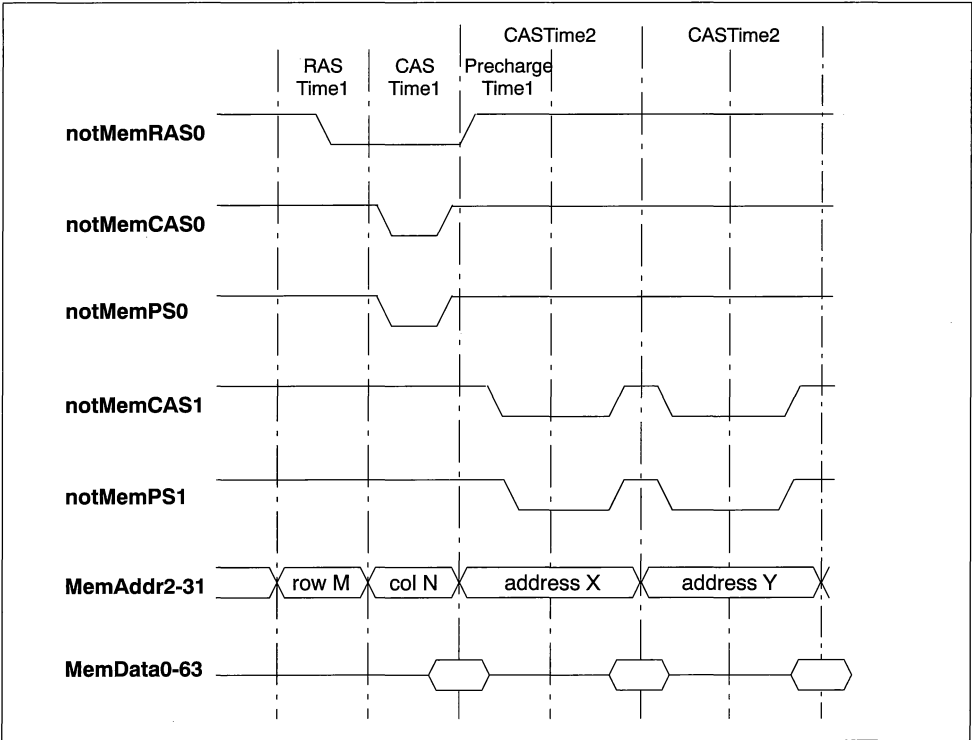


Figure 10.11 DRAM bank 0 to SRAM bank 1 switching, no bus release time

10.2.4 Cache refill cycles

The IMS T9000 instruction and data cache has four 32 bit words per line. Any read from external memory can become a cache-refill cycle. The refill engine examines the value of the **CacheMode** bit in the format control register (see page 165) for the selected bank. If the access is a read or write to a cacheable location the refill engine will initiate a cache-line refill cycle for all four words of a cache line. The refill engine generates the subsequent addresses of the cache line. The refill engine is defined to present the miss address first and then 'wraparound' the other addresses as detailed in section 9.3.4 of the Cache chapter.

64 bit port sizes are defined to be cacheable and any reads and writes always transfer a full 64 bit operand to and from external memory to the internal cache. No byte transfers take place and so no byte writes are active. During 64 bit reads the **notMemWrB0-3** strobes are all inactive. During 64 bit writes all of the **not-MemWrB0-3** strobes are active. For transfers from a 64 bit interface the full cache line is transferred in two external reads and **MemAddr2** is not used. The first external read is the miss address.

Table 10.5 defines the low order address bit activity during a cache line refill cycle from a 64 bit interface. **MemAddr2** is not used during 64 bit transfers.

First MemAddr3:2	Second MemAddr3:2
0:X	1:X
1:X	0:X

where, X denotes 0 or 1

Table 10.5 Low order address bit activity during a cache line refill cycle from a 64 bit interface

The first **MemAddr2-3** pair defines the miss address presented by the refill engine and transferred to the external address pins. The low order address bit activity during a cache line refill cycle from a non-64 bit interface is detailed in section 9.3.4 of the Cache chapter.

Cache refills from 8/16 bit ports

For port sizes of 8 or 16 bits the PMI generates the low order address bits (**A0-A1**) on the **notMemWrB2-3** strobes to assemble full 32 bit operands. This is repeated for each of the four words which make up the required cache line.

Write-back cycles

The IMS T9000 instruction and data cache is a write-back cache, thus main memory is only updated when a line from the IMS T9000 cache is selected for displacement and written to main memory. Each 32 bit word in the cache has a dirty bit associated with it. This dirty bit is set whenever any of the IMS T9000 internal functional units writes to an address location which is currently cached. When write-back cycles are performed, only those words in the line which have their associated dirty bit set, are written to main memory.

Wait states and cache-refill cycles

The IMS T9000 PMI will allow the **MemWait** pin to extend any of the cycles generated in the process of a cache-line refill. The **WaitEnable** bit must be set in the timing control registers (see page 171).

10.2.5 External DMA

The **MemReqIn** pin high causes the transputer to tri-state its memory buses so that an external DMA can access memory. **MemReqIn** is sampled during the last cycle of any external transfer. If the current cycle is a four-word cache line refill then the four words of the line are read in before the PMI responds. If the cycle is a refresh cycle it is allowed to terminate before the PMI responds. If the current cycle is a word transfer to a smaller port then the interface completes the transfer of the sub-words before responding. **MemGranted** is asserted during the first clock cycle after the current cycle terminates. The address bus, data bus and control signals are floated during this clock cycle.

Deassertion of **MemReqIn** is sampled during each **ProcClockOut** and **MemGranted** is deasserted during the next clock period.

Strobes are left floating during the DMA transfer. If DMA is active for longer than one programmed refresh interval then external logic is responsible for providing refresh.

DMA allows a bootstrap program to be loaded into external memory for execution after reset. If **MemReqIn** is held high during reset, **MemGranted** will be asserted before bootstrapping from external memory begins. The bootstrap sequence will continue when **MemReqIn** is deasserted. This will not prevent bootstrapping from the control links taking place.

The IMS T9000 processor can continue to execute out of the cache until a request to external memory is required. If a DMA operation is in progress then execution will stall until the DMA is complete. The **MemReqOut** indicates to external logic that IMS T9000 external bus cycles are pending and execution has stalled. External logic can use this information to continue or interrupt the DMA transfer in progress.

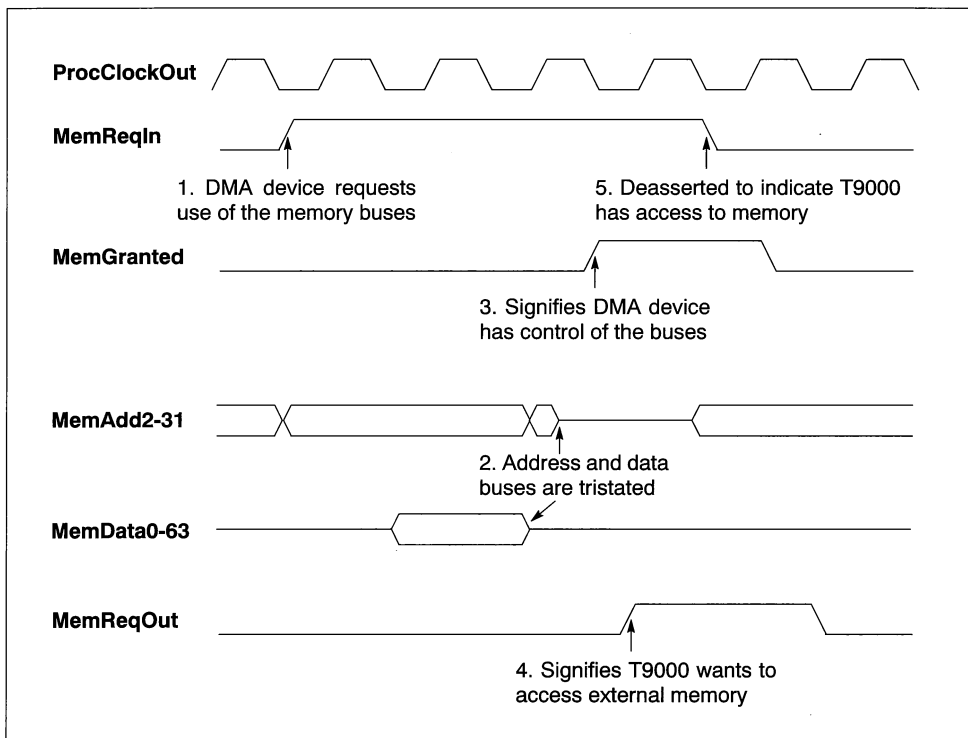


Figure 10.12 DMA

10.3 PMI configuration registers

The PMI (in common with a number of other sub-systems of the IMS T9000) is controlled via a separate configuration address space. The registers in this address space are accessed via the *ldconf* (load from configuration register) and *stconf* (store to configuration register) instructions, or via *CPeek* and *CPoke* command messages down **CLink0**. This section describes the functionality of the PMI to be controlled by the associated configuration registers.

Note, all INMOS reserved bits in the following tables must always be written with 0's.

The PMI configuration registers are divided into 2 sets. The *bank address* registers define the structure of the external address space and how it is allocated to the four banks and the *strobe timing* registers define the timing of the strobe edges for the four banks. The function of the registers is to eliminate external decode and timing logic.

The structural registers require external memory banks to be placed on 'natural' address boundaries for the memory banks. If a bank is of size 2^n words then the bank must be placed on a 2^n boundary. Operation of the PMI is undefined if this requirement is not met. External memory banks need not be placed on contiguous boundaries as long as the requirement above is met.

External memory banks can be overlapped as long as the banks are placed on natural boundaries. Overlapping banks are handled according to the prioritization table below, with bank 0 having the highest priority and the boot bank the lowest priority.

→ decreasing priority					External activity
Bank 0	Bank 1	Bank 2	Bank 3	Boot bank	
0	0	0	0	1	Access to boot bank
0	0	0	1	X	Access to bank 3
0	0	1	X	X	Access to bank 2
0	1	X	X	X	Access to bank 1
1	X	X	X	X	Access to bank 0

where,

- 0 represents address not in range
- 1 represents address in range
- X denotes 0 or 1

Table 10.6 Prioritization of banks

10.3.1 Bank address registers

The addresses of operands generated by IMS T9000 internal subsystems are analyzed by the PMI. It uses the values of the configuration registers to establish which bank the address is applicable to and the type of access. The incoming address and the bank address registers are compared. The bits that are not of interest are masked off by the mask address registers (**Mask0**, **Mask1**, **Mask2**, **Mask3**). This is performed in parallel for all four banks.

Address registers

The address registers (**Address0**, **Address1**, **Address2**, **Address3**) define the base address for each of the four banks. The base address must be word aligned.

Address0-3		#0202, #0205, #0208, #020B	Read/Write
Bit	Bit field	Function	
31:2	BaseAddress	Defines the word base for the bank address	
1:0		INMOS reserved	

Table 10.7 Bit fields in the **Address0-3** registers

Mask registers

The mask registers (**Mask0**, **Mask1**, **Mask2**, **Mask3**) define the bits in the address which should be compared to the address register for the appropriate bank.

1 in a given bit position indicates that the corresponding bits should be compared.

0 indicates they should be ignored.

If all bits which are to be compared are the same in the presented address and the address register for a bank then the address is a hit on the bank.

Mask0-3		#0203, #0206, #0209, #020C	Read/Write
Bit	Bit field	Function	
31:2	MaskField	Defines the mask for the bank address	
1:0		INMOS reserved	

Table 10.8 Bit fields in the **Mask0-3** registers

RAS bits registers

The RAS bits registers (**RASBits0**, **RASBits1**, **RASBits2**, **RASBits3**) define the bits in the address which should be compared to the last access to the same bank to determine whether a page hit has occurred. The register contents are only used if the **RASTime** in the **TimingControl** register is programmed to be non-zero.

1 in a given bit position indicates that the corresponding bits should be compared.

0 indicates they should be ignored.

If all bits which are to be compared are the same in the presented address as in the previous access, then the address is a page hit and a **RAS** cycle will not be generated.

RASBits0-3		#020E, #020F, #0210, #0211	Read/Write
Bit	Bit field	Function	
31:2	RASField	Defines the RAS bits	
1:0		INMOS reserved	

Table 10.9 Bit fields in the **RASBits0-3** registers

Format control registers

The format control registers (**FormatControl0**, **FormatControl1**, **FormatControl2**, **FormatControl3**) control general aspects of operation of each bank of the PMI.

FormatControl0-3		#0204, #0207, #020A, #020D	Read/Write
Bit	Bit field	Function	
31:28	ShiftAmount	Right shift for the on-chip multiplexing for the bank address	
4:3	PortSize	Bit width of the bank (8, 16, 32 or 64 bits)	
2	CacheMode	Cacheability status of addresses in the bank	
1:0, 27:5		INMOS reserved	

Table 10.10 Bit fields in the **FormatControl0-3** registers

The **ShiftAmount** is defined as the amount of right shift to be applied to the external address field, for the duration of the **RASTime**, see table 10.11. The shift amount is only active during the **RASTime**.

	ShiftAmount (bits 31:28 of format control register)															
	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
MemAddr2	a2	a3	a4	a5	a6	a7	a8	a9	a10	a11	a12	a13	a14	a15	a16	a17
MemAddr3	a3	a4	a5	a6	a7	a8	a9	a10	a11	a12	a13	a14	a15	a16	a17	a18
MemAddr4	a4	a5	a6	a7	a8	a9	a10	a11	a12	a13	a14	a15	a16	a17	a18	a19
MemAddr5	a5	a6	a7	a8	a9	a10	a11	a12	a13	a14	a15	a16	a17	a18	a19	a20
MemAddr6	a6	a7	a8	a9	a10	a11	a12	a13	a14	a15	a16	a17	a18	a19	a20	a21
MemAddr7	a7	a8	a9	a10	a11	a12	a13	a14	a15	a16	a17	a18	a19	a20	a21	a22
MemAddr8	a8	a9	a10	a11	a12	a13	a14	a15	a16	a17	a18	a19	a20	a21	a22	a23
MemAddr9	a9	a10	a11	a12	a13	a14	a15	a16	a17	a18	a19	a20	a21	a22	a23	a24
MemAddr10	a10	a11	a12	a13	a14	a15	a16	a17	a18	a19	a20	a21	a22	a23	a24	a25
MemAddr11	a11	a12	a13	a14	a15	a16	a17	a18	a19	a20	a21	a22	a23	a24	a25	a26
MemAddr12	a12	a13	a14	a15	a16	a17	a18	a19	a20	a21	a22	a23	a24	a25	a26	a27
MemAddr13	a13	a14	a15	a16	a17	a18	a19	a20	a21	a22	a23	a24	a25	a26	a27	a28
MemAddr14	a14	a15	a16	a17	a18	a19	a20	a21	a22	a23	a24	a25	a26	a27	a28	a29
MemAddr15	a15	a16	a17	a18	a19	a20	a21	a22	a23	a24	a25	a26	a27	a28	a29	a30
MemAddr16	a16	a17	a18	a19	a20	a21	a22	a23	a24	a25	a26	a27	a28	a29	a30	a31
MemAddr17	a17	a18	a19	a20	a21	a22	a23	a24	a25	a26	a27	a28	a29	a30	a31	U
MemAddr18	a18	a19	a20	a21	a22	a23	a24	a25	a26	a27	a28	a29	a30	a31	U	U
MemAddr19	a19	a20	a21	a22	a23	a24	a25	a26	a27	a28	a29	a30	a31	U	U	U
MemAddr20	a20	a21	a22	a23	a24	a25	a26	a27	a28	a29	a30	a31	U	U	U	U
MemAddr21	a21	a22	a23	a24	a25	a26	a27	a28	a29	a30	a31	U	U	U	U	U
MemAddr22	a22	a23	a24	a25	a26	a27	a28	a29	a30	a31	U	U	U	U	U	U
MemAddr23	a23	a24	a25	a26	a27	a28	a29	a30	a31	U	U	U	U	U	U	U
MemAddr24	a24	a25	a26	a27	a28	a29	a30	a31	U	U	U	U	U	U	U	U
MemAddr25	a25	a26	a27	a28	a29	a30	a31	U	U	U	U	U	U	U	U	U
MemAddr26	a26	a27	a28	a29	a30	a31	U	U	U	U	U	U	U	U	U	U
MemAddr27	a27	a28	a29	a30	a31	U	U	U	U	U	U	U	U	U	U	U
MemAddr28	a28	a29	a30	a31	U	U	U	U	U	U	U	U	U	U	U	U
MemAddr29	a29	a30	a31	U	U	U	U	U	U	U	U	U	U	U	U	U
MemAddr30	a30	a31	U	U	U	U	U	U	U	U	U	U	U	U	U	U
MemAddr31	a31	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

where, U denotes unused pins – these are undefined

Table 10.11 Mapping of internal address bits (a2-a31) onto external address pins (**MemAddr2-31**)

The **PortSize** (bits 4:3 of format control registers) defines the size of the external port that occupies the selected bank. The coding of the bits are defined in table 10.12. The **PortSize** parameter is used by the byte-alignment network to assemble/disassemble data bytes to transfer arbitrary-sized operands to arbitrary-sized ports.

PortSize	Programmed bus widths
00	64 bits
01	32 bits
10	16 bits
11	8 bits

Table 10.12 **PortSize**

The **CacheMode** bit (bit 2 of format control registers) field defines if the bank is occupied by devices whose contents can be transferred to the IMS T9000 internal cache. Note, any bank which is programmed to have 64 bit memory is defined as a cacheable area.

CacheMode	Cacheability
0	Non-cacheable
1	Cacheable

Table 10.13 **CacheMode**

Figure 10.13 illustrates programming of the configuration registers for given bank configurations. Banks 0 and 1 are shown containing an external bus width of 32 bits, banks 2 and 3 with an external port size of 64 bits. All the banks have a programmed **RASTime**, except bank 4 which has **RASTime** set to zero. In the example shown in the figure the base addresses for each of the four banks are as follows:

Bank	Physical address
0	C3C00000 – C3FFFFFF
1	B0300000 – B03FFFFFF
2	50E00000 – 50FFFFFF
3	0E1F8000 – 0E1FFFFFF

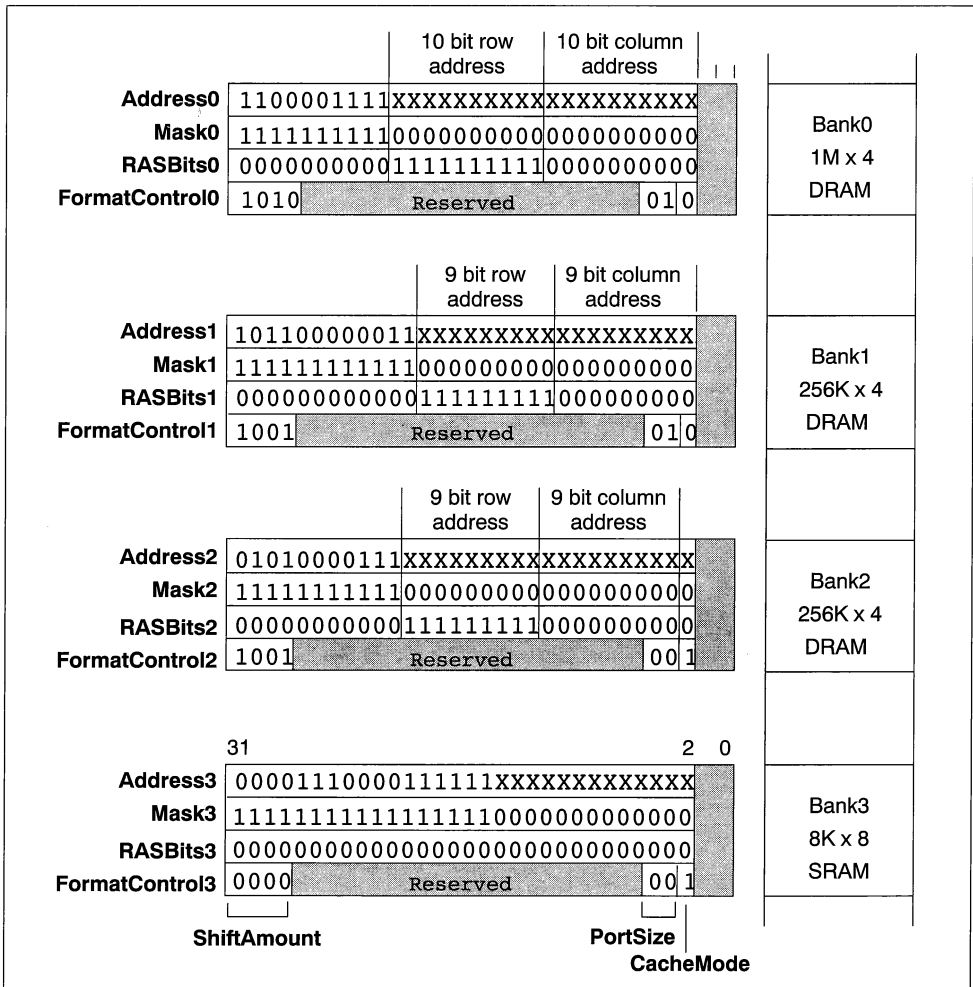


Figure 10.13 Programming page configuration registers

DoPMIConfigured register

This register is write only. Once the configuration registers have been set up, a write to the **DoPMIConfigured** register initializes the PMI. External memory accesses can then be performed.

Error address register

If an external memory error occurs (i.e. an attempt is made to access non-existent memory) which is detected by the PMI, the processor is halted and the address which caused the error is saved in the error address (**ErrorAddress**) register. Refer to section 10.4 for details of PMI errors and their effects.

ErrorAddress #0213		Read only
Bit	Bit field	Function
31:2	Error address	"Illegal" address which caused an external memory error.
1:0		INMOS reserved

Table 10.14 Bit fields in the **ErrorAddress** register

10.3.2 Strobe timing registers

The PMI constructs control waveforms with the required timing in the appropriate bank from the contents of the timing control registers. The internal pipeline structure of the IMS T9000 allows internally pending cycles to be analyzed while the bus is currently in use. The bus control logic can construct the required timing and control waveforms from information about the current bus cycle and the next pending cycle.

Note: A *cycle* is one processor clock cycle and a *phase* is one quarter of the duration of one processor clock cycle.

Strobe registers

The RAS strobe registers (**RASStrobe0**, **RASStrobe1**, **RASStrobe2**, **RASStrobe3**), CAS strobe registers (**CASStrobe0**, **CASStrobe1**, **CASStrobe2**, **CASStrobe3**), programmable strobe registers (**ProgStrobe0**, **ProgStrobe1**, **ProgStrobe2**, **ProgStrobe3**) and the write strobe registers (**WriteStrobe0**, **WriteStrobe1**, **WriteStrobe2**, **WriteStrobe3**) all have a common format, as given in table 10.15. The falling (E1) and rising (E2) edges of a waveform are defined to occur during the **CASTime**. During other sub-cycles the programmable strobe pins are held in the inactive state. Figure 10.14 illustrates the strobe activity within a memory cycle.

RASStrobe0-3	#0401, #0402, #0403, #0404	Read/Write	
CASStrobe0-3	#0405, #0406, #0407, #0408	Read/Write	
ProgStrobe0-3	#0409, #040A, #040B, #040C	Read/Write	
WriteStrobe0-3	#040D, #040E, #040F, #0410	Read/Write	
Bit	Bit field	Function	Units
31:26	E1Time	Location of falling edge from CASTime start	phases
25:20	E2Time	Location of rising edge from CASTime start	phases
19:18	ActiveCode	Cycle type in which strobe is active	—
17:0		INMOS reserved	—

Table 10.15 Bit fields in the **RASStrobe0-3**, **CASStrobe0-3**, **ProgStrobe0-3** and **WriteStrobe0-3** registers

ActiveCode (bits 19:18 of the strobe registers) determines the type of cycle (read or write) during which the strobe will be active. The coding of these bits is indicated below.

ActiveCode	Bus activity
00	Inactive
01	Active during read only
10	Active during write only
11	Active during read and write

Table 10.16 **ActiveCode**

The timing programmed in the **WriteStrobe** register for a bank is used for all four byte-write strobes for writes in that bank.

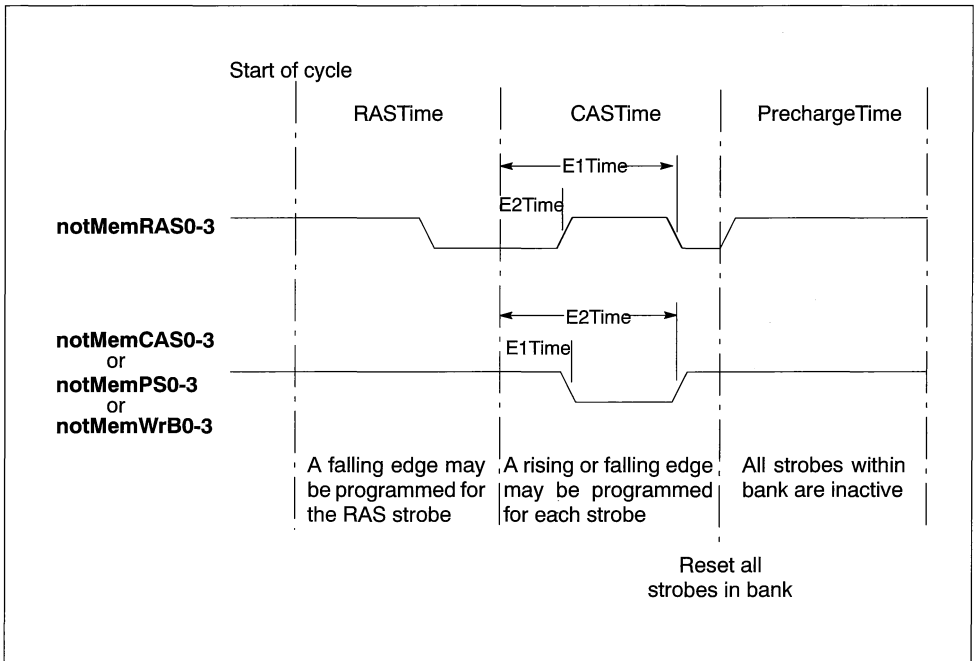


Figure 10.14 Strobe activity within a memory cycle

The programmability of the strobe edges allows misplacement of the edges within the **CASTime**. Special programming situations and the machine response are described below.

- The position of the rising edge is programmed to occur before the falling edge: In this case the rising edge is omitted and a falling edge occurs at the programmed position. The strobe stays low for the duration of the **CASTime**.
- The position of the falling edge is programmed to lie outside the sub-cycle: In this case the strobe stays inactive and no transitions occur.
- The position of the rising edge is defined to lie outside the sub-cycle: In this case the strobe makes the falling transition and stays low for the duration of the **CASTime**.
- The position of both are programmed to lie outside the sub-cycle: In this case the strobe remains inactive and no transitions occur.
- The position of both edges are defined to be coincident within the sub-cycle: no edge transitions occur and the strobe stays inactive.

If DRAM is present in a bank the **E2Time** for the associated **RASStrobe** register is typically programmed to be shorter than the **E1Time** so that the RAS strobe falls during the **RASTime** of the cycle and rises again during the **CASTime** of the cycle.

Timing control registers

The timing control registers (**TimingControl0**, **TimingControl1**, **TimingControl2**, **TimingControl3**) define for each bank timing parameters for the peripheral devices allocated to that memory bank. The parameters defined by the register are shown in table 10.17.

TimingControl0-3		#0411, #0412,#0413,#0414	Read/Write
Bit	Bit field	Function	Units
31:28	RASTime	Duration of RAS sub-cycle	cycles
27:22	RASEdgeTime	Delay from start of RAS sub-cycle to falling edge of RAS strobe	phases
21:18	CASTime	Duration of CAS sub-cycle	cycles
17:14	PrechargeTime	Duration of precharge time	cycles
13:10	BusReleaseTime	Duration of bus release time	cycles
9	WaitEnable	Enables the MemWait pin	—
8:0		INMOS reserved	—

Table 10.17 Bit fields in the **TimingControl0-3** registers

RASTime sets the length of the RAS sub-cycle. If this is programmed to zero then no RAS sub-cycle will occur.

RASEdgeTime sets the delay from the start of the RAS sub-cycle to when the RAS strobe goes low. This is only required if **RASTime** is programmed to be non-zero.

The **WaitEnable** bit enables or disables the use of the external **MemWait** pin. The PMI is designed to synthesize the required waveforms from the parameters in the **TimingControl0-3** registers. However, external control can be provided if required. If the **WaitEnable** bit is disabled (set to 0) then the external cycles are controlled exclusively by the **TimingControl0-3** registers. If the **WaitEnable** bit is enabled (set to 1) then **MemWait** is defined to operate as follows:

- The **MemWait** pin is sampled during every clock cycle of the **RASTime** and the **CASTime** only.
- The action of **MemWait** is to suspend the state of the cycle counters and the level of the strobes. This is maintained as long as **MemWait** is asserted.
- When **MemWait** is deasserted counting continues as defined by the **TimingControl0-3** registers.

Figure 10.15 shows the effect of **MemWait** on the cycle and strobe definition. Figure 10.16 gives an example of the programming of the strobe timing registers for bank 1.

Note: There is no defined relationship between **ProcClockOut** and **MemWait** or between **ProcClockOut** and the strobe signals.

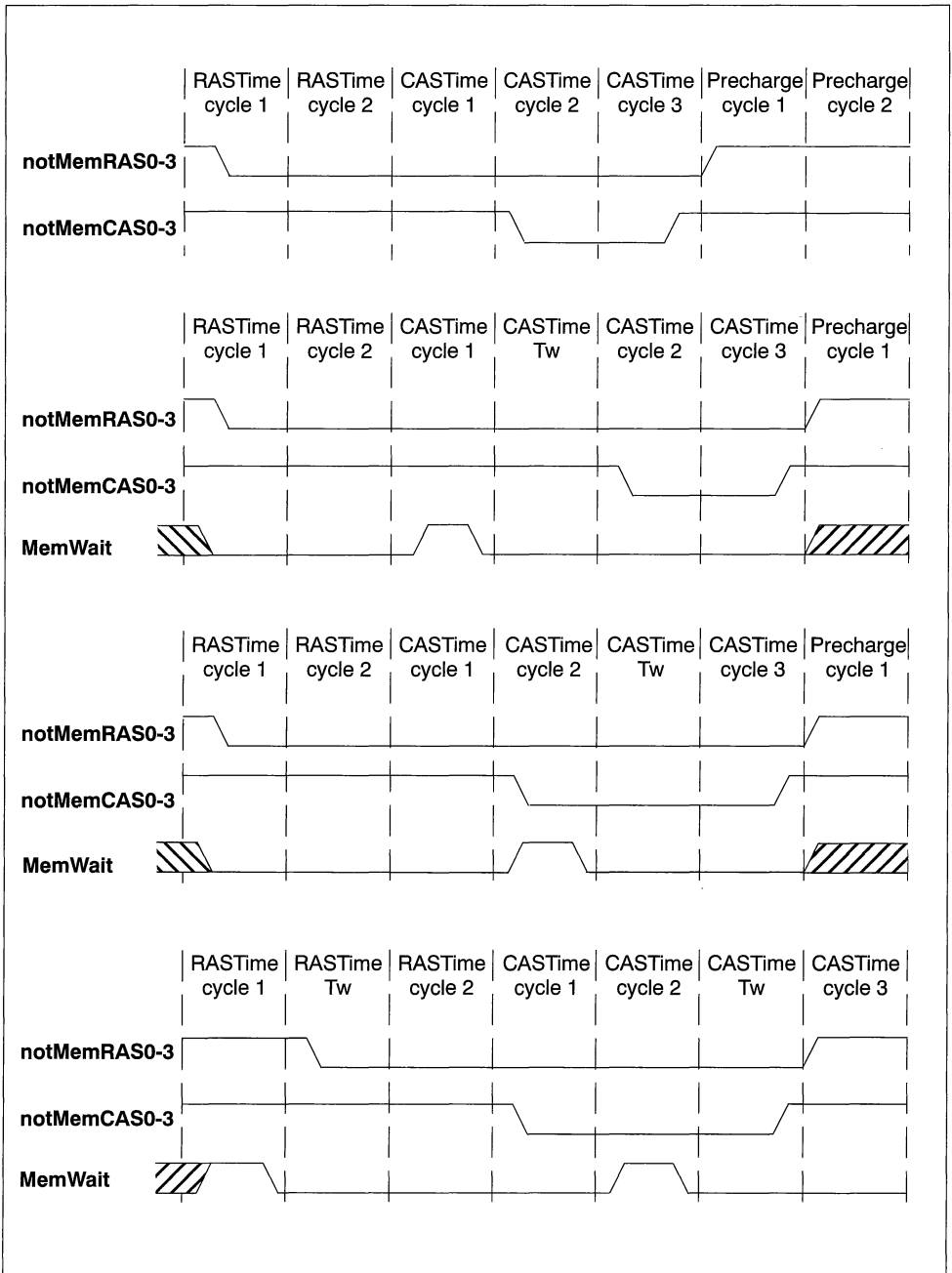


Figure 10.15 MemWait

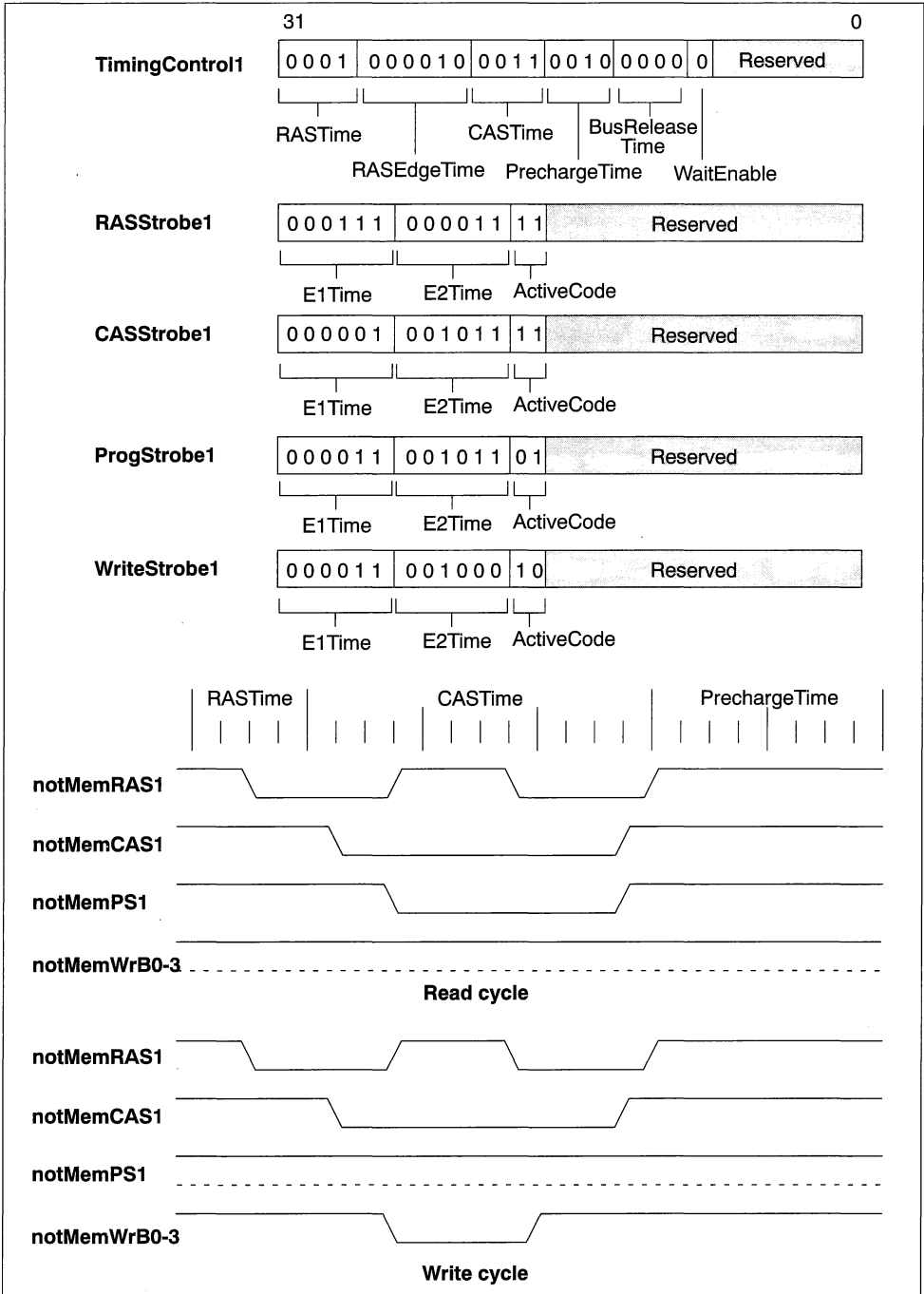


Figure 10.16 Example programming of the strobe timing registers for bank 1

Refresh control register

The refresh control register (**RefreshControl**) specifies the banks which require refreshing and the interval between successive refreshes. The refresh timing is also programmed in this register, and is the same for all banks.

The PMI ensures that **CAS** and **RAS** are both high for the required time before every refresh cycle by inserting a **PrechargeTime** in the last bank being accessed and ensuring all **PrechargeTimes** are complete.

The **CASStrobe** is taken low at the beginning of the refresh time. The position of the **RAS** falling edge (**RASedge**) and the time before **RAS** and **CAS** can be taken high again (**RefreshTime**) are programmed. Each of these actions occurs in sequence for each bank. A cycle is inserted between each bank in order to spread current peaks. If no DRAM has been programmed for a bank then no transitions occur on the **RAS** or **CAS** strobcs.

When the **CyclePendingMask** bit is set the **MemReqOut** pin can be used to indicate that a refresh cycle is pending. When this bit is not set the **MemReqOut** pin is set by the processor either to indicate to a DMA device that it wants to access memory or that it wants to perform a refresh cycle.

The **RefreshControl** register is loaded during the configuration phase and if the **Refresh Interval** is zero, then no refreshes will take place. The register bit fields are allocated the following functions.

RefreshControl		#0415	Read/Write
Bit	Bit field	Function	Units
31:22	RefreshInterval	Defines DRAM refresh interval	cycles
21:18	RefreshTime	Refresh time	cycles
17:12	RASedge	Refresh RAS falling edge	phases
11:8	DRAM3:0	Defines which banks require refresh	—
7	CyclePendingMask	Masks the memory access cycle. When this bit is set to 1 the MemReqOut pin is used to indicate that a refresh cycle is pending.	—
6:0		INMOS reserved	

Table 10.18 Bit fields in the **RefreshControl** register

Figure 10.17 illustrates programming of the refresh control register. The example shows DRAM programmed in banks 0, 1 and 3, no DRAM programmed in bank 2. Each bank is programmed with a **RefreshTime** of 4 cycles and a **RefreshInterval** of 400 cycles. After a refresh, a **PrechargeTime** is introduced after each bank has completed in turn. In the example shown the **PrechargeTime** for each bank is programmed (in the strobe timing registers) as 1 cycle. After refresh has completed in bank 3, further accesses may proceed for all banks once any precharge times are complete.

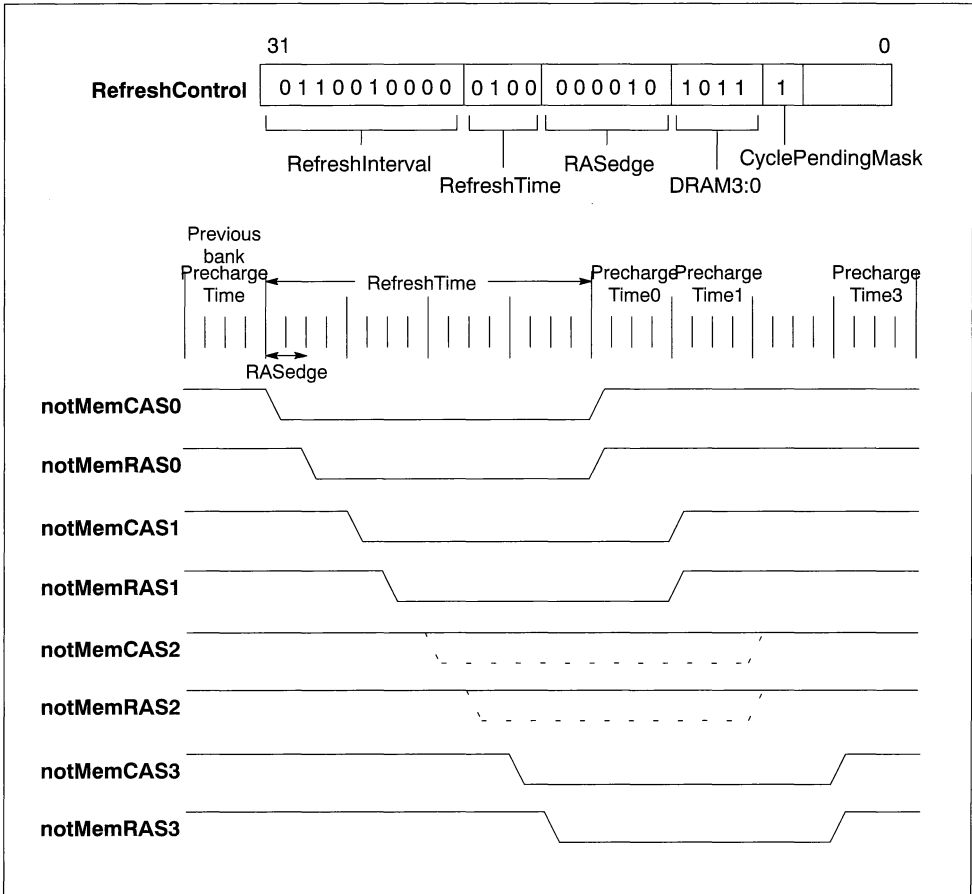


Figure 10.17 Programming of the refresh control register

Remap boot bank register

The remap boot bank register (**RemapBootBank**) can be used to connect the boot bank (fifth bank with fixed decode and timing parameters) to another bank, for example as part of a 32 bit ROM configuration, refer to section 10.6.3 for full details. This register is write only.

10.3.3 PMI write lock register

The PMI write lock register (**PMIWriteLock**) provides the PMI configuration registers protection from configuration writes by the CPU, so that the IMS T9000 is guaranteed to behave sensibly with respect to the rest of the network regardless of program behavior. It prevents a program from stopping communications by disabling memory.

Reset clears this register.

The PMI bank address subsystem and PMI strobe timing subsystem both have a write lock at the same local address which can be set by a single write to the combined (PMI bank address and PMI strobe timing) subsystem at address #0600.

PMIWriteLock		#0600	Read/Write
Bit	Bit field	Function	
0	PMIWriteLock	When set to 1 it inhibits modification of the PMI registers by the CPU.	

Table 10.19 Bit fields in the **PMIWriteLock** register

10.4 PMI errors

10.4.1 Errors detected by the PMI

The PMI detects illegal attempted memory accesses. Illegal accesses include: accesses to addresses which are not in the range of any of the memory banks; and non-device accesses to banks defined by the configuration registers as device banks.

The illegal access may be ignored or may be signalled as an error either by the CPU or the PMI. It is ignored if the access was a speculative access initiated by the CPU and not used. Other illegal accesses by the CPU will be signalled by the CPU or by the PMI. If the access was not initiated by the CPU then an error is signalled by the PMI.

PMI errors signalled by the CPU

An illegal memory access signalled by the CPU causes the processor to halt with a reason code of 'external memory error'. The address which caused the error is saved in the **EmiBadAddress** CPU configuration register.

PMI errors signalled by the PMI

Illegal memory accesses not signalled by the processor, such as illegal VCP memory accesses, are signalled by the PMI and cause the processor to halt with a reason code of 'halted'. The address which caused the error is saved in the **ErrorAddress** PMI configuration register.

The **EmiBadAddress** and **ErrorAddress** registers can be read to aid debugging. They must be read after resetting but before rebooting the IMS T9000, by the control link *CPeek* command, as the values of these registers become undefined after rebooting.

10.5 Initialization of the PMI

After power up and reset the PMI is disabled. The IMS T9000 transputer must be bootstrapped either from a ROM device (such as an EPROM or a Flash EPROM), or down control link (**CLink0**) from a host transputer (or a computer with a link adaptor) with the bootstrap code executed in internal memory. The PMI configuration registers are set up by the bootstrap code to match the external hardware, and the PMI is enabled.

10.5.1 Bootspace allocation

The IMS T9000 includes support for a fifth bank (boot bank) of external memory which is not user programmable. It has fixed decode and timing parameters. The function of this bank is to provide a configuration/bootstrap area of external memory. All parameters for this bank are hardwired into the PMI. The port size of this bank is hardwired to be a byte wide interface. Reads for configuration words are assembled from bytes. All addresses in the bootspace are regarded as legal. The boot bank address space covers 16 Mbytes down from the top of store. Note that the address space is signed, and as such the bottom of memory occurs at #80000000.

Parameter name	Function	Address
TopOfStore	Defines top of bootspace	#7FFFFFFF
BootBankBase	Defines bottom of bootspace	#7F000000

Table 10.20 Bootspace

This is a large block of external store dedicated to bootspace and therefore once the PMI is configured the user is able to reclaim this space by mapping the boot bank onto bank 0, see section 10.6.3 on re-mapping the boot bank.

10.5.2 The boot sequence

After power up and reset the control unit will perform two memory *Peeks* from the top of store fetching first the **BootWptr** followed by the **BootIptr**, refer to the memory map below.

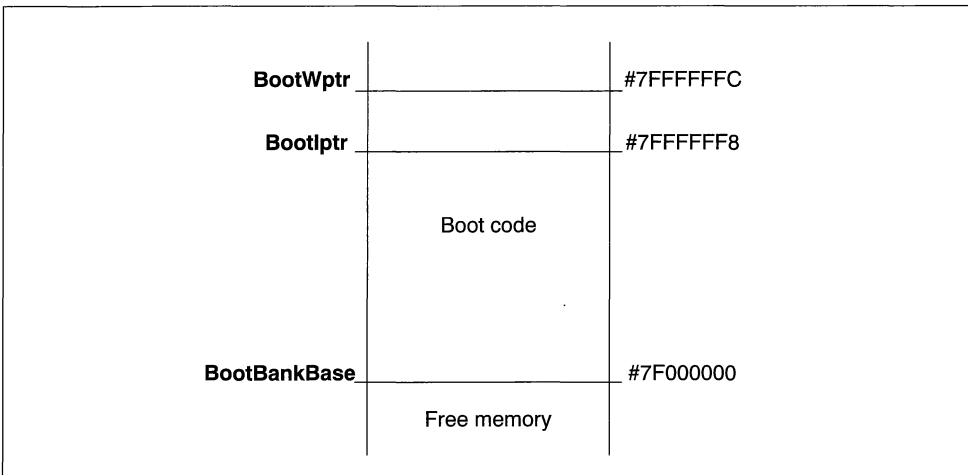


Figure 10.18 Memory map

The control unit then performs *stconf* (store to configuration register) instructions into the CPU configuration space and allows the CPU to execute the first instructions. The first instructions allow configuration of the various functional parts of the IMS T9000 to begin their relevant actions.

Once the configuration registers have been setup, writing to configuration register **DoPMIConfigured** initializes the PMI and refresh timing commences as programmed. External memory accesses can then be performed.

Before **DoPMIConfigured** is set only the boot bank can be accessed. If an access is attempted to one of the four programmable banks an error occurs. Once **DoPMIConfigured** is set an error will occur if an access is made to an address outside the four programmable banks. The 'offending' address following an external memory error can be read from the **ErrorAddress** register.

10.5.3 Bootspace timing

The timing for accesses into the bootspace is not user programmable. The timing values used are defined in figure 10.19. The timings have been deliberately relaxed to cover a wide range of ROM and T9000 speeds, typically 30 MHz to 60 MHz.

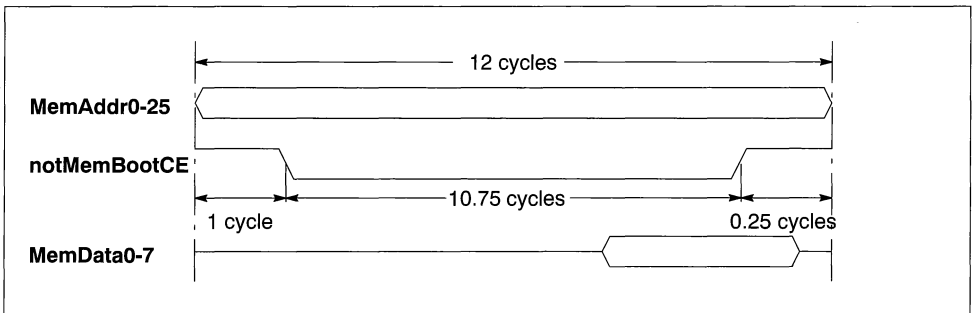


Figure 10.19 ROM timing for 8 bit bus

10.6 Booting from ROM

The IMS T9000 transputer can be bootstrapped from a ROM device, such as an EPROM or a Flash EPROM, or down control link (**CLink0**) from a host transputer or a computer with a link adaptor. For details on booting down **CLink0** refer to chapter 8 on the control system. In order to boot from ROM the **StartFromROM** pin must be held high.

10.6.1 Booting from EPROM

EPROMs are normally byte wide devices that contain a programmable non-volatile memory array. These can be programmed to bootstrap the IMS T9000 system using the relevant software tools to generate the programming data. Figure 10.20 gives an example of booting from EPROM.

Prior to bootstrap the IMS T9000 transputer is not able to read an EPROM as the PMI strobe timings are not yet defined. The boot bank provides slow access to configuration/ bootstrap code stored in ROM. The **notMemBootCE** pin is used to access the boot bank of external memory. The PMI configuration registers can then be set up by the ROM code. The **StartFromROM** pin must be held high during bootstrap when bootstrapping from ROM.

The boot bank address space can also be used to access code/data which is not bootstrap code if required.

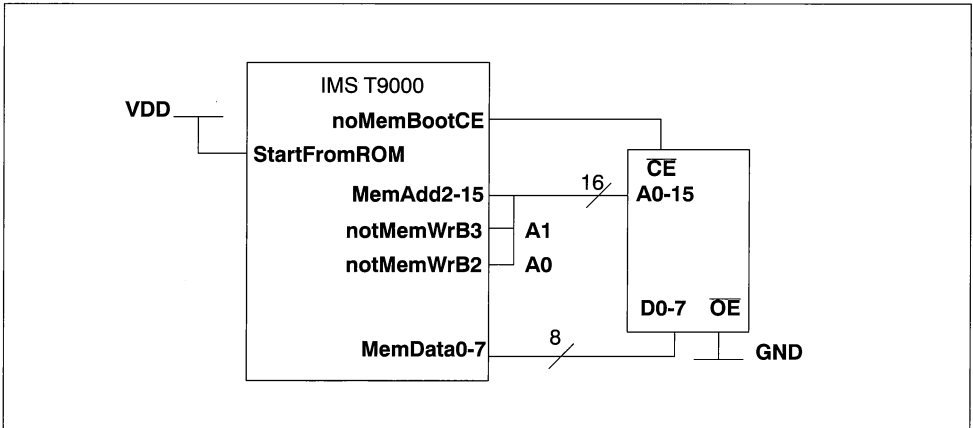


Figure 10.20 Adding EPROM to the IMS T9000 transputer

The location of the EPROM within the external memory space is at the top 16 Mbytes of the address space. The IMS T9000 reads a **Wptr** and an **lptr** from two fixed locations near the top of memory, see figure 10.18.

A high priority L-process then starts executing with this **Wptr** and **lptr**. Configuration of the IMS T9000 can be performed by a series of *stconf* instructions.

Note that workspace is available even before the PMI is configured since after power-on-reset the cache operates as internal RAM.

10.6.2 Booting from Flash EPROM

A Flash EPROM is a non-volatile electrically erasable ROM. Figure 10.21 shows an example system using Flash EPROM. Flash EPROM can be used to boot a system, such as a prototype system, as it enables changes to be made to the bootstrap code.

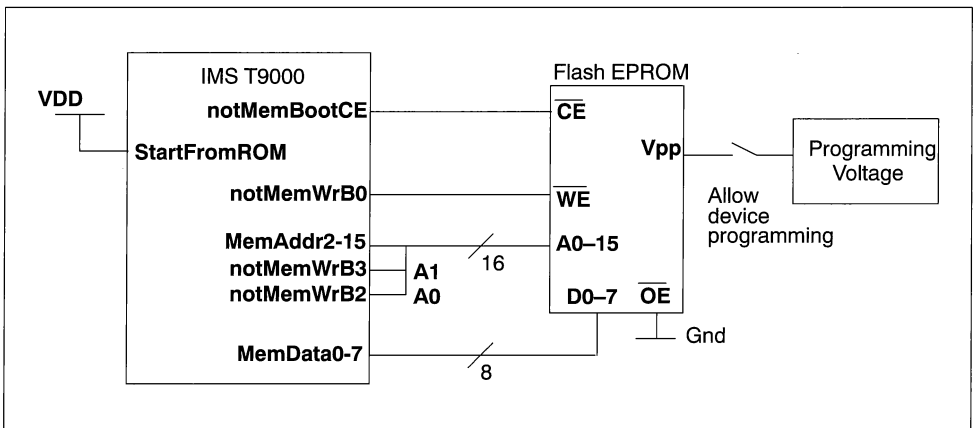


Figure 10.21 Adding Flash EPROM to the IMS T9000 transputer

At bootstrap the Flash EPROM resides in the boot bank address space. The **RemapBootBank** register can be used to make the Flash EPROM appear to reside in one of the four programmable external memory banks. The PMI registers can then be accessed via the *ldconf* and *stconf* instructions, or via *CPeek* and

CPoke command messages received along control link **CLink0**, to program the timing parameters to enable configuration of the PMI.

The EPROM, which would now appear to be in the address space of one of the four memory banks can be read, written or erased. During this time, the **notMemCAS0** strobe for the bank 0 in which the Flash EPROM has been mapped, is used to drive the **notMemBootCE** pin. Therefore, no external logic is required, provided the Flash EPROM timing parameters can be met.

Once programming is complete, the configuration registers can be reset and on bootstrap, the new data stored in the Flash EPROM will be used to configure the transputer, and/or execute a new program.

10.6.3 Re-mapping the boot bank

The boot bank is read only, in order to write to the boot bank it must be re-mapped.

The boot bank is connected using **notMemBootCE** at boot time. The **RemapBootBank** register can be used to connect the boot ROM to another bank, for example as part of a 32 bit ROM configuration. The ROM should be connected to bank0. Bank0 can then be remapped so that the **notMemCAS0** signal can be routed internally to become what was **notMemBootCE**. This is achieved by performing a *stconf* instruction to the **RemapBootBank** configuration register. Bank0 addresses must be configured to coincide with the appropriate boot bank addresses.

It should be noted that overlaying the boot bank space with any other bank does not cause any internal error, therefore care should be taken when connecting address and strobe pins.

RemapBootBank register	Bank 0	Boot bank	External activity
X	0	0	No hits, no external activity
0	0	1	Normal hit to boot bank, notMemBootCE active
0	1	X	Priority access to bank 0, bank 0 strobes active
1	0	1	Remapped access, bank 0 strobes active, CAS timing for bank 0 copied onto notMemBootCE
1	1	X	Remapped access, bank 0 strobes active, CAS timing for bank 0 copied onto notMemBootCE

where,

0 represents address not in range

1 represents address in range

X denotes 0 or 1

Table 10.21 Remapping boot bank prioritization

10.7 PMI AC timing characteristics

The following AC timing characteristics are based on simulations of the 50 MHz version of the IMS T9000 chip, and may change when full characterization is completed. The simulations were run under a loading of 75 pF unless otherwise stated.

The IMS T9000 PMI has three types of pins; address, data and strobe pins. The falling and rising edges of the strobe pins are programmable through the configuration registers. The following specification is given either as an absolute timing value or as a skew (Δt_{SPEC}) from the nominal programmed value (t_n). The relationship between the parameter (t_{SPEC}) as specified on the timing diagrams, the skew and the programmed value is as follows:

$$t_{SPEC} = t_n + \Delta t_{SPEC}$$

The following section gives the data setup and hold times for read, write and wait cycles. It also specifies the maximum skew from the nominal programmed values between the address and strobe pins.

The table below gives the PMI AC specifications and example timing diagrams follow.

Note, a minimum timing value with a negative value indicates that the transition of the second signal can occur before the first signal. For example the minimum timing specification for address setup to strobe valid is -6 ns, i.e. the strobe can be valid 6 ns before address setup.

Symbol	No.	Parameter	Min	Max	Units	Notes
Δt_{AVSV}	1	Address setup to strobe valid	-6	+2	ns	1,2
Δt_{SVAV}	2	Address hold after strobe valid	-2	+6	ns	1,3
Δt_{SVSV}	3	Strobe valid to strobe valid	-4	+4	ns	1,4
Δt_{SLSH}	4	Strobe low to strobe high	-4	+4	ns	1,5
Δt_{SHSL}	9	Strobe high to strobe low	-4	+4	ns	1,5
trDS	5	Read data valid after notMem-CAS low		$5((N*4)-e1)-18$	ns	9
trDH	6	Read data hold after notMem-CAS high	$5((N*4)-e2)-10$		ns	10
Δt_{wDS}	7	Write data setup before related strobe low	-4	+4	ns	6, 8
Δt_{wDH}	8	Write data hold after related strobe high	-4		ns	7, 8
tSVWV	10	Wait setup time		$5((m*4)-e1)-8$	ns	11, 12
tSVWI	11	Wait hold time	$5((m*4)-e1)+7$		ns	11, 12

Table 10.22 PMI AC specifications

Notes

- 1 The timings are based on the following loading conditions: address pin loaded with 75 pF and strobe pins loaded with 75 pF.
- 2 The nominal value is given by the programmed position of the falling edge (**RASEdgeTime** or **E1Time**) of the **RAS**, **CAS** or **PS** strobe.
- 3 The nominal value is given by the programmed length of the sub-cycle (**RASTime** or **CASTime**) minus the programmed position of the falling strobe edge (**RASEdgeTime** or **E1Time**).
- 4 The nominal value is given by the absolute difference between any pair of falling edges (**RASEdgeTime** or **E1Time**) or any pair of rising edges (**E2Time**) or any falling to rising pair of strobes.
- 5 The nominal strobe pulse width is given by the difference between the programmed falling (**RASEdgeTime** or **E1Time**) and the programmed rising edge (**E2Time**) of a **RAS**, **CAS** or **PS** strobe. This is applicable to a positive or negative pulse.
- 6 The nominal value is given by the programmed **E1Time** for **notMemCAS0-3** or **not-MemWrB0-3**.
- 7 The nominal value is given by the programmed **CASTime** *minus* the programmed position of the rising edge of **notMemCAS0-3** or **notMemWrB0-3**. Output data may be held indefinitely if there are no subsequent external cycles.
- 8 Timings are for all four byte write strobes **notMemWrB0-3**.
- 9 Where, **N** is the number of **CASTime** cycles and **e1** is the programmed falling edge position (**E1Time**).
- 10 Where, **N** is the number of **CASTime** cycles and **e2** is the programmed rising edge position (**E2Time**).
- 11 A wait state will be inserted between cycle **m** and cycle **m+1** in an **n** cycle access if the specified relationship is met.
- 12 In an **n** cycle access a wait cycle cannot be added after cycle **n**. Also, a wait cycle cannot be added before cycle 1.

Read cycle

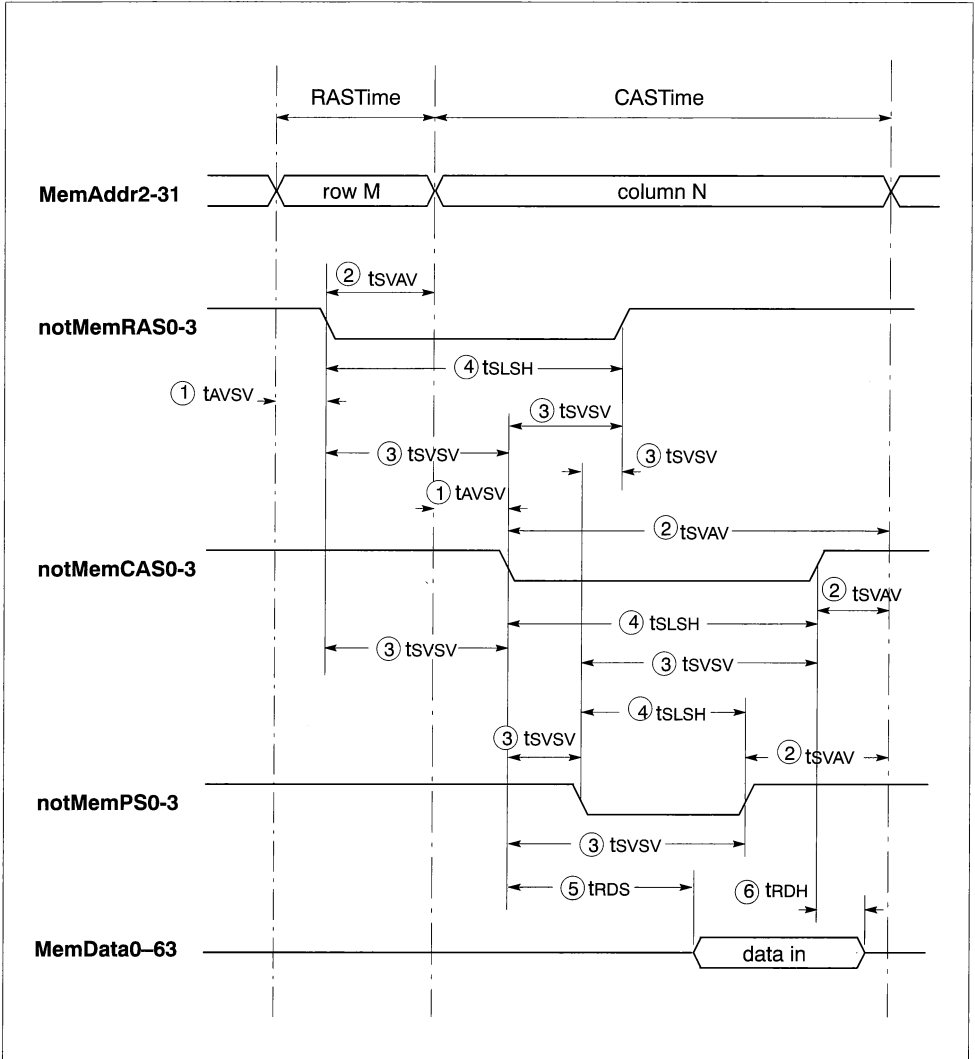


Figure 10.22 Read cycle timing

Write cycle

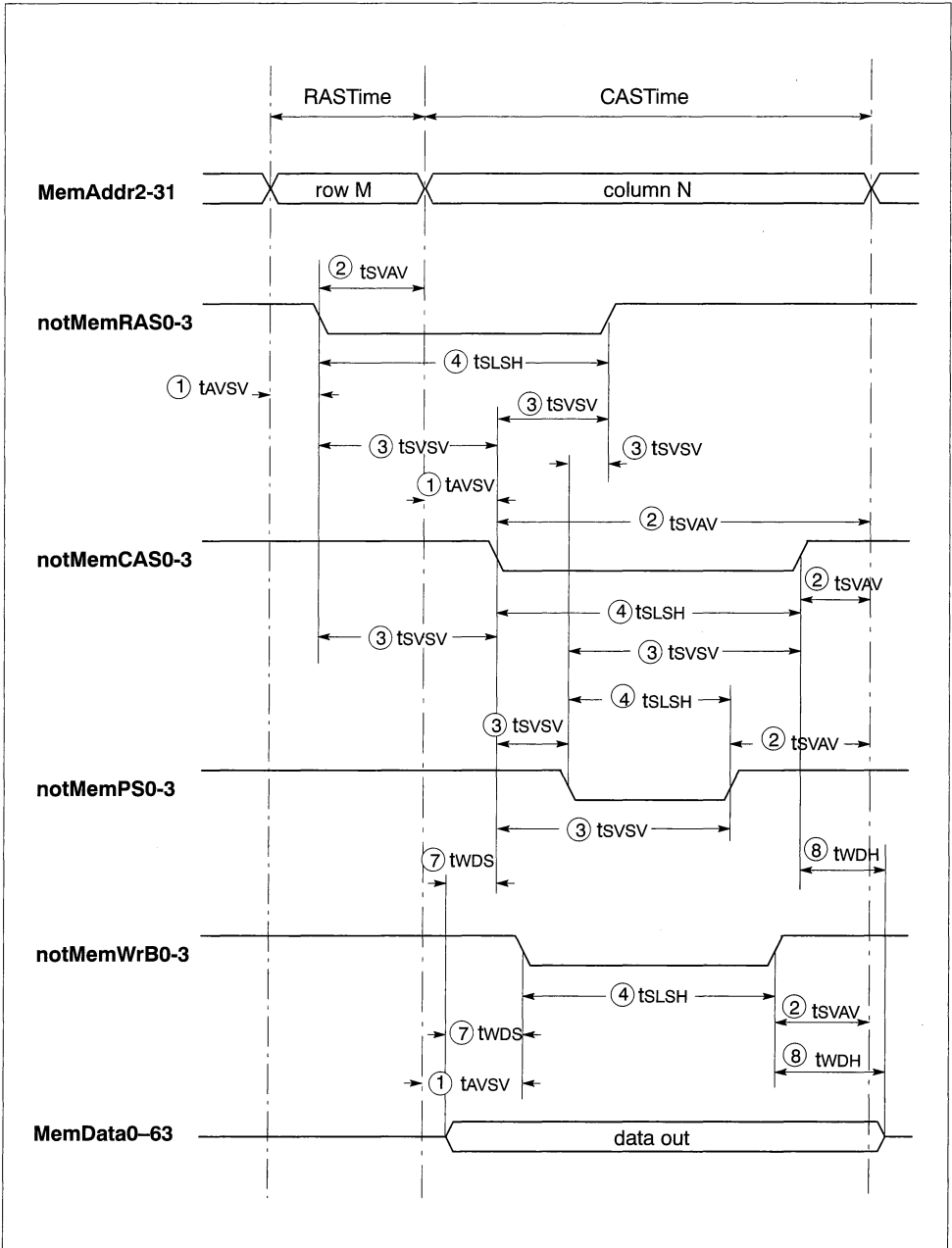


Figure 10.23 Write cycle timing

Consecutive cycles

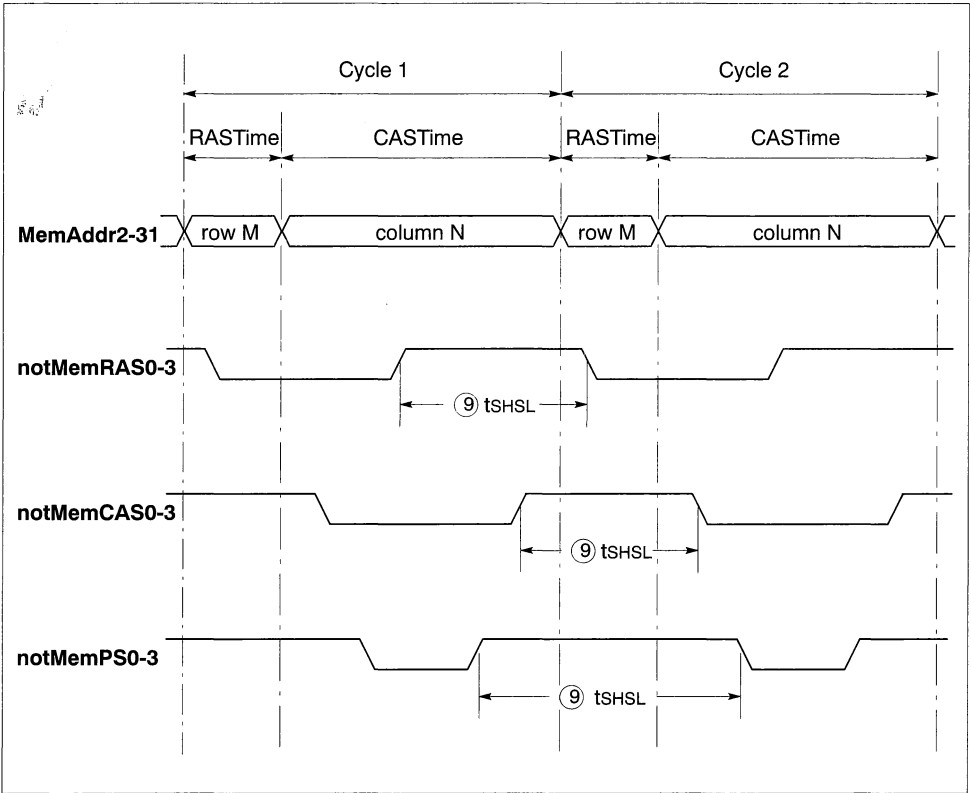


Figure 10.24 Cycle to cycle timing

Memory wait

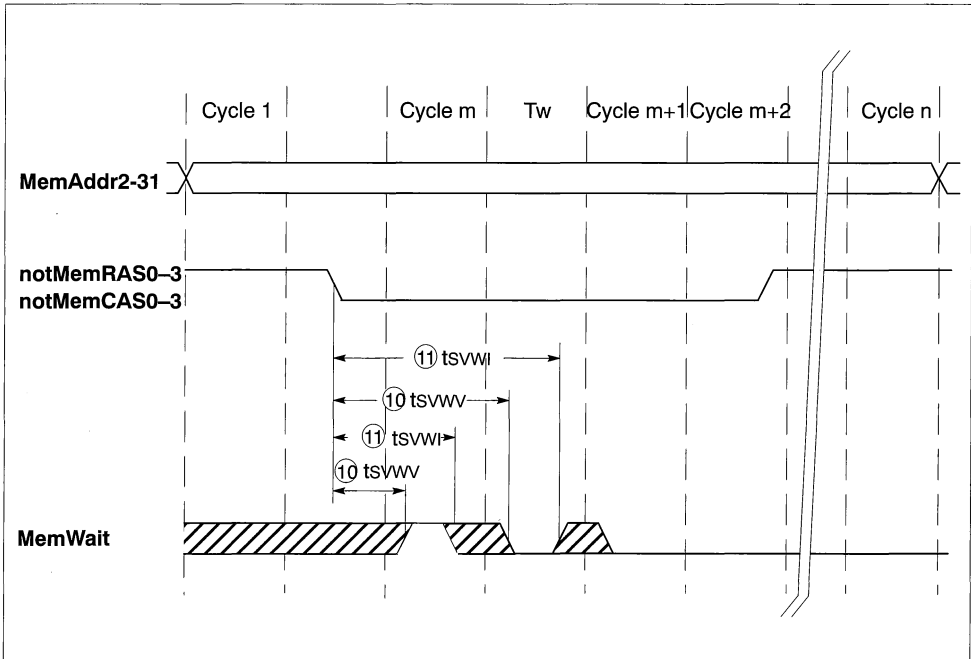


Figure 10.25 Memory wait timing

11 Communications

The IMS T9000 provides hardware support for process-to-process communication by means of channels. Channel communication is point-to-point, synchronized, uni-directional and unbuffered. As a result, a channel needs no process queue, no message queue and no message buffer, and so can be implemented very efficiently.

11.1 Overview

11.1.1 Channels

A channel is used for synchronization and data-transfer between two processes and is termed either an internal channel or an external channel as described below.

Internal channels

- An internal channel (sometimes referred to as a soft channel) connects processes that are on the same processor.

External channels

For communication between processes on different transputers, or communication between a process and a non-transputer device, external channels must be used. There are three types of external channel: virtual channels, byte-stream channels and event channels.

- A virtual channel allows communication between processes on different IMS T9000 transputers. These transputers do not have to be directly connected, provided there is a connecting path via a communications network.
- A byte-stream channel allows communication between a process on an IMS T9000 transputer and a process on a neighboring T2/T4/T8-series transputer.
Note: Links between these transputers need data and control protocol translation. This can be achieved using an IMS C100 system protocol converter. See the *IMS C100 datasheet (document number 42 1475 02)* for details.
- An event channel allows a communication between a process and an external device. This type of channel can carry no message: it is purely for synchronization. For example one use of the event channel is as an external interrupt input. Events are described separately in chapter 12.

An internal channel between two processes executing on the same transputer is implemented by a single word in memory; an external channel between processes executing on different transputers is implemented by means of point-to-point links, which are described separately in the Data/Strobe links chapter 13.

11.1.2 Channel addresses

A channel is uniquely identified by a 'channel address'. For an internal channel, the channel address corresponds to a memory location which may be allocated by the compiler. For an external channel, the channel address belongs to a range of addresses reserved for external channels (see section 11.6).

11.1.3 Communication instructions

The processor provides a number of operations to support message passing along channels, the most important being *in* (input message) and *out* (output message).

The *in* and *out* instructions use the address of the channel to determine whether the channel is internal or external. Thus the same instruction sequence can be used for either, allowing a process to be written and compiled without knowledge of whether its channels are connected to other processes on the same transputer, or other transputers.

Channel communication takes place when both the inputting and outputting processes are ready. Thus, the process which first becomes ready must wait until the second is also ready. A process performs an input or output by loading the evaluation stack with: a pointer to a message; the address of a channel; and a count of the number of bytes to be transferred, and then executing an input or output instruction. Data is transferred if the other process is ready. If the other process is not ready then the one executing the communications instruction will be descheduled. When the other process becomes ready they both continue to execute.

Communication may be: zero-length; fixed-length; or variable-length.

11.1.4 Efficient variable-length communications

Communication using the *in* and *out* instructions requires both communication processes to have knowledge of the length of the message that is to be transferred. To allow the secure and efficient communication of variable-length data, the *vin* (variable input) and *vout* (variable output) instructions may be used instead of *in* and *out*. Variable length communication requires only the outputting process to have knowledge of the length of the message prior to transfer.

When both a *vin* and a *vout* instruction have been executed by processes referring to the same channel, providing the length specified by *vout* does not exceed the length specified by *vin*, data is transferred from the outputting process to the inputting process just the same as if *in* and *out* had been used.

However, in the case where the length specified by *vout* exceeds that specified by *vin*, a special value is returned in the count location of the workspace of the inputting process, to indicate that an error has occurred in communication.

The *ldcnt* (load count) instruction is provided to enable the inputting process to determine either how much data was transferred during a variable length communication, or whether an error in communication occurred.

11.2 Virtual channel processor

The IMS T9000 incorporates a hardware communications processor, called the *Virtual Channel Processor* (VCP), which is able to multiplex any number of *virtual channels* over each physical link. Each message is split into a sequence of packets, and packets from different messages may be interleaved over each physical link. Interleaving packets from different messages allows any number of processes to communicate simultaneously via each physical link. IMS T9000 transputers may be connected directly or via a network of IMS C104 dynamic routing devices. Communication channels can be established between any two processes regardless of where they are physically located, or whether the channels are routed through a network. Thus, programs can be independent of network topology.

In order that packets which are parts of different messages can be distinguished by the VCP of the transputer which receives them, each received packet contains one or two bytes which identify a virtual input channel of the receiving transputer. When a packet is transmitted it may also contain information to route the packet through a packet switching network. The combination of any routing information and the identification of the virtual input channel of the receiving transputer is called the packet header. Every packet of a message ends with an end-of-packet (EOP) token, except the last packet which ends with an end-of-message (EOM) token.

The maximum length of data in each packet is 32 bytes. All but the last packet of a message contain the maximum amount of data; the last contains the maximum amount of data or less. Each packet has the structure illustrated in figure 11.1. The header bytes (containing routing and channel information) are transmitted first, followed by the data bytes of the packet (if any), followed by the encoded end of packet marker.

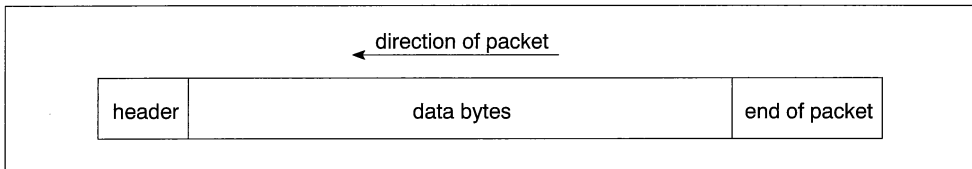


Figure 11.1 Structure of a packet

The VCP distinguishes three types of packet, depending on whether or not there are any bytes of data in the packet, and whether it is terminated with an EOP or an EOM.

- If the packet is terminated with an EOM token it is the last, possibly the only, packet of a message.
- If it contains data and is terminated with an EOP token it is part of a message.
- If it contains no data and is terminated with an EOP token it is taken as the acknowledgement of a previously transmitted packet.

11.2.1 VCP protocol

The VCP enforces a high-level protocol on each virtual channel. Each packet of data sent along a virtual channel must be acknowledged before the next is sent to ensure that no data is lost. The last packet must be acknowledged before the outputting process is rescheduled to ensure synchronized communication. Data packets received on a virtual channel are acknowledged by the VCP by sending acknowledge packets on another virtual channel back to the VCP which sent them. Intermediate network components are transparent and acknowledgement is process-to-process (processor-to-processor).

11.2.2 Virtual links

Virtual channels always occur in pairs between pairs of communicating processors, with one virtual channel in each direction. If a message is being communicated in one direction the virtual channel in the opposite direction is used to return acknowledge packets to the sender. The associated pair of virtual channels

is referred to as a *virtual link*. A virtual link can transfer messages in both directions at the same time with data packets and acknowledge packets being interleaved on both of the virtual channels. Because virtual channels are always paired in this way it is not necessary to include source information in the packets. Thus packet headers only represent their destinations, thus maximizing the use of link bandwidth.

11.2.3 VCP link queues

The VCP sends messages out of links a packet at a time, waiting for the receipt of an acknowledge packet for every data packet sent before sending the next data packet on that virtual link. The VCP sends an acknowledge packet as soon as it receives the start of the data packet, provided a process is ready to receive the message, thus acknowledge transmission can overlap data transmission. In this way, the next data packet can be sent immediately after the previous one with no gap in between, in most cases.

The IMS T9000 sends all packets of a particular message down the same link. If more than one virtual channel is using a link then packets belonging to messages on the different virtual channels are interleaved.

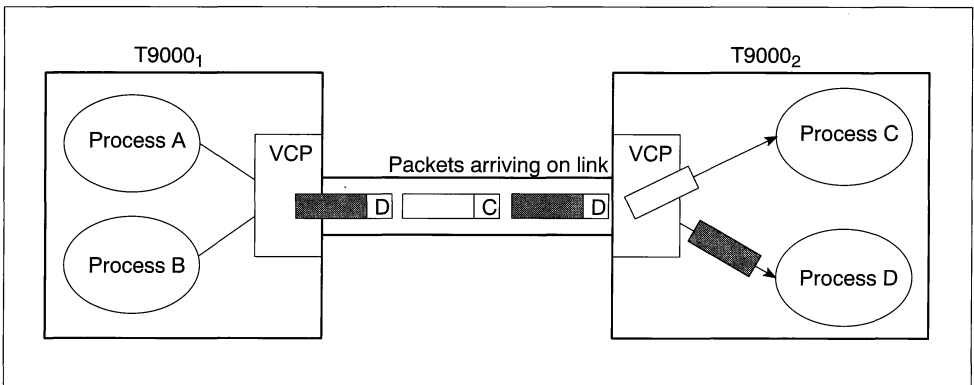


Figure 11.2 Example of multiple virtual channels using a single link

There can only be one message on any channel at one time, and any process can only be communicating on one channel at any time. Incoming packets (data or acknowledge) for any channel may arrive on any link.

Channels share links. This is achieved by maintaining queues of packets ready to be sent for each link. As a link finishes sending a packet it is fed the next packet off the queue. The receipt of an acknowledge packet for a channel puts the next data packet, if there is more of the message to send, onto the back of the queue for the appropriate link.

Four queues are maintained for each link, these are for:

- high priority data packets;
- high priority acknowledge packets;
- low priority data packets;
- low priority acknowledge packets.

Requests are only processed from the low priority queues of a link if both the high priority queues are empty. Within each priority, acknowledge and data packets are sent alternately as long as neither queue is empty.

Note, packetization, acknowledgements and link queuing are all handled automatically by the VCP.

11.2.4 Virtual link control blocks

In the IMS T9000, each end of a virtual link is represented by a data structure, called a *virtual link control block* (VLCB). The VCP uses VLCBs to record the state of communications on the virtual links. A virtual link has two associated VLCBs, one in the memory of each transputer connected by the link. In these blocks information is stored about the current progress of messages on both virtual channels of the virtual link.

The following information on VLCB's is included for completeness only. The user does not need to know full details of the VLCB's as a number of instructions are provided on the IMS T9000 for manipulating these data structures. These instructions may be used to establish the links, dynamically alter the connections, activate, deactivate and reset the channels, place channels into resource mode and debug parallel programs. For information on how VLCBs are set up, inspected and modified by means of specialized instructions refer to the *T9000 Instruction Set Manual*.

Each VLCB is 8 words long and aligned on an 8-word boundary. Table 11.1 shows the information stored in each VLCB. Note, users do not need to know the precise content of the VLCB, it is included here for reference only.

VLCB slot name	Function
vl.HeaderCtrl	Header and control word, see table 11.2.
vl.DataQueueLink	Pointer to next VLCB with a data packet to send.
vl.OutputWdesc	Workspace descriptor of outputting process.
vl.OutputLimit	Limiting address from which data may be sent.
vl.InputWdesc	Workspace descriptor of inputting process.
vl.InputLimit	Limiting address to which data may be written.
vl.BufferPointer	Pointer to the input packet buffer.
vl.AckQueueLink	Pointer to next VLCB with an acknowledge packet to send.

Table 11.1 Content of the VLCB data structure

vl.OutputWdesc and **vl.InputWdesc** store the workspace descriptors of the processes (if any) sending and/or receiving messages on the virtual link. The input workspace descriptor (**vl.InputWdesc**) must be initialized to *NotProcess.p* (refer to Appendix A for this value) since there is initially no process communicating.

vl.InputLimit and **vl.OutputLimit** are pointers to the limit address for input data and output data.

Associated with each virtual link is a buffer which is used to store a packet which arrives on that virtual link in the case that no process has a pending input on that virtual link. **vl.BufferPointer** is a 32 bit pointer to a region of memory allocated for this purpose. Buffers must be word aligned.

vl.HeaderCtrl word

The **vl.HeaderCtrl** word contains a number of bits of control and header information. The contents of the three least significant bytes (bytes 0, 1 and 2) of the word are dependent on the information contained in the most significant byte (byte 3) of the word, see table 11.2. Bytes 0, 1 and 2 either contain: one, two or three bytes of packet header to be included with each packet (data or acknowledge) sent from the transputer on that virtual link; or the length of the header and an unsigned word offset from the **HdrAreaBase** register to the location in memory where the header is held. The header bytes (if any) are sent in the order 0, 1, 2. Table 11.2 shows the content of the most significant byte of the **HeaderCtrl** word.

Bit field	Function
Link number (2 bits)	The physical link used by this virtual link.
Input normal	The virtual input channel is operating normally.
Output normal	The virtual output channel is operating normally.
Header type (2 bits)	Signifies the contents of bytes 0, 1 and 2. 00 bytes 0,1 are a word offset, byte 2 is the header length 01 byte 0 is the header 10 bytes 0,1 are the header 11 bytes 0,1,2 are the header

Table 11.2 Content of the most significant byte of the **vi.HeaderCtrl** word

Headers up to 3 bytes long are held in the VLCB; longer headers are held in a special region of memory. The encoding of short headers within the **vi.HeaderCtrl** word saves a memory access on every packet sent.

vi.DataQueueLink and **vi.AckQueueLink** words

The physical links are shared by a number of virtual links by threading the VLCBs of the virtual links waiting to use the links, on linked lists.

Each channel of the virtual link may carry both data packets (sent as a result of outputs on the output virtual link) and acknowledge packets (sent as a result of inputs on the input virtual link). Thus there may be data and/or acknowledge packets to be sent on the virtual link. Therefore the VLCB contains two queue pointers for threading onto the lists; the **vi.DataQueueLink** and **vi.AckQueueLink** locations. These locations must be initialized to zero.

Note, the **vi.AckQueueLink** location is not automatically free when an input completes, as the VLCB may be on the queue to send the acknowledge for some time after the process is rescheduled.

11.3 Operation of the VCP

All internal (soft) channel communication is managed by the CPU. All external communication is handled by the VCP. When a communication is started on an external channel, the CPU deschedules the process (i.e. removes it from the scheduling queue) and passes it, and its request, to the VCP for processing. When the communication has completed the VCP reschedules the process directly without the intervention of the CPU. For full details of the requests which can be made between the CPU and the VCP, refer to the *T9000 Instruction Set Manual*.

The VCP operates concurrently with the CPU so that data transfers do not suspend CPU operation.

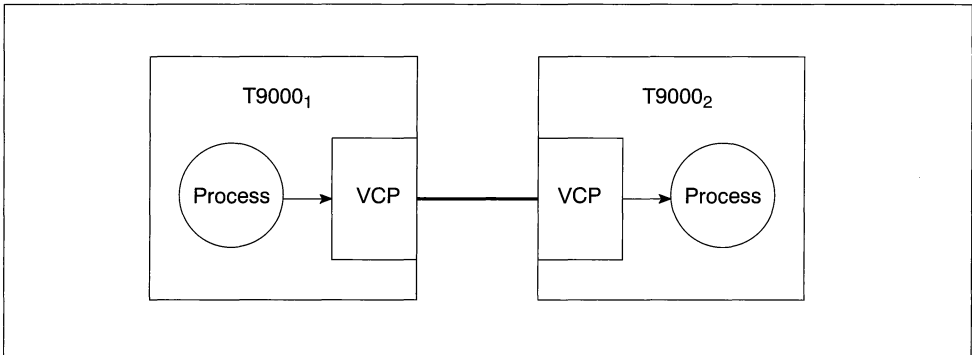


Figure 11.3 Communication between two IMS T9000 transputers

Consider a communication between two transputers; $T9000_1$ and $T9000_2$ as shown in figure 11.3. The VCP of $T9000_1$ will send the first packet of the message on a virtual channel to $T9000_2$ after the CPU performs an output (*out*, *outbyte*, *outword* or *vout*) instruction. When the VCP of $T9000_2$ receives the packet, it identifies the virtual channel on which the packet was received from the packet header. If the process on $T9000_2$ has performed an input (*in*, *vin*) instruction on the channel, the data contained in the packet is stored in the data space of the inputting process and an acknowledge is sent. If there is no process ready to receive the first packet then it is placed in an in-store packet buffer associated with the virtual link, which is large enough to hold the 32-byte maximum data length. In order that the data contained in the buffer is not overwritten, the VCP of $T9000_1$ does not send another packet on that channel until it receives an acknowledgment that a process on $T9000_2$ has become ready to receive the message. When the inputting process becomes ready the first packet is copied from the buffer into the data space of the process and an acknowledge packet is sent. This buffering is transparent to the processes because it is never in use when the processes are active.

11.3.1 Channel states

A channel may be deactivated, and later re-activated. If a channel is deactivated:

- no messages will be sent out of the channel.
- no actions (e.g. queueing acknowledge packets, scheduling processes) are performed but are recorded as pending.

All pending actions are performed when the channel is re-activated.

Individual virtual channels may be stopped and/or reset. Stopping is a planned shutdown, while resetting is a drastic action that cleans out a channel (that may be damaged, e.g. by incorrect software).

Resetting channels

In the event of a failure of a physical link (which may have resulted in the loss of one or more packets) a virtual link can be reset, using the *resetch* (reset channel) instruction. This resets one end of the virtual

channel. Since there may be packets in transit, it is generally necessary to reset both ends of a virtual channel, then wait long enough for any such packets to arrive, before resetting both ends again.

Resetting a channel does not affect whether it is deactivated.

Stopping channels

A channel can be stopped using the *stopch* (stop virtual channel) instruction. When an input channel is stopping, any received packets are acknowledged. When an output channel is stopping, any packet queued will not be sent, but acknowledges will be received for any packets which have already been sent. When all transmitted packets have been acknowledged, the process that executed the *stopch* instruction is rescheduled, leaving the output channel ready for re-use. The input channel can then be made ready for re-use by a single *resetch* (reset channel) instruction.

11.4 Resources

The IMS T9000 supports the efficient implementation of shared resources. This is done by enabling a number of (user) processes to communicate with a single server process via resource channels. The processes may be on the same or different transputers. Access to the server process is controlled by a resource data structure (RDS) which resides on the same transputer as the server process. The resource data structure consists of three words in memory. The word slots in the data structure are shown in table 11.3, and must be initialized with all three words set to *NotProcess.p*. The queue contains resource channels (local and/or remote) on which processes waiting to access the server have performed an output.

Slot name	Purpose
rds.Front	Pointer to workspace of first process waiting on the resource queue.
rds.Back	Pointer to workspace of last process waiting on the resource queue.
rds.Proc	Process descriptor of resource server process.

Table 11.3 Contents of a resource data structure

A resource channel is a channel which has been set into resource mode using the *mkrc* (mark resource channel mode) instruction. A channel in resource mode has an additional two word data structure. The contents of the resource channel data structure is shown in table 11.4. The **rc.Id** slot is used to indicate the mode of the channel. If it has the special value *NotProcess.p*, then the channel is in normal mode, otherwise it is in resource mode. The initial state of the two-word data structure is **rc.Id** word set to *NotProcess.p* as the channel is in normal mode when memory is first allocated.

Slot name	Purpose
rc.Id	Resource channel identifier / mode indicator.
rc.Ptr	Pointer to resource data structure or next resource channel.

Table 11.4 Contents of a resource channel data structure

The user processes communicate to the server process by executing normal output (*out*, *outbyte*, *outword* and *vout*) instructions. The server process is connected to a user process by means of the *grant* instruction; once connected the server process can receive data from the user process by means of input instructions.

The *enbg* (enable grant) and *disg* (disable grant) instructions enable resources to be used in alternative constructs.

11.5 Byte-stream mode

Each IMS T9000 data link (**Link0-3**) may be set to operate either in virtual channel mode or in byte-stream mode by setting the associated bit in the VCP link mode registers (see section 11.7.6). The IMS T9000 links can be set independently of each other, enabling each IMS T9000 to be connected to several different networks.

Byte-stream mode is for use in interfacing with individual T2/T4/T8-series transputers or transputer systems comprising both T2/T4/T8-series transputers and T9000 transputers. Links between T2/T4/T8-series transputers and T9000 transputers need data and control protocol translation which can be achieved using an IMS C100 system protocol converter. Refer to the *IMS C100 datasheet* for details.

A special protocol is used between the IMS C100 and the IMS T9000. This protocol is invisible to the user, and is described here for completeness. Data is transferred along the DS-Link in the form of packets each with a single byte header. Each packet is terminated with either an EOP or EOM token. The packets sent to/from a DS-Link in byte-stream mode are given in tables 11.5 and 11.6. Software on the IMS T9000 sends and receives messages normally via a pair of byte-stream channels.

Header	Data	Terminator	Interpretation	Notes
0	32 bytes	EOP	Part of message	
0	1 to 32 bytes	EOM	End of message	
0	none	EOP	Acknowledgement	
1	1 to 4 bytes	EOM	Input count	1

Notes

- 1 The 'input count' packet contains the count of the data bytes the IMS T9000 expects to receive from the T2/T4/T8 transputer, via the IMS C100.

Table 11.5 Packets from IMS T9000 to IMS C100

Header	Data	Terminator	Interpretation	Notes
0	32 bytes	EOP	Part of message	
0	1 to 32 bytes	EOM	End of message	
0	none	EOP	Acknowledgement	1
0	none	EOM	Unsolicited byte	2

Notes

- 1 The acknowledgement packet is sent when the IMS C100 is ready to receive more data.
- 2 If a byte is received from the OS-Link whilst the output count is zero, the count is effectively reduced to -1 and an unsolicited packet is sent.

Table 11.6 Packets from IMS C100 to IMS T9000

For links operating in byte-stream mode the packet headers are fixed and the byte-stream channels cannot be deactivated, stopped or put into resource mode. For compatibility with the T8xx, the **Wdescriptors** of processes actually communicating on byte-stream channels are stored in the 8 words at the bottom of memory. The base and limit pointers for the message are then stored in the **pw.Link** and **pw.Pointer** locations of the process workspace, see table 3.2, page 69.

Note that if a link is set into byte-stream mode, there should be no VLCB whose **vl.HeaderCtrl** word specifies that link number. If there is and a communication is performed on the corresponding virtual link, the behavior is unspecified.

Note, a link operating in byte-stream mode must be configured to expect 1 byte headers, otherwise its behavior is undefined.

Note, two links in byte-stream mode connected directly together will not operate correctly.

11.6 Memory and channel address spaces

A channel address is the value used to access a channel in a communication instruction. The IMS T9000 channel address space is shown in figure 11.4. The IMS T9000 memory map is shown in figure 11.5.

The memory map is byte addressed. The main memory address space starts at address #80000000.

Event channels are always accessed via channel addresses. The physical links are normally accessed via virtual channels. However, each IMS T9000 physical link set to operate in byte-stream mode is accessed via the byte-stream channel addresses.

11.6.1 Channel address space

The bottom eight channels (0 to 7) are used for compatibility with T2/T4/T8-series transputers mapping channels onto hard links. The next eight (8 to 15) are the event channels.

Each virtual link corresponds to two channels, one input and one output, which are represented by distinct channel addresses. The input channel of the first virtual link will be represented by the address (**MostNeg+16** words), and the output channel by (**MostNeg+17** words), i.e. all the input channels are at even word addresses, and all the output channels are at odd ones (see the channel address space). The channel address is the machine form of the channel number. The CPU input and output instructions identify channels to the VCP by quoting the channel address.

Above the virtual channels there is a region of invalid channel numbers and then the region of valid soft (internal) channel numbers.

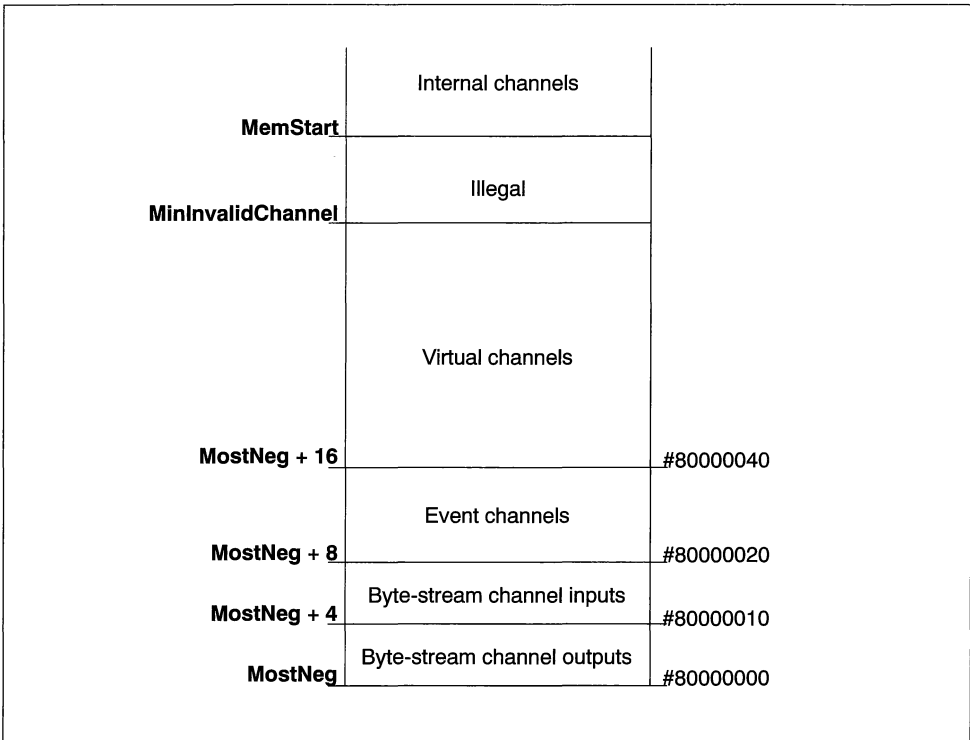


Figure 11.4 IMS T9000 channel address space

11.6.2 Memory allocation for virtual links

The data structure to implement virtual channel communication on the IMS T9000 consists of four parts:

- VLCBs;
- buffers for unsolicited data packets;
- two-word data structures for using input event and virtual channels in resource mode;
- area of memory for headers which are too long to fit into VLCBs.

With the exception of the packet buffers, each part consists of a single block of store. The block of memory for the VLCBs starts from a fixed location, at **MostNeg+16** words. Each VLCB requires 8 words. The blocks for the resource mode words and long headers are pointed to by the external resource channel base register (**ExternalRCBase**) and the header base register (**HdrAreaBase**) respectively. The virtual link packet buffers can be placed anywhere in memory.

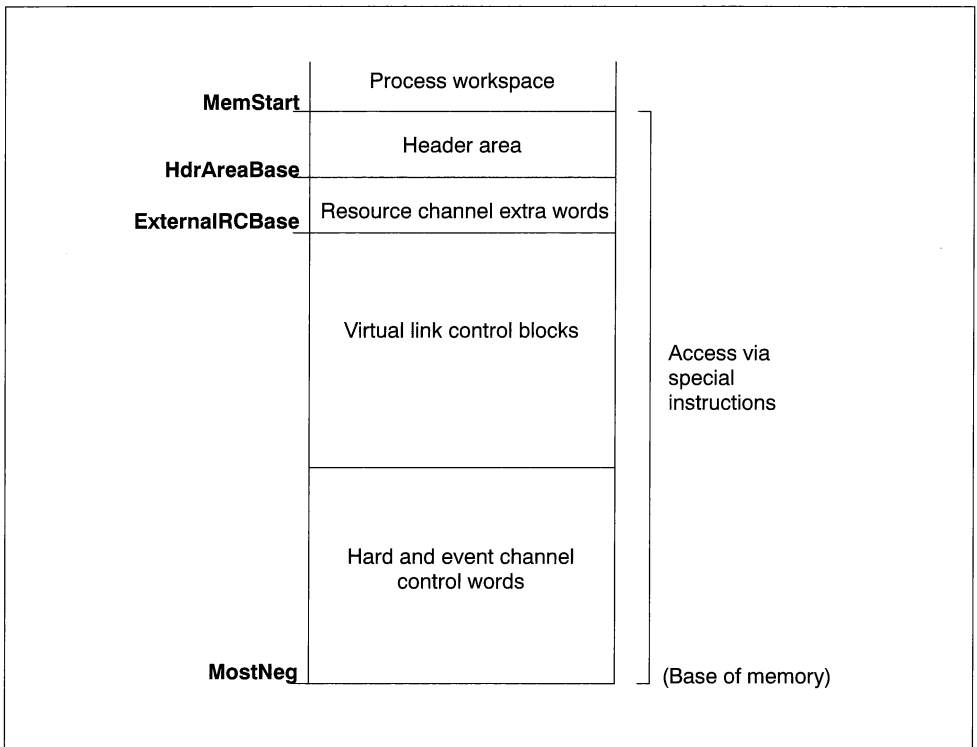


Figure 11.5 IMS T9000 memory map showing memory allocation for virtual links

There are three CPU configuration registers which define the communication space allocated.

Memory start value register

The **MemStart** register contains a pointer to the start of memory.

The communications instructions operate by treating all channel addresses at or above **MemStart** as being internal ("soft") channel communications – that is between processes executing on the same processor. All channel communications below this address are transferred to the VCP, after checking for illegal addresses.

The *ldmemstartval* instruction can be used to obtain the value of **MemStart**.

Packet buffers may be allocated below **MemStart** in the memory map.

Minimum invalid virtual channel register

There is a range of channel addresses below **MemStart** which do not correspond to valid virtual channels, and which will normally contain VLCBs and headers. The first channel address which corresponds to an invalid virtual channel is held in the **MinInvalidChannel** register.

MinInvalidChannel should always be \leq **MemStart**

Note that all external communications can be disabled by setting the **MinInvalidChannel** register to **MostNeg**.

External resource channel base register

A resource channel is a channel which may be in *normal mode* or *resource channel mode*, plus a two word data structure (see section 11.4 on resources). For local users the extra two words are contiguous with the

word used as the channel. For remote users an extra two words are associated with each input virtual channel and Event input. These extra words are allocated together in a block, and the base of the block is defined by the **ExternalRCBase** register. Note that the two words must be allocated for all four event channels at the bottom of the block before any are allocated for virtual channels. Note also that, since the mapping between channels and this block is fixed, there must not be any gaps in the allocation. Cache usage efficiency is maximized by ensuring that this block is two-word aligned.

11.7 VCP configuration registers

The functionality of the VCP is controlled by nineteen VCP configuration registers. Four are associated directly with the VCP and four are associated with each of the four links. The registers are accessed via the *ldconf* (load from configuration register) and *stconf* (store to configuration register) instructions, or via *CPeek* and *CPoke* command messages received along control link **CLink0**. This section describes the functionality of the VCP to be controlled by bit fields in the associated configuration registers. It also defines the relationship between the addressing of channels and the addressing of store.

Note, all INMOS reserved bits in the following tables must always be written with 0's.

11.7.1 VCP command register

The VCP command register (**VCPCommand**) enables commands to be issued to the VCP. Each bit of the register corresponds to a command, see table 11.7 below. The command is executed when the bit is set. Each write to the register can set only one bit.

VCPCommand		#0804	Read/Write
Bit	Bit field	Function	
0	Reset	Reset the VCP – stops the VCP and resets the registers to their undefined level 2 state.	
1	Start	Start the VCP	
2	Stop	Stop the VCP 'cleanly' so that channel states are preserved. The VCP accepts messages currently in transit but no new messages can be sent.	
31:3		INMOS reserved	

Table 11.7 Bit fields in the **VCPCommand** register

11.7.2 VCP status register

The VCP status register (**VCPStatus**) contains information following the occurrence of an error on an input packet. Once an error is flagged the packet body is discarded and the following information is returned to the **VCPStatus** register; the header of the packet, the error code, and the link number on which the packet was input. Any subsequent errors are not recorded.

Writes to this register clear the contents regardless of the value written.

VCPStatus		#0802	Read/Write
Bit	Bit field	Function	
0	Error	Error indicator bit set to 1 to signal that an error has been detected.	
2:1	ErrorCode	Details the type of error which has occurred, see table 11.9.	
5:4	LinkNumber	Number of link on which the packet causing the error was received.	
31:16	InputHeader	Header of errant input packet.	
3, 15:6		INMOS reserved	

Table 11.8 Bit fields in the **VCPStatus** register

Error type	Error code
Header out of range	00
Header too short	01
Packet too short	10
Packet too long	11

Table 11.9 Error type

11.7.3 Header area base register

If the header associated with a virtual channel is longer than three bytes, it is not held in the VLCB associated with that channel, but resides in a separate region of store. The (word-aligned) base of this region is defined by the header area base register (**HdrAreaBase**).

HdrAreaBase #0901		Read/Write
Bit	Bit field	Function
31:0	HeaderAreaBase	Defines the base of the region of memory reserved for headers longer than 3 bytes.

Table 11.10 Bit fields in the **HdrAreaBase** register

11.7.4 Header offset registers

The VCP must convert the channel address and header number to the memory address of the VLCB.

The VCP header offset registers for each link (**VCPLink0HdrOffset**, **VCPLink1HdrOffset**, **VCPLink2HdrOffset**, **VCPLink3HdrOffset**) are programmed with an offset which is subtracted from the value contained in the header of a packet which has been input on the associated physical link. The address of the VLCB to which a packet is directed is calculated by the VCP hardware using the following formula:

$$\text{Memory address} = \text{vlink.base} + ((\text{Header} - \text{HeaderOffset}) \ll \text{vlink.shift})$$

where for the IMS T9000: $\text{vlink.base} = \text{MostNeg} + 64$, and $\text{vlink.shift} = 5$.

Figure 11.6 shows the mapping of channel addresses and header numbers to the memory address of the VLCB. The example given shows 3 virtual links (6 virtual channels) using 2 words for long headers.

Note, the header offset registers must be initialized to 0 when not used.

VCPLink0-3HdrOffset #0808, #080C, #0810, #0814		Read/Write
Bit	Bit field	Function
15:0	HeaderOffset	Header offset for associated link.

Table 11.11 Bit fields in the **VCPLink0-3HdrOffset** registers

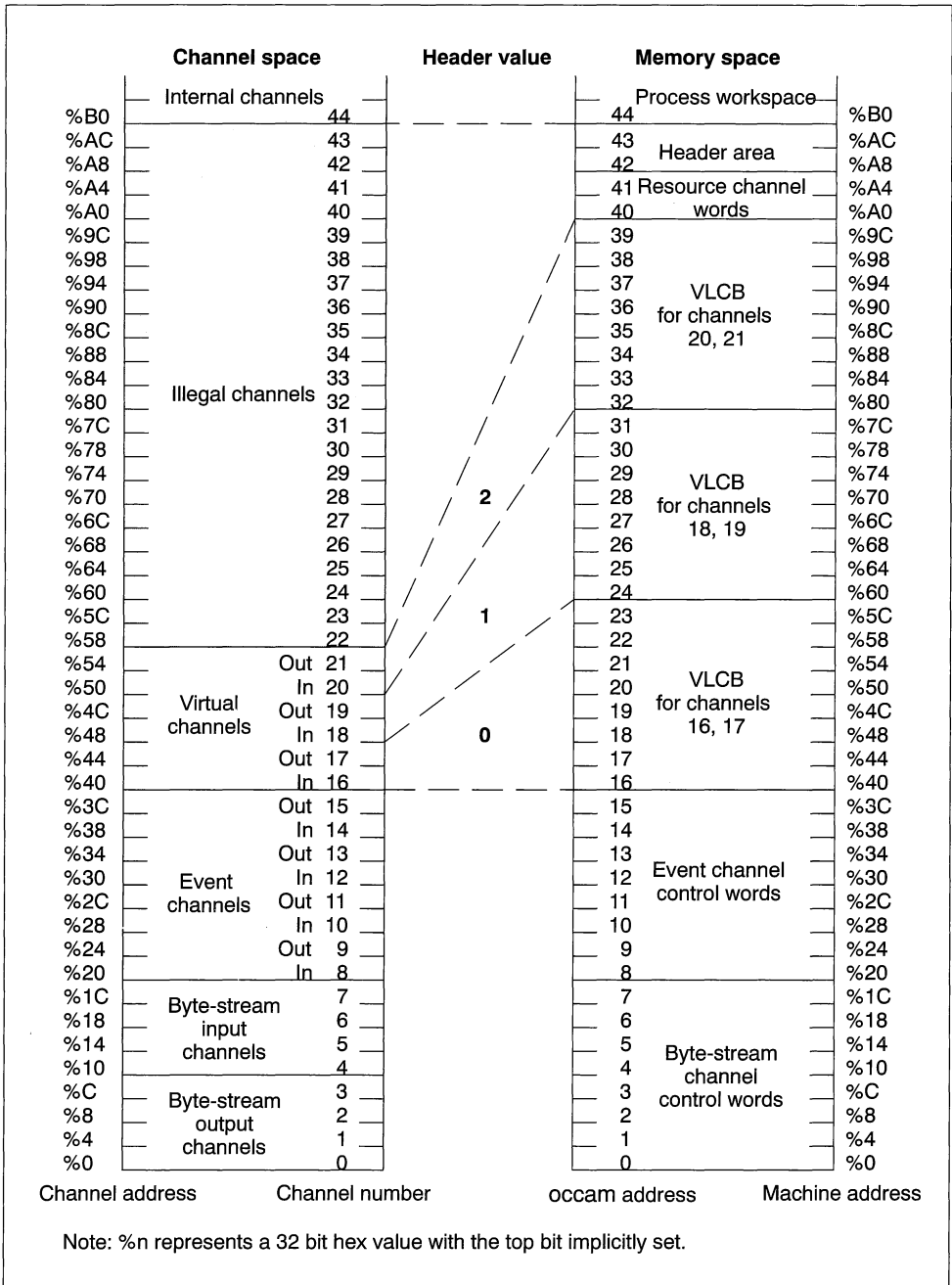


Figure 11.6 Mapping of channel addresses and header numbers to the memory address of the VLCB

11.7.5 Packet header limit registers

The base and limit of packet headers which are acceptable on each physical link are stored in the minimum header registers (**VCPLink0MinHeader**, **VCPLink1MinHeader**, **VCPLink2MinHeader**, **VCPLink3MinHeader**) and maximum header registers (**VCPLink0MaxHeader**, **VCPLink1MaxHeader**, **VCPLink2MaxHeader**, **VCPLink3MaxHeader**) for each link **Link0-3** respectively. Out of range headers cause the associated packets to be discarded and, unless the **LocalizeError** flag (see section 11.7.6) is set, will generate errors. These registers can be used for enhanced system security.

Note, all bits in the minimum header registers should be set to 0 (packet header minimum value = 0) and all bits in the maximum header registers should be set to 1 (packet header maximum value = 65,535) when not used.

VCPLink0-3MinHeader #0807, #080B, #080F, #0813			Read/Write
Bit	Bit field	Function	
15:0	MinHeader	Base of packet headers for associated link.	

Table 11.12 Bit fields in the **VCPLink0-3MinHeader** registers

VCPLink0-3MaxHeader #0806, #080A, #080E, #0812			Read/Write
Bit	Bit field	Function	
15:0	MaxHeader	Limit of packet headers for associated link.	

Table 11.13 Bit fields in the **VCPLink0-3MaxHeader** registers

11.7.6 VCP link mode registers

The VCP link mode registers (**VCPLink0Mode**, **VCPLink1Mode**, **VCPLink2Mode**, **VCPLink3Mode**) contain information about the four links **Link0-3**.

If bit 1 is set to 1 errors detected by the VCP on the associated link do not cause an error to be signalled to the control unit, and if bit 2 is 1 then two byte headers are expected on that link.

If bit 0 is 1 the link operates in byte-stream mode in conjunction with an IMS C100. For a link operating in byte-stream mode bit 2 must be 0, i.e. set to expect 1 byte headers, otherwise its behavior is undefined.

VCPLink0-3Mode #0805, #0809, #080D, #0811			Read/Write
Bit	Bit field	Function	
0	ByteMode	When set to 1 the associated link is set to operate in byte-stream mode (see section 11.5).	
1	LocalizeError	When set to 1 link errors detected by the VCP are no longer reported to the control unit, see section 11.9 for details of errors.	
2	HeaderLength	Programs the expected length of the incoming packet header (1 or 2 bytes) for each associated physical link. 0 1 byte header 1 2 byte header	
31:3		INMOS reserved	

Table 11.14 Bit fields in the **VCPLink0-3Mode** registers

11.7.7 ChanWriteLock

The **ChanWriteLock** register is a 1 bit register which when set to 1 prevents the CPU from writing to the following registers: CPU registers; VCP registers; and scheduler registers including the write lock register accessed via the configuration bus. Reads are not affected, nor are writes by the control unit.

The **ChanWriteLock** register is shared between the VCP, CPU and scheduler.

ChanWriteLock #4900		Read/Write
Bit	Bit field	Function
0	ChanWriteLock	When set to 1 it inhibits modification of the VCP, CPU and scheduler registers by the CPU.

Table 11.15 Bit fields in the **ChanWriteLock** register

11.8 Initialization of the VCP

11.8.1 VCP state on start up

The VCP is initially in a waiting state. In this state it does not dequeue VLCBs or transmit packets, nor does it initiate processing of incoming packets, which are stalled at the link inputs. However, if a packet is already being processed when reset is asserted it will be completed. In this state it will respond to requests from the CPU, which may cause VLCBs to be put on queues.

11.8.2 VCP state following reset

The VCP may be reset either by:

- a reset by the control unit.
In this case the VCP remains in the 'resetting' state for a fixed number of cycles.
- setting the **Reset** bit in the **VCPCommand** register.
The **Reset** bit must be set for a minimum of 256 cycles, when it is unset the VCP returns to the 'waiting' state with empty queues. The processor must not interact with the VCP other than to unset the **Reset** bit whilst the VCP is in this state.

The VCP is put into the 'running' state from the 'waiting' state by setting the **Start** bit in the **VCPCommand** register. Once in this state the VCP responds to requests from the CPU and incoming packets on all links. The VCP configuration registers should be programmed and all the VLCBs initialized before the VCP is started. The VCP may be returned to the 'waiting' state at any time by unsetting the **Start** bit.

When the VCP is in the 'running' state and the control unit sends a stop signal, the VCP goes into a 'stopping' state in which it deactivates every virtual channel on which it performs some action before performing that action. The exception to this is *activate channel* which is not performed. If stop is signalled when the VCP is in the 'waiting' state, it does not perform any actions until the **Start** bit is set, it then goes into the 'stopping' state. The effect of a stop signal from the control unit is the same as setting the **Stop** bit in the **VCPCommand** register. Note that this information is preserved for debugging purposes. It is not guaranteed that communication can be resumed simply by re-activating the virtual channels.

If the control unit signals an error to the VCP when in any state other than the 'resetting' state, the VCP goes into the 'discarding' state, which it remains in until reset. In the 'discarding' state the VCP does not initiate any packets. In order to avoid blocking the network, it accepts packets on all links and discards them.

11.9 Errors

The links can detect disconnection and parity errors. Both these errors cause an error to be signalled to the control system, unless this is prevented by the setting of the corresponding **LocalizeError** flag of the link on which the offending packet is received. Refer to the DS-Links chapter 13 for more information on link errors and their localization.

The VCP can detect non-attributable errors (errors which cannot be attributed to a particular process) and attributable errors as detailed below.

- Non-attributable soft errors:
 - Invalid header
 - Short non-terminal packet
 - Over-size packet
- Attributable soft errors:
 - Length overrun on an input

A length overrun error is dealt with by recording an invalid message length in the workspace of the inputting process. This can be detected by the process after it has been rescheduled by means of the *ldcnt* (load count) instruction.

The other errors cause an error to be signalled to the control system. The invalid header, short packet and long packet errors can be masked from the control system by setting the **LocalizeError** bit of the **VCPLink0-3Mode** register of the link.

Note that there are two registers which contain a **LocalizeError** bit. The **LocalizeError** bit in **Link0-3Mode** registers masks link errors detected by the link (refer to section 13.5, page 213). The **LocalizeError** bit in **VCPLink0-3Mode** registers masks different classes of link errors detected by the VCP. If the link errors are to be masked for a link the **LocalizeError** bits in both the **VCPLink0-3Mode** register and **Link0-3Mode** register for the link should be set together.

Null buffer pointers

If a data packet arrives on a virtual link which has a null buffer pointer the packet will be written to the bottom eight words of memory and no error will be signalled. This will cause undefined behavior if any of the links is being used in byte-stream mode.

12 Events

The **EventIn0-3** and **EventOut0-3** pins provide an asynchronous handshake interface between external events and internal processes. Event channels provide process synchronization but cannot transfer any data. Each pair of **EventIn** and **EventOut** pins can act either as an input or an output event channel, but not both simultaneously.

Input event channel

When an external event takes an **EventIn** pin high the associated external event channel is made ready to communicate with a process. When both the event channel and the process are ready the processor takes the associated **EventOut** pin high and the process, if waiting, is scheduled. **EventOut** is removed after **EventIn** goes low.

Output event channel

The IMS T9000 asserts the **EventOut** pin to instruct external hardware to perform an action. When both the event channel and the external hardware are ready the external hardware asserts the associated **EventIn** pin and responds to the instruction. **EventIn** should be removed after **EventOut** goes low.

Only one process may use each event channel at any given time. If no process requires an event to occur **EventOut** will never be taken high. Although an **EventIn** triggers the channel on a transition from low to high, it must not be removed before **EventOut** is high. **EventOut** is taken low when **Reset** occurs or when a *resetch* instruction is executed on that channel.

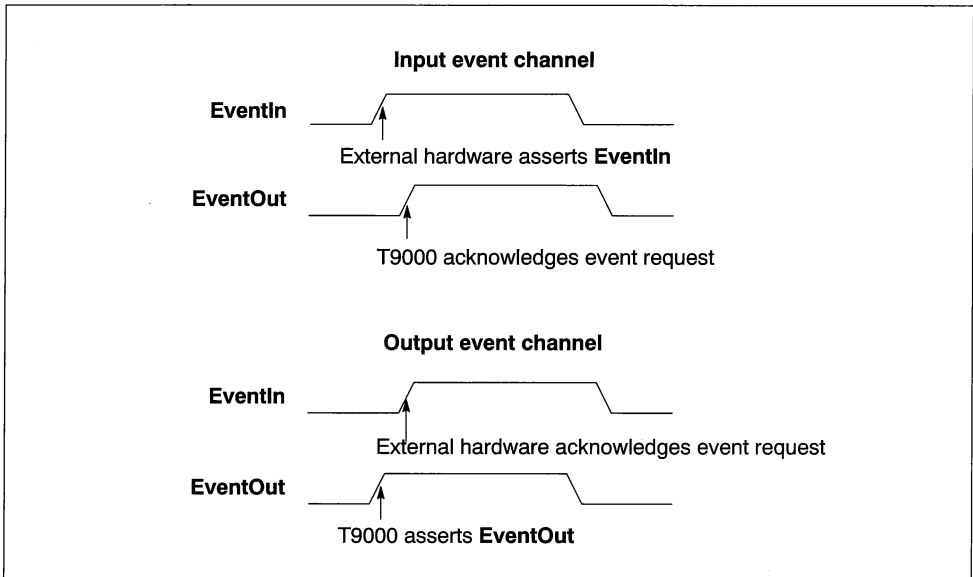


Figure 12.1 Event

Use of event channels with interrupts

An input event channel can be used for interrupts. Refer to the *T9000 Transputer Instruction Set Manual* for further information on how this can be achieved.

12.1 Event channel addresses

There are 8 words of memory reserved for event channels, from address #80000020 to #8000003C inclusive, see figure 12.2. These addresses are also the channel addresses of the event channel. Each successive pair of words corresponds to one event channel. The lower address is used when the event is configured for input, the higher address is used when the event is configured for output. Thus the event channel address space is consistent with the virtual channel address space. In addition, provided the first event channel (**Event0**) is configured for input, compatibility with the event channel address on the IMS T805 is achieved.

Each event input channel has a two word data structure to enable it to be used in resource mode. The (word-aligned) base of the block of memory for the resource mode words is pointed to by the **ExternalRC-Base** configuration register. The two words for each of the four event channels are allocated at the bottom of the block.

Address	Channel word	Channel address
MemStartVal	Internal channels	
MinInvalidChannel	Illegal	
	Virtual channels	
MostNeg + 16		#80000040
	Out 15	#8000003C
	In 14	#80000038
	Out 13	#80000034
	In 12	#80000030
	Out 11	#8000002C
	In 10	#80000028
	Out 9	#80000024
MostNeg + 8	In 8	#80000020
	Event channels	
MostNeg + 4	Byte-stream channel inputs	#80000010
MostNeg	Byte-stream channel outputs	#80000000

Figure 12.2 IMS T9000 channel address space

12.2 Event channel state

The values contained in each event channel word pair reflect the state of the event channel which is available for inspection. The following tables give the values corresponding to different states, when the event channel is configured as an input and when it is configured as an output. Note that the words must be set to the Empty state by writing *NotProcess.p* to both words before the channel is used. The constant *Deactivated.p* has the value #80000001. Refer to Appendix A for values and definitions of all the constants.

State	Word 0	Word 1
Empty (deactivated)	<i>NotProcess.p</i>	<i>Deactivated.p</i>
Inputting	Process WDesc	<i>NotProcess.p</i>
Inputting (deactivated)	Process WDesc	<i>Deactivated.p</i>
Enabled	Process WDesc	<i>NotProcess.p</i>
Enabled (deactivated)	Process WDesc	<i>Deactivated.p</i>
Resource mode	<i>ResChan.p</i>	<i>NotProcess.p</i>
Resource mode (deactivated)	<i>ResChan.p</i>	<i>Deactivated.p</i>

Table 12.1 Event channel states when event configured as input

State	Word 0	Word 1
Empty (deactivated)	<i>Deactivated.p</i>	<i>NotProcess.p</i>
Outputting	<i>NotProcess.p</i>	Process WDesc
Outputting (deactivated)	<i>Deactivated.p</i>	Process WDesc

Table 12.2 Event channel states when event configured as output

13 Data/Strobe links

The IMS T9000 has four bidirectional links for normal inter-processor communications, and two additional links which can only be used for control purposes. All of these links use a protocol with two wires in each direction, one for data and one to carry a strobe signal, see figure 13.1. These links are therefore referred to as data/strobe (DS-Links). The DS-Links are capable of:

- Up to 100 Mbits/s.
- 80 Mbytes/s peak total bidirectional data rate (20 Mbytes/s per link).
- Support for virtual channels and through routing.

The links are TTL compatible and are series matched to 100 ohm transmission lines. The links can be directly connected with no external buffering or other glue logic.

Figure 13.1 shows two transputers connected via DS-Links. In this example **Link0** of T9000₁ is connected to **Link3** of T9000₂.

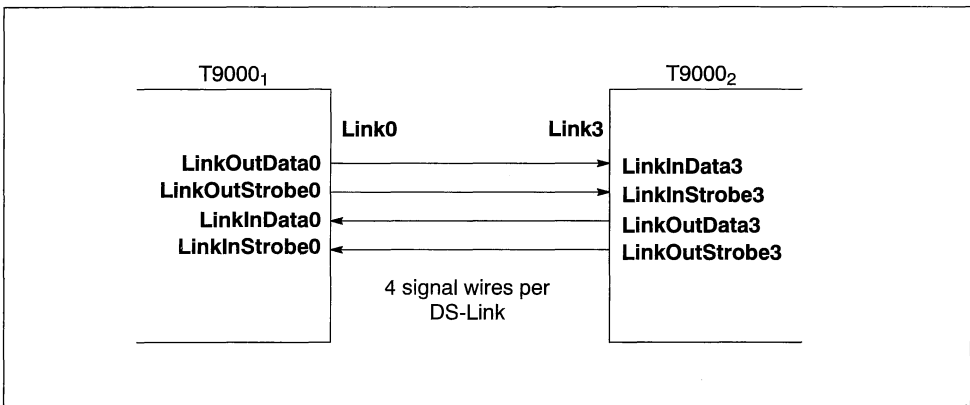


Figure 13.1 DS-Links

13.1 Link format and protocol

Each DS pair carries tokens and an encoded clock. The tokens can be data bytes or control tokens. Figure 13.2 shows the format of data and control tokens on the data and strobe wires. Data tokens are 10 bits long and contain a parity bit, a flag which is set to 0 to indicate a data token, and 8 bits of data. Control tokens are 4 bits long and contain a parity bit, a flag which is set to 1 to indicate a control token, and 2 bits to indicate the type of control token.

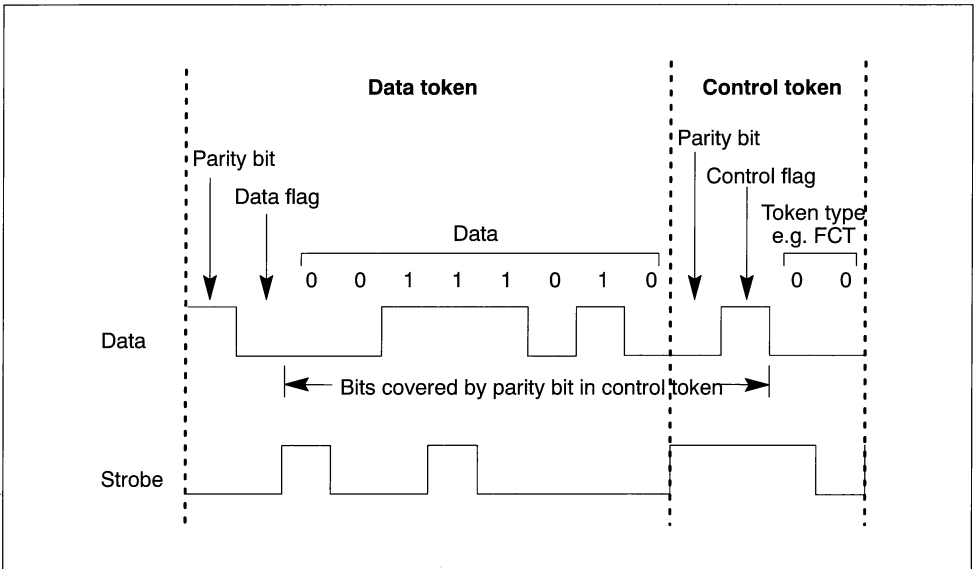


Figure 13.2 Link data and strobe formats

The DS-Link protocol ensures that only one of the two wires of the data strobe pair has an edge in each bit time. The levels on the data wire give the data bits transmitted. The strobe signal changes whenever the data signal does not. These two signals encode a clock together with the data bits, permitting asynchronous detection of the data at the receiving end.

The data and control tokens are of different lengths, for this reason the parity bit in any token covers the parity of the data or control bits in the previous token, and the data/control flag in the same token, as shown in figure 13.2. This allows single bit errors in the token type flag to be detected. **Odd** parity checking is used. Thus the parity bit is set/unset to ensure that the bits covered, inclusive of the parity bit (see figure 13.2), always contain an odd number of 1's. The coding of the tokens is shown in table 13.1.

Token type	Abbreviation	Coding
Data token	–	P0DDDDDDDD
Flow control token	FCT	P100
End of packet	EOP	P101
End of message	EOM	P110
Escape token	ESC	P111

P = parity bit
D = data bit

Table 13.1 Token codings

One of the four control tokens is an escape token. The escape token is used to construct other 'composite' control tokens, see table 13.2. One of these is the null token. Null tokens are sent in the absence of other tokens to ensure the immediate detection of parity errors and to enable link disconnection to be detected.

Another type of composite control token is data alignment tokens (DATs). These are not supported at present, although they will be implemented in future revisions of the chip. Data alignment tokens are sent to compensate for any skew. Although the data and strobe signals follow near identical paths, it is possible in extreme cases for there to be sufficient skew to corrupt the data. To overcome this problem, the DS-Links permit an automatic skew correction system. On start up a burst of 128 DAT tokens are sent to compensate for static skew, subsequently intermittent DATs are sent to compensate for any low frequency dynamic skew.

Note: The skew correction system is not implemented on early silicon, and bit 4 in the **Link0-3Mode** registers (see table 13.5) should be set to 1 to disable DATs. At present DATs are transmitted and decoded but no skew correction takes place.

Token type	Composite token	Coding	Abbreviation	Function
Null token	ESC P100	P111 P100	NUL	Disconnect detection
Data alignment token	ESC P0x1	P111 P0x1	DAT	Skew correction

P = parity bit
 x = 0 or 1

Table 13.2 Composite control tokens

13.2 Link functional description

A link module consists of the units shown in figure 13.3. The link control unit controls: link initialization, speed control, error reporting, etc. It connects to the link registers in the configuration space (refer to section 13.6). The link input unit interprets edge streams as tokens. The link output unit converts tokens to streams of edges.

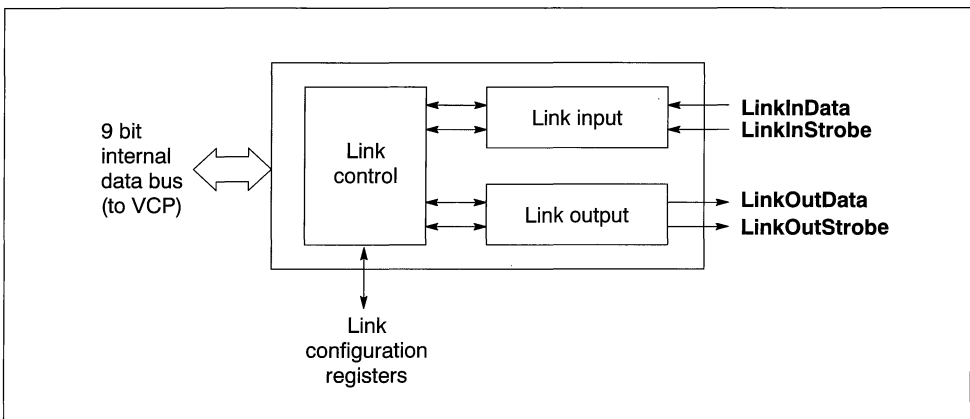


Figure 13.3 Link module

13.3 Low-level flow control

The DS-Link protocol separates the functions of flow control and process synchronization. Flow control is done entirely within the link module and process synchronization is built into the higher-level packet system, which is described in chapter 11, Communications.

The IMS T9000 sends all packets of a particular message down the same DS-Link. After the first packet, subsequent packets are only sent once the previous packet has been acknowledged. If more than one virtual channel is using a link then packets belonging to messages on the different virtual channels are interleaved.

There can only be one message on any channel at one time, and any process can only be communicating on one channel at any time. Incoming packets (data or acknowledgement) for any channel may arrive on any link.

Token-level flow control is performed in each link module, and the additional flow control tokens used are not visible to the higher-level packet protocol. The token-level flow control mechanism prevents a sender from overrunning the input buffer of a receiving link. Each receiving link input contains a buffer for at least 8 tokens (more buffering than this is in fact provided). Whenever the link input has sufficient buffering available to receive a further 8 tokens a FCT is transmitted on the associated link output. The FCT gives the sender permission to transmit a further 8 tokens. Once the sender has transmitted a further 8 tokens it waits until it receives another FCT before transmitting any more tokens. The provision of more than 8 tokens of buffering on each link input ensures that in practice the next FCT is received before the previous block of 8 tokens has been fully transmitted. Thus, the token-level flow control does not restrict the maximum uni-directional bandwidth of the link. The DS-Links of the IMS T9000 are capable of recording the receipt of up to 8 FCTs at any one time.

Note that FCT and ESC/NUL tokens are local to the connection between T9000₁ and T9000₂, whereas data, EOP and EOM are passed to and from logic within T9000₁ and T9000₂.

13.4 Link speed select

The IMS T9000 DS-Links support a range of communication speeds. The speed of a DS-Link is programmed by writing to registers in the configuration space.

Only the transmission speed of a DS-Link is programmed as reception is asynchronous. This means that DS-Links transmitting at different speeds can be connected, provided that each device is capable of receiving at the speed of the connected transmitter.

The transmission speed of all four DS-Links on the IMS T9000 are related to the speed of the 10 MHz base clock. This 10 MHz clock is multiplied by a programmable value to provide the root clock for all the DS-Links. The multiplication factor is programmed by writing to the **SpeedMultiply** bit in the **DSLInkPLL** register in System Services, see section 8.3.4. This root clock is then optionally divided (by programming the **SpeedDivide** bits) by 1, 2, 4 or 8 independently for each DS-Link, giving a range of speeds. This arrangement allows each DS-Link to be run at one of four transmission speeds, as shown in table 13.3.

SpeedMultiply	SpeedDivide1:0				BaseSpeed
	0:0	0:1	1:0	1:1	
	/ 1	/ 2	/ 4	/ 8	
8	80	40	20	10.0	10
10	100	50	25	12.5	10
12	Reserved	60	30	15.0	10
14	Reserved	70	35	17.5	10
16	Reserved	80	40	20.0	10
18	Reserved	90	45	22.5	10
20	Reserved	100	50	25.0	10

Table 13.3 DS-Link transmission speed in Mbits/sec

Note also that each DS-Link can be programmed to use a base rate clock of 10 MHz. At reset all DS-Links are configured to run at the **BaseSpeed** of 10 Mbits/sec. The **SpeedSelect** bit in the **Link0-3Mode** registers when set to 1 sets the respective DS-Link (**Link0-3**) to the speed selected by the **SpeedMultiply** and **SpeedDivide** bits, as opposed to the default base speed of 10 Mbits/s.

13.5 Errors on DS-Links

DS-Link inputs can detect parity and disconnection conditions as errors. A single bit odd parity system will detect single bit errors at the link token level. The protocol to transmit NUL tokens in the absence of other tokens enables disconnection of a DS-Link to be detected. A disconnection error indicates one of two things:

- the DS-Link has been physically disconnected;
- an error has occurred at the other end of the DS-Link, which has then stopped transmitting.

The **LinkError** bit in the **Link0-3Status** registers flags that a parity and/or disconnection error has occurred on the **Link0-3**. The bit fields **ParityError** and **DiscError** indicate when parity and disconnect errors occur respectively.

When a DS-Link detects a parity error on its input it halts its output. This is detected as a disconnect error at the other end of the DS-Link, causing this to halt its output also. Detection of an error causes the DS-Link to be reset. Thus, the disconnect behavior ensures that both ends are reset. Each end can then be re-started.

Note, when one end of a DS-Link is started up before the other end of a DS-Link, a disconnect error does not occur as no tokens have been received and once the other end of the DS-Link is started communication can commence. A disconnect error is only flagged once a token has been received on a DS-Link and transmission is subsequently interrupted.

The DS-Links are designed to be highly reliable within a single subsystem and can be operated in one of two environments, 'reliable' or 'unreliable' determined by the **LocalizeError** bit (set in **Link0-3Mode** register) in each DS-Link. Note that there is also a **LocalizeError** bit in the VCP configuration registers **VCPLink0-3Mode** which when set masks different classes of link errors detected by the VCP (refer to section 11.9). If the link errors are to be masked for a DS-Link the **LocalizeError** bits in both the **VCPLink0-3Mode** register and **Link0-3Mode** register for the DS-Link should be set together.

The **LocalizeError** bit is set on a per link basis, therefore it is possible to have some DS-Links in a system marked as reliable and others as unreliable. The consequence of a link error depends on which environment the DS-Link is in.

13.5.1 Reliable links

In the majority of applications, the communications system should be regarded as being totally reliable. In this environment errors are considered to be very rare, but are treated as being catastrophic if they do occur, requiring the intervention of the control system. This environment is the default on power-on reset, with all DS-Links having their **LocalizeError** bit set to 0. If an error occurs it will be detected and reported via a message sent along **CLink0**, or will cause a reset and reboot if in stand-alone mode. The CPU and VCP of the IMS T9000 will be halted. Normal practice will then be to reset the subsystem in which the error has occurred and to restart the application.

Handling of errors on reliable links

- An error is reported from both devices connected by the DS-Link and the location of the error identified.
- The DS-Link halts and stops transmitting on its output. All communication on the DS-Link is frozen until both ends are restarted. Packets in transit at the time will be stopped, although headers may propagate through a network, blocking other communication through the network.
- The VCP goes into the 'discarding' state (see page 203).
- The CPU is stopped enabling debugging to be performed.
- An error message is sent on control link0.
- An external control process (root process) analyses the status of the network and records any error information.
- The system is reset and restarted, by the external control process, via *CPoke* commands down **CLink0**.

Alternatively, to avoid the need to explicitly reset the network the error can be localized to the link. Localizing the error and resetting the link causes any truncated packets to be terminated thus clearing the blockage in the network. Note, this can cause a 'packet too short' error to be generated by a subsequent node, in this case the truncated packet is discarded. Alternatively it may cause a process to be incorrectly rescheduled, this can be avoided by ensuring the receiving transputer is stopped first. Before the links are localized, reset and restarted, the *Stop* command should be sent to all the T9000's so that any error messages resulting from the truncation of packets will be distinguishable from the errors which occur before the processes are stopped. Thus ensuring no processes will be rescheduled with invalid or incomplete data.

13.5.2 Unreliable links

For some applications, for instance when a disconnect or parity error may be expected during normal operation, an even higher level of reliability is required. This level of fault tolerance is supported by localizing errors to the link on which they occur. This is achieved by setting the **LocalizeError** bit in the **Link0-3Mode** register to 1. When the **LocalizeError** bit is set, errors are no longer reported to the control link and consequently do not result in stopping the CPU. In this mode the link is considered 'unreliable'.

When in unreliable mode, processes must communicate using defensive software which can detect errors at the message level. These processes are responsible for establishing and maintaining a higher level flow control, using time-out to detect that a message has not completed, and requesting re-transmission. If an error occurs, packets in transit at the time of the error will be discarded or truncated, and the link will be reset without the error being reported via the control link. Code to implement error recovery must be run on each virtual channel. This application software is provided for the user in libraries contained in INMOS toolset products.

A link error in unreliable mode results solely in packets in transit at the time of the error being discarded or truncated.

13.6 Link configuration registers

The DS-Links (in common with a number of other sub-systems of the IMS T9000) are controlled via a separate configuration address space. The registers in this address space are accessed via the *ldconf* and *stconf* instructions, or via *CPeek* and *CPoke* command messages received along **CLink0**.

The tables below describe the functionality of the DS-Links to be controlled, and the associated bit fields in the configuration registers. All INMOS reserved bits should be set to 0.

Each DS-Link has four registers, the **LinkMode** register, **LinkCommand** register, **LinkStatus** register and **LinkWriteLock** register.

In addition the configuration space contains the **DSLlinkPLL** register which contains the **SpeedMultiply** field. This determines the multiplication factor for the 10 MHz clock, to provide the root clock for all the DS-Links.

DSLlinkPLL		#1005	Read/Write
Bit	Bit field	Function	
5:0	SpeedMultiply	Sets DS-Link master clock to required value (see table 13.3). Note , 0 through to 7 are invalid values and should not be used.	
31:6		INMOS reserved	

Table 13.4 Bit fields in the **DSLlinkPLL** register

The **Link0Mode**, **Link1Mode**, **Link2Mode**, **Link3Mode** registers (one for each of the four DS-Links) power up into a default state and may be re-programmed before or after the DS-Link has been started.

Note also that each DS-Link can be programmed to use a base rate clock of 10 MHz. At reset all DS-Links are configured to run at the **BaseSpeed** of 10 Mbits/sec. The **SpeedSelect** bit in the **Link0-3Mode** registers when set to 1 sets the respective DS-Link (**Link0-3**) to the speed selected by the **SpeedMultiply** and **SpeedDivide** bits, as opposed to the default base speed of 10 Mbits/s.

Link0-3Mode0-3		#8001, #8101, #8201, #8301	Read/Write
Bit	Bit field	Function	
1:0	SpeedDivide	The master link clock frequency is divided down by setting the SpeedDivide bits to obtain the transmission frequency for each DS-Link. Sets the transmit speed of the DS-Link (see table 13.3). 00 = / 1, 01 = / 2, 10 = / 4, 11 = / 8	
2	SpeedSelect	Sets the DS-Link to transmit at the speed determined by the SpeedDivide bits as opposed to the base speed of 10 Mbits/s.	
3	LocalizeError	When set errors detected by the DS-Link are no longer reported to the control unit. Packets in transit at the time of an error will be discarded or truncated.	
4	DisableDATs	When set to 1, disables the transmission of Data Alignment Tokens (DATs), otherwise DATs are sent every 1024 bit periods.	
31:5		INMOS reserved	

Table 13.5 Bit fields in the **Link0-3Mode** registers

The **Link0Command**, **Link1Command**, **Link2Command**, **Link3Command** registers are write only and contain four bits which when set cause a specific action to be taken by the DS-Link.

Link0-3Command		#8002, #8102, #8202, #8302	Write only
Bit	Bit field	Function	
0	ResetLink	Resets the link engine of the Link0-3 . The token state is reset, the flow control credit is set to zero, the buffers are marked as empty, and the parity state is reset. It also resets the LinkStatus register error bits.	
1	StartLink	When a transition from 0 to 1 occurs Link0-3 will be initialized and commence operation.	
2	ResetOutput	Sets both outputs of Link0-3 low.	
3	WrongParity	The Link0-3 output will generate incorrect parity. This may be used to force a parity error on the transputer at the other end of the Link0-3 .	
31:4		INMOS reserved	

Table 13.6 Bit fields in the **Link0-3Command** registers

The **Link0Status**, **Link1Status**, **Link2Status**, **Link3Status** registers are read only and contain six bits which contain information about the state of the DS-Link.

Link0-3Status		#8003, #8103, #8203, #8303	Read only
Bit	Bit field	Function	
0	LinkError	Flags that an error has occurred on the Link0-3 .	
1	LinkStarted	Flags that the output Link0-3 has been started and no errors have been detected.	
2	ResetOutputComplete	Flags that ResetOutput has completed on the Link0-3 .	
3	ParityError	Flags that a parity error has occurred on the Link0-3 .	
4	DiscError	Flags that a disconnect error has occurred on the Link0-3 .	
5	TokenReceived	Flags that a token has been seen on the Link0-3 since ResetLink .	
31:6		INMOS reserved	

Table 13.7 Bit fields in the **Link0-3Status** registers

In addition to the above mentioned registers each DS-Link has an associated **LinkWriteLock** register. The write lock register provides protection from writes by the processor, so that the link is guaranteed to function regardless of program behavior.

Setting the write lock register inhibits all configuration writes to this link from the CPU. All control link reset commands clear all write locks.

Note, the CPU can start and reset the links by means of *resetch* and *setchmode* instructions even if the write lock is set.

Link0-3WriteLock		#8004, #8104, #8204, #8304	Read/Write
Bit	Bit field	Function	
0	WriteLock	When set to 1 it inhibits link configuration writes from the CPU.	

Table 13.8 Bit fields in the **Link0-3WriteLock** registers

13.7 Initialization

13.7.1 Link state on start up

After power-on all **LinkData** and **LinkStrobe** signals are low, without clocks. Following power-on reset an initialization sequence sets the speed of the link clock. The DS-Links are initially inactive, with a default configuration. They are configured and started by configuration writes. Their status can be determined by configuration reads. Each DS-Link (**Link0-3**) must be explicitly started by writing to the **Link0-3Command** registers respectively. When a DS-Link is started up it transmits control tokens.

Data may not be transferred over the link until the receiving link has sent a FCT to signify that it has enough free buffer space to receive the data. The data/strobe outputs are held low until the first FCT is sent.

The receiving link receives and correctly decodes the tokens. However, only when the receiving link has been explicitly started by writing across the (internal) configuration bus can it send tokens back. Figure 13.4 gives the sequence of initial tokens sent on start-up, when the DS-Link is configured to run at the base speed. NUL tokens are then sent until data is required. If the DS-Link is not configured to run at the base speed, these initial tokens are preceded by a burst of 128 DATs.

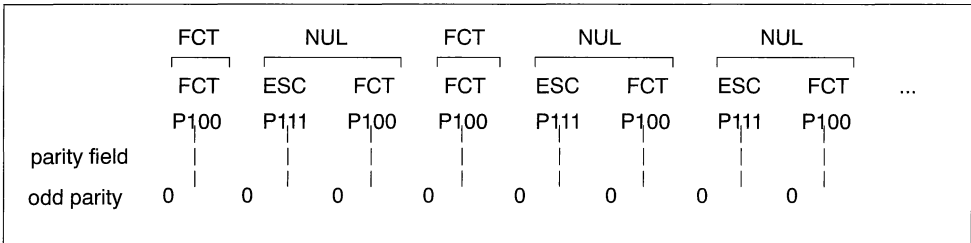


Figure 13.4 Sequence of tokens sent on start-up

13.7.2 Link state following reset

Both **LinkData** and **LinkStrobe** are low following reset to level 0 or level 1. The DS-Links output pins are unchanged after a reset to level 2. Note that a particular DS-Link can be explicitly reset via the configuration bus.

The control logic is responsible for resetting the link input, output and skew alignment logic following an error, however it does not reset the configuration registers, or the Data and Strobe outputs. These may all be reset independently via the configuration registers.

Physical links can be reset in one of the following ways:

- A *resetch* (reset channel) instruction whose channel parameter designates a byte-stream channel. This resets the queue registers for the link and resets and restarts the link.
- A *setchmode* (set channel mode) instruction with a parameter of 0, or by setting the **ResetLink** bit in the **Link0-3Command** configuration register via the configuration bus. The token state is reset, the flow control credit is set to zero, the buffers are marked as empty, and the parity state is reset.
- VCP global reset. This resets all four links.

The link automatically enforces a delay between reset and restart sufficient to guarantee that the other end will detect a disconnect error and complete its reset.

13.8 Link connections

DS-Links are not synchronized with **ClockIn** or **ProcClockOut** and are insensitive to their phases. Thus, links from independently clocked systems may communicate.

DS-Links are TTL compatible and intended to be used in electrically quiet environments, between devices on a single printed circuit board or between two boards via a backplane. Direct connection may be made between devices separated by a distance of less than 300 millimeters. For longer distances a matched 100 ohm transmission line should be used, see figure 13.5.

The inputs and outputs have been designed to have minimum skew at the 1.5 V TTL threshold.

Buffers may be used for very long transmissions. If so, their overall propagation delay should be stable within the skew tolerance of the link, although the absolute value of the delay is immaterial.

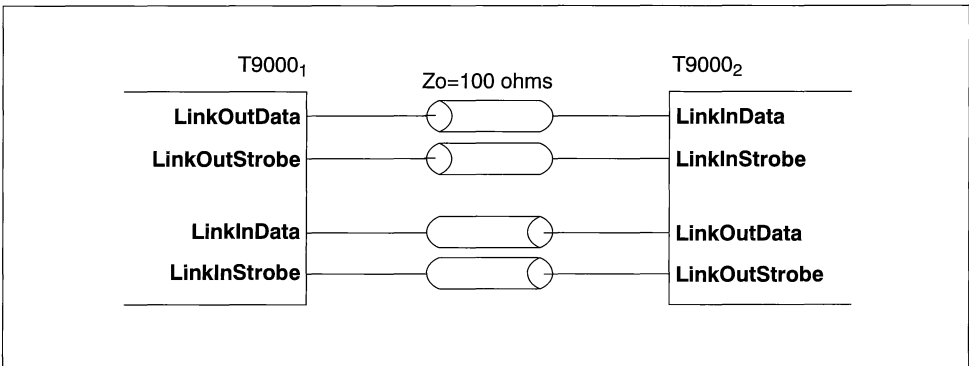


Figure 13.5 DS-Links connected by transmission line

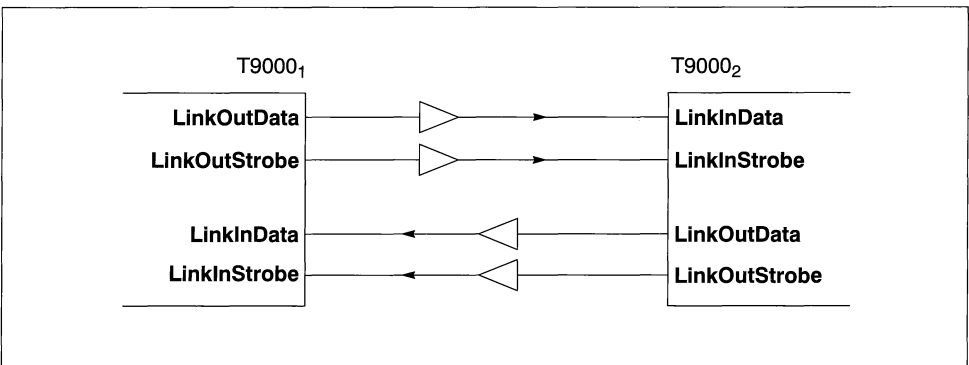


Figure 13.6 DS-Links connected by buffers

13.9 DS-Link timings

The following AC timing characteristics are based on simulations of the IMS T9000 chip, and may change when full characterization is completed.

Symbol	Parameter	Min	Nom	Max	Units
tLODSr	LinkOut rise time		4		
tLODSf	LinkOut fall time		4		
tLIDSr	LinkIn rise time		4		
tLIDSf	LinkIn fall time		4		
tLIHL	Input edge resolution	2			ns
tDSDS	Bit period	10		100	ns
Δt_{DSO}	Data / strobe output skew			1	ns
CLIZ	LinkIn capacitance		7		pF

Table 13.9 DS-Link timings

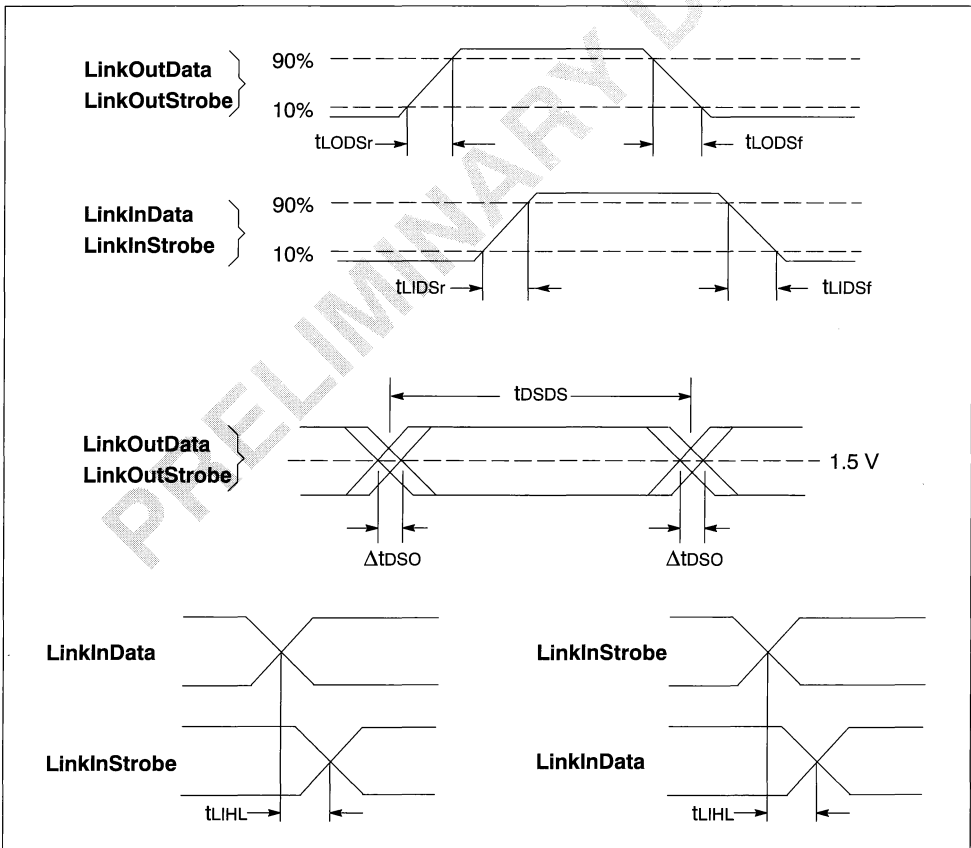


Figure 13.7 DS-Link timing

14 Clocking phase locked loops

Two on-chip phase locked loops (PLL) generate all the internal high frequency clocks from a single clock input, simplifying system design and avoiding problems of distributing high speed clocks externally. This chapter details the PLL input specifications, decoupling requirements and speed selections.

There is one PLL for the system clocks and one for the link clocks. Each PLL has six outputs, four quad phase signals and two bi-phase signals. The global system clocks use the four quad phase signals from the system clock PLL. The link clock is one of the bi-phase 100 MHz clocks from the link PLL.

14.1 Clock input

The high frequency internal clocks are derived from the clock frequency supplied by the user. The user supplies the clock frequency for input to the PLL's via the **ClockIn** input. The nominal frequency of this clock for all transputer family components is 5 MHz, regardless of device type, transputer word length or processor cycle time.

A number of transputer devices may be connected to a common clock, or may have individual clocks providing each one meets the specified stability criteria. In a multi-clock system the relative phasing of **ClockIn** clocks is not important, due to the asynchronous nature of the links. Mark/space ratio is unimportant provided the specified limits of **ClockIn** pulse widths are met.

Oscillator stability is important. **ClockIn** must be derived from a crystal oscillator; RC oscillators are not sufficiently stable. **ClockIn** must not be distributed through a long chain of buffers. Clock edges must be monotonic and remain within the specified voltage and time limits.

The timing requirements for **ClockIn** are given in section 14.5.

14.2 PLL decoupling

The PLL's require a decoupled power supply for satisfactory operation. The decoupling is performed externally by connecting a $1\mu\text{F}$ ceramic capacitor between the **CapPlus** and **CapMinus** pins on the chip. A surface mounted ceramic capacitor with an ESR (Equivalent Series Resistance) of less than 3 ohms should be used. In order to keep stray inductances low, the total PCB track length should be less than 20 mm, thus the capacitor should be no more than 10 mm from the chip. The connections must not touch power supplies or other noise sources.

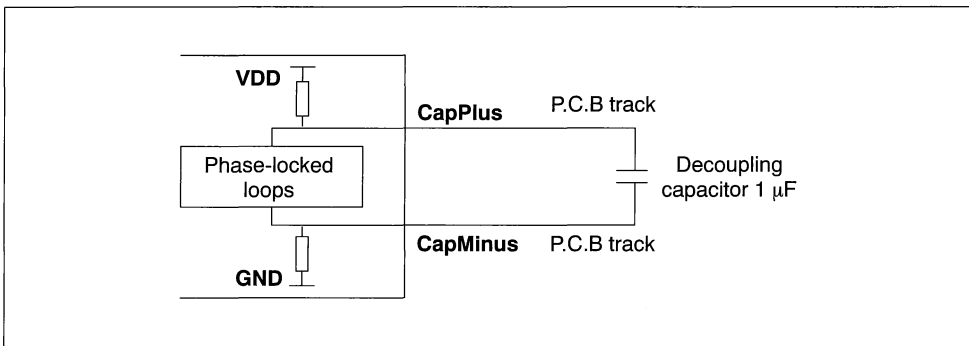


Figure 14.1 Recommended PLL decoupling

14.3 Processor speed selection

The processor internal clock rate is variable in discrete steps. The clock rate at which the IMS T9000 runs at is determined by the logic levels applied on the three speed select lines **ProcSpeedSelect0-2** as detailed in table 14.1. Note that the processor cycle time given in table 14.1 is a nominal value; it can be calculated more accurately using the phase lock loop factor **PLLx** (see section 14.4 below).

ProcSpeed Select2	ProcSpeed Select1	ProcSpeed Select0	Processor clock speed MHz	Processor cycle time ns	Phase lock loop factor (PLLx)
0	0	0	30	33.3	6.0
0	0	1	35	28.6	7.0
0	1	0	40	25.0	8.0
0	1	1	45	22.2	9.0
1	0	0	50	20.0	10.0
1	0	1	INMOS reserved		
1	1	0	INMOS reserved		
1	1	1	INMOS reserved		

Table 14.1 Processor speed selection

14.4 Processor clock output

The processor output clock (supplied on the **ProcClockOut** output) is derived from the internal processor clock, which is in turn derived from **ClockIn**. It provides an output timing signal at the rated clock frequency of the device.

Its period is equal to one internal microcode cycle time, and can be derived from the formula

$$t_{PCLPCL} = t_{DCLDCL} / PLLx$$

where **tPCLPCL** is the **ProcClockOut Period**, **tDCLDCL** is the **ClockIn Period** and **PLLx** is the phase lock loop factor for the relevant speed part.

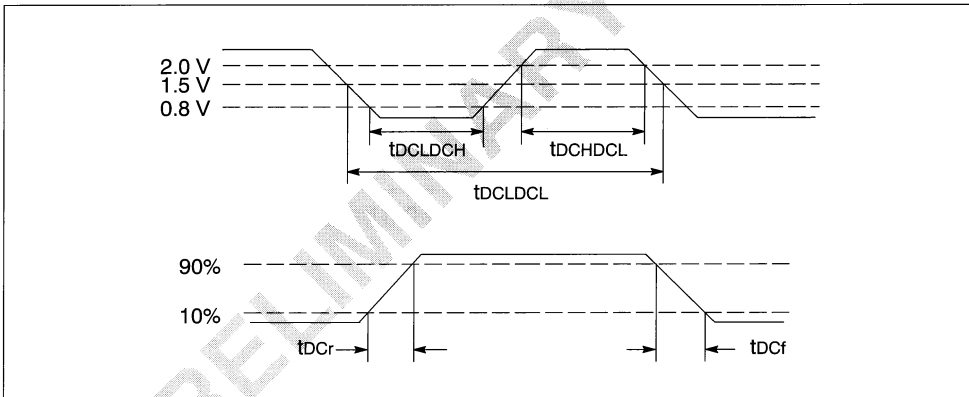
14.5 ClockIn timings

Symbol	Parameter	Min	Nom	Max	Units	Notes
tDCLDCH	ClockIn pulse width low	40			ns	
tDCHDCL	ClockIn pulse width high	40			ns	
tDCLDCL	ClockIn period		200		ns	1, 2
tDCr	ClockIn rise time			10	ns	3
tDCf	ClockIn fall time			8	ns	3

Table 14.2 **ClockIn** timings

Notes

- 1 Measured between corresponding points on consecutive falling edges.
- 2 This value allows the use of 200 ppm crystal oscillators for two devices connected together by a link.
- 3 Clock transitions must be monotonic within the range V_{IH} to V_{IL} (refer to Electrical specifications chapter).

Figure 14.2 **ClockIn** timing

14.6 ProcClockOut timings

The following timings are based on simulations of the 50 MHz version of the IMS T9000 chip, and may change when full characterization is completed. The simulations were run under a range of output loads from 10 pF to 70 pF (capacitive only).

Symbol	Parameter	Min	Nom	Max	Units	Notes
tPCLPCL	ProcClockOut period		20		ns	
tPCHPCL	ProcClockOut pulse width high	9.0	10	11.4	ns	
tPCLPCH	ProcClockOut pulse width low	8.6	10	11.0	ns	
tPCstab	ProcClockOut stability				%	1

Table 14.3 **ProcClockOut** timings

Notes

- 1 Stability is the variation of cycle periods between two consecutive cycles, measured at corresponding points on the cycles.

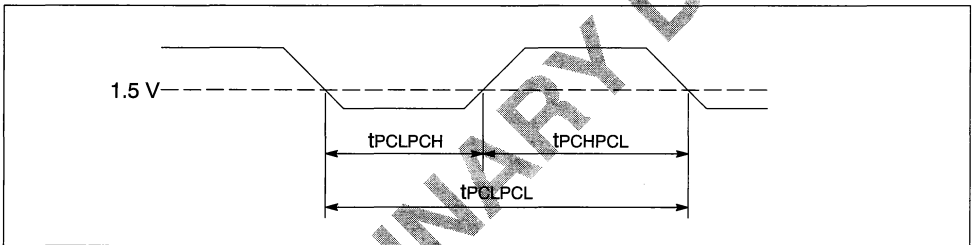


Figure 14.3 **ProcClockOut** timing

15 Configuration register reference guide

The following chapter lists the full set of configuration registers for each of the sub-systems of the IMS T9000 transputer and gives the addresses of the registers in the configuration space. The complete bit format of each of the registers and its functionality is given in the relevant sub-system chapter.

15.1 Configuration bus

The IMS T9000 transputer has several sub-systems which need to be initialized on bootstrap. This initialization is performed via the configuration bus.

The configuration bus is an internal serial bus that allows restricted sharing of certain registers, referred to as configuration registers, between the subsystems or units of the IMS T9000.

There are two bus masters: the CPU; and the control unit. Any of the subsystems may be bus slaves. The bus gives the bus masters (CPU and control unit) restricted read and/or write access to all the configuration registers. None of the other subsystems can write to a configuration register via the bus.

The configuration bus has full address decoding and any invalid address will be reported as an error.

15.2 Subsystem addresses

The registers in the configuration space are accessed via the *ldconf* (load from configuration register) and *stconf* (store to configuration register) instructions, or via *CPeek* and *CPoke* command messages received along **CLink0**. A 2 byte 16 bit address is issued, the most significant byte refers to the subsystem, the least significant byte refers to the local register within the subsystem.

There are 13 subsystems which access the configuration space. The subsystems and their assigned addresses are listed in table 15.1 below.

Subsystem	A15	A14	A13	A12	A11	A10	A9	A8	Hex
CPU	0	0	0	0	0	0	0	1	#01
PMI bank address	0	0	0	0	0	0	1	0	#02
PMI strobe timing	0	0	0	0	0	1	0	0	#04
VCP	0	0	0	0	1	0	0	0	#08
System services	0	0	0	1	0	0	0	0	#10
Cache	0	0	1	0	0	0	0	0	#20
Scheduler	0	1	0	0	0	0	0	0	#40
Link0	1	0	0	0	0	0	0	0	#80
Link1	1	0	0	0	0	0	0	1	#81
Link2	1	0	0	0	0	0	1	0	#82
Link3	1	0	0	0	0	0	1	1	#83
Control link0	1	1	1	1	1	1	0	1	#FD
Control link1	1	1	1	1	1	1	1	0	#FE

Table 15.1 Subsystem addresses

15.2.1 Shared registers

There are a number of registers which are effectively shared between two or more slave subsystems. This is implemented by having a separate copy of each shared register in each slave subsystem which uses

it. The addressing system allows all the copies to be addressed simultaneously, using a unique 'subsystem set' address, see table 15.2. The shared address referring to the set of subsystems should be used when writing (poking) to those registers and when reading (peeking) from those registers. Writing to a single copy causes an address error. These shared registers can only be modified via the configuration bus, thus ensuring all copies hold the same value.

Subsystem set	A15	A14	A13	A12	A11	A10	A9	A8	Hex
PMI bank and strobe	0	0	0	0	0	1	1	0	#06
CPU and VCP	0	0	0	0	1	0	0	1	#09
CPU and Scheduler	0	1	0	0	0	0	0	1	#41
CPU, VCP and Scheduler	0	1	0	0	1	0	0	1	#49

Table 15.2 Subsystem set addresses

15.3 CPU write locking

Every subsystem except the cache includes a 1 bit register called a write-lock. If this register is set to 1 the effect is to prevent all writes from the CPU to **any** register in that subsystem including the write-lock. Reads are not affected nor are writes by the control unit.

Reset clears the write lock registers.

There are two write-lock registers which are shared between subsystems and therefore if set prevent writes from the CPU to any register in either subsystem. These are the **PMIWriteLock** register and the **ChanWriteLock** register. The **PMIWriteLock** register prevents the CPU from writing to any of the PMI strobe timing and PMI bank address registers. The **ChanWriteLock** register prevents the CPU from writing to: all CPU registers; all VCP registers; and all scheduler registers.

15.4 Subsystem registers

The following tables list all the registers in each subsystem and their addresses. For shared registers the **Shared with:** column indicates which subsystem the register is shared with. The shared address is given for registers which are shared between subsystems as this is the address which should be used to write to or read from the register. The **Read/Write** column indicates whether the register is read-only (R), write-only (W), or read-write (R/W).

All registers are 32 bits long, and 32 bits are always read or written. The column labelled **Bit size** indicates the number of valid bits in the register. A register is listed as having 30 bits when it is used to contain a word-aligned pointer, in this case only the upper 30 bits (bits 31:2) of the register are significant. The byte selector (bits 1:0) are not significant. For registers listed as less than 30 bits, the lowest bits in the register are significant. This is shown in the **Significant bits** column. When writing to a register, any non-significant bits must always be zero. When reading from a register, any non-significant bits are undefined. For registers listed as containing 0 bits, any write to the corresponding address causes some action to occur.

15.4.1 CPU configuration registers

Register	Address	Bit size	Significant bits	Read/Write	Shared with:
ChanWriteLock	#4900	1	0	R/W	VCP and scheduler
HdrAreaBase	#0901	30	2-31	R/W	VCP
MemStart	#410E	30	2-31	R/W	scheduler
MinInvalidChannel	#010F	30	2-31	R/W	—
ExternalRCBase	#4110	30	2-31	R/W	scheduler
InitialIptr	#0111	32	0-31	R/W	—
InitialWptr	#0112	30	2-31	R/W	—
Reason	#0113	32	0-31	R/W	—
EmiBadAddress	#0115	32	0-31	R/W	—

Table 15.3 CPU configuration registers

15.4.2 PMI configuration registers

PMI bank address configuration registers

Register	Address	Bit size	Significant bits	Read/Write	Shared with:
PMIWriteLock	#0600	1	0	R/W	PMI strobe timing
Address0	#0202	30	2-31	R/W	—
Mask0	#0203	30	2-31	R/W	—
FormatControl0	#0204	30	2-31	R/W	—
Address1	#0205	30	2-31	R/W	—
Mask1	#0206	30	2-31	R/W	—
FormatControl1	#0207	30	2-31	R/W	—
Address2	#0208	30	2-31	R/W	—
Mask2	#0209	30	2-31	R/W	—
FormatControl2	#020A	30	2-31	R/W	—
Address3	#020B	30	2-31	R/W	—
Mask3	#020C	30	2-31	R/W	—
FormatControl3	#020D	30	2-31	R/W	—
RASBits0	#020E	30	2-31	R/W	—
RASBits1	#020F	30	2-31	R/W	—
RASBits2	#0210	30	2-31	R/W	—
RASBits3	#0211	30	2-31	R/W	—
DoPMIConfigured	#0212	0	—	W	—
ErrorAddress	#0213	30	2-31	R	—

Table 15.4 PMI bank address configuration registers

PMI strobe timing configuration registers

Register	Address	Bit size	Significant bits	Read/Write	Shared with:
PMIWriteLock	#0600	1	0	R/W	PMI bank address
RASStrobe0	#0401	30	2-31	R/W	—
RASStrobe1	#0402	30	2-31	R/W	—
RASStrobe2	#0403	30	2-31	R/W	—
RASStrobe3	#0404	30	2-31	R/W	—
CASStrobe0	#0405	30	2-31	R/W	—
CASStrobe1	#0406	30	2-31	R/W	—
CASStrobe2	#0407	30	2-31	R/W	—
CASStrobe3	#0408	30	2-31	R/W	—
ProgStrobe0	#0409	30	2-31	R/W	—
ProgStrobe1	#040A	30	2-31	R/W	—
ProgStrobe2	#040B	30	2-31	R/W	—
ProgStrobe3	#040C	30	2-31	R/W	—
WriteStrobe0	#040D	30	2-31	R/W	—
WriteStrobe1	#040E	30	2-31	R/W	—
WriteStrobe2	#040F	30	2-31	R/W	—
WriteStrobe3	#0410	30	2-31	R/W	—
TimingControl0	#0411	30	2-31	R/W	—
TimingControl1	#0412	30	2-31	R/W	—
TimingControl2	#0413	30	2-31	R/W	—
TimingControl3	#0414	30	2-31	R/W	—
RefreshControl	#0415	30	2-31	R/W	—
RemapBootBank	#0416	0	—	W	—

Table 15.5 PMI strobe timing configuration registers

15.4.3 VCP configuration registers

Register	Address	Bit size	Significant bits	Read/Write	Shared with:
ChanWriteLock	#4900	1	0	R/W	CPU and scheduler
HdrAreaBase	#0901	30	2-31	R/W	CPU
VCPStatus	#0802	32	0-31	R/W	—
VCPCommand	#0804	3	0-2	R/W	—
VCPLink0Mode	#0805	3	0-2	R/W	—
VCPLink0MaxHeader	#0806	16	0-15	R/W	—
VCPLink0MinHeader	#0807	16	0-15	R/W	—
VCPLink0HdrOffset	#0808	16	0-15	R/W	—
VCPLink1Mode	#0809	3	0-2	R/W	—
VCPLink1MaxHeader	#080A	16	0-15	R/W	—
VCPLink1MinHeader	#080B	16	0-15	R/W	—
VCPLink1HdrOffset	#080C	16	0-15	R/W	—
VCPLink2Mode	#080D	3	0-2	R/W	—
VCPLink2MaxHeader	#080E	16	0-15	R/W	—
VCPLink2MinHeader	#080F	16	0-15	R/W	—
VCPLink2HdrOffset	#0810	16	0-15	R/W	—
VCPLink3Mode	#0811	3	0-2	R/W	—
VCPLink3MaxHeader	#0812	16	0-15	R/W	—
VCPLink3MinHeader	#0813	16	0-15	R/W	—
VCPLink3HdrOffset	#0814	16	0-15	R/W	—

Table 15.6 VCP configuration registers

15.4.4 System services configuration registers

Register	Address	Bit size	Significant bits	Read/Write
DeviceID	#1001	16	0-15	R
DeviceRevision	#1002	16	0-15	R
ModeStatus	#1003	2	0-1	R/W
ErrorCode	#1004	8	0-7	R
DSLlinkPLL	#1005	6	0-5	R/W
SysServWriteLock	#1006	1	0	R/W

Table 15.7 System services configuration registers

15.4.5 Cache configuration registers

Register	Address	Bit size	Significant bits	Read/Write
RamSize	#2001	2	0-1	R/W
DoRamSize	#2002	0	–	W
RamLineNumber	#2003	8	0-7	R/W
RamAddress	#2004	32	0-31	R/W
DoAllocate	#2005	0	–	W

Table 15.8 Cache configuration registers

15.4.6 Scheduler configuration registers

Register	Address	Bit size	Significant bits	Read/Write	Shared with:
ChanWriteLock	#4900	1	0	R/W	CPU and VCP
MemStart	#410E	30	2-31	R/W	CPU
ExternalRCBase	#4110	30	2-31	R/W	CPU

Table 15.9 Scheduler configuration registers

15.4.7 Link configuration registers

Register	Address	Bit size	Significant bits	Read/Write
Link0Mode	#8001	5	0-4	R/W
Link0Command	#8002	4	0-3	W
Link0Status	#8003	6	0-5	R
Link0WriteLock	#8004	1	0	R/W
Link1Mode	#8101	5	0-4	R/W
Link1Command	#8102	4	0-3	W
Link1Status	#8103	6	0-5	R
Link1WriteLock	#8104	1	0	R/W
Link2Mode	#8201	5	0-4	R/W
Link2Command	#8202	4	0-3	W
Link2Status	#8203	6	0-5	R
Link2WriteLock	#8204	1	0	R/W
Link3Mode	#8301	5	0-4	R/W
Link3Command	#8302	4	0-3	W
Link3Status	#8303	6	0-5	R
Link3WriteLock	#8304	1	0	R/W

Table 15.10 Link0-3 configuration registers

15.4.8 Control link configuration registers

Register	Address	Bit size	Significant bits	Read/Write
CLink0Mode	#FD01	5	0-4	R/W
CLink0Command	#FD02	4	0-3	W
CLink0Status	#FD03	6	0-5	R
CLink0WriteLock	#FD04	1	0	R/W
CLink1Mode	#FE01	5	0-4	R/W
CLink1Command	#FE02	4	0-3	W
CLink1Status	#FE03	6	0-5	R
CLink1WriteLock	#FE04	1	0	R/W

Table 15.11 **CLink0-1** configuration registers

16 Package specifications

The IMS T9000 is available in a 208 pin ceramic leaded chip carrier (CLCC) package. It is intended for cavity down assembly. The dimensions and thermal characteristics detailed below apply to cavity down installation.

16.1 208 pin CLCC package pinout

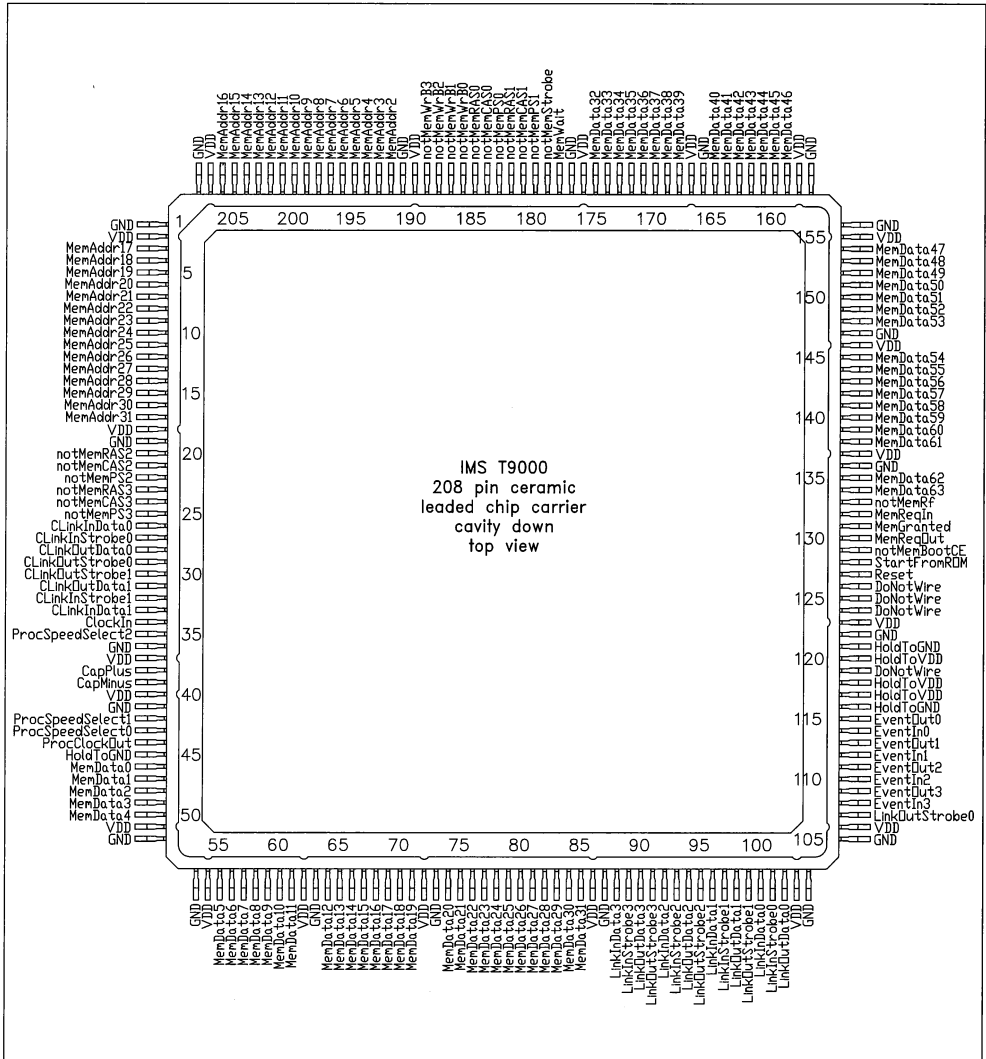


Figure 16.1 208 pin CLCC package pinout

16.2 208 pin CLCC package dimensions

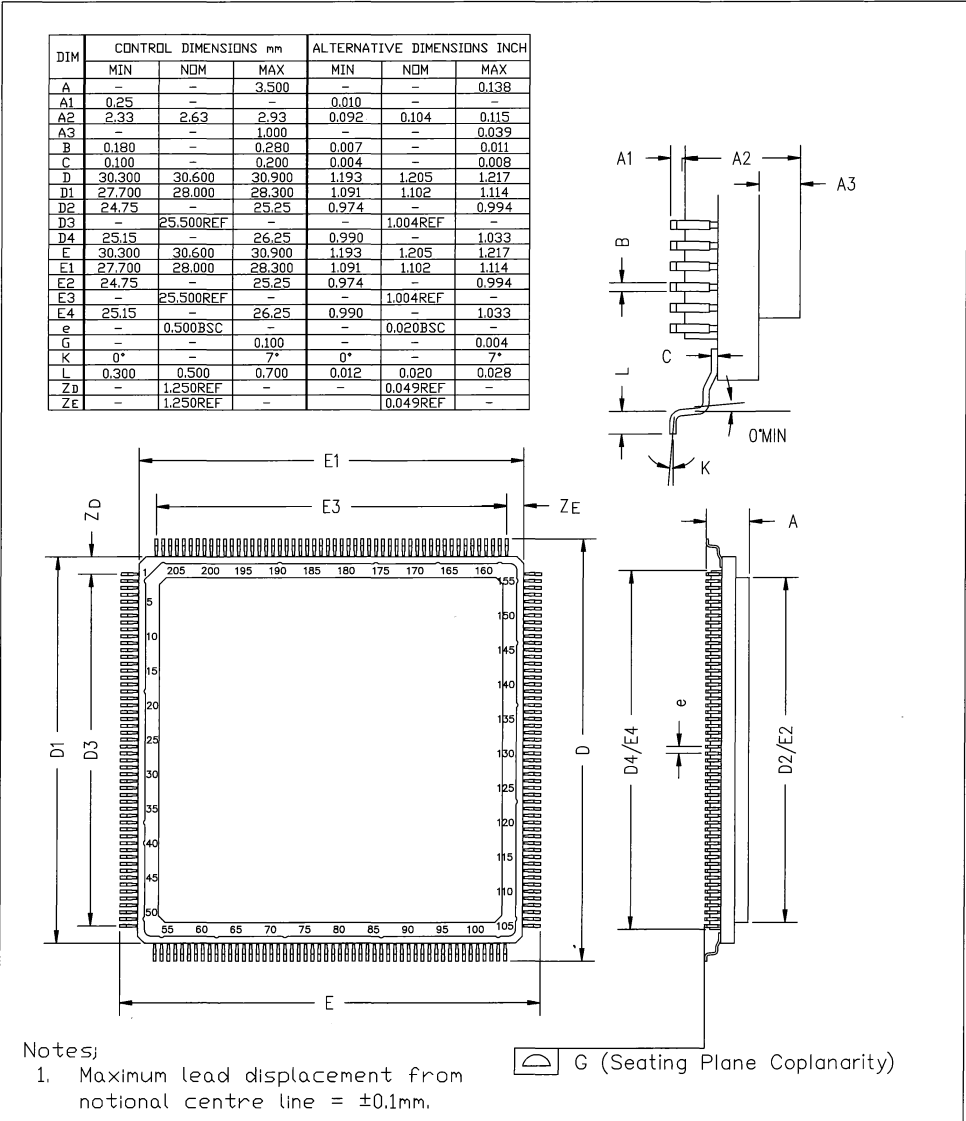


Figure 16.2 208 pin CLCC package dimensions

16.3 208 pin CLCC package thermal characteristics

The junction to case thermal resistance (θ_{JC}) of the package is given below. Information on thermal management of the IMS T9000 is given in chapter 17.

Symbol	Parameter	Min	Nom	Max	Units
θ_{JC}	Junction to case thermal resistance			1	°C/W

Table 16.1 Thermal resistance

17 Thermal management

In order to achieve the specified IMS T9000 device speed and performance, it is necessary to optimize the method of heat removal. The method of heat removal should be chosen taking into account system requirements. The die temperature will increase as a function of power dissipation and duty cycle of the external memory interface. As specified in table 16.1 the junction to case thermal resistance of the 208 pin CLCC package is 1 °C/W.

Table 17.1 gives the maximum ambient temperature at which device performance is guaranteed when operating under continuous loading for different power dissipations.

Power (Watts)	Heat spreader temperature ‡ (°C)	Still air temperature (°C)
3	97	35.5
4	96	16
5	95	-4
6	94	-23
7	93	-42
8	92	-61

‡ Case temperature at center of heat spreader

Table 17.1 Device thermal characteristics in still air for various power dissipations

The 208 pin CLCC package is a cavity down package, thus heat can be dissipated at the package surface from the heat spreader.

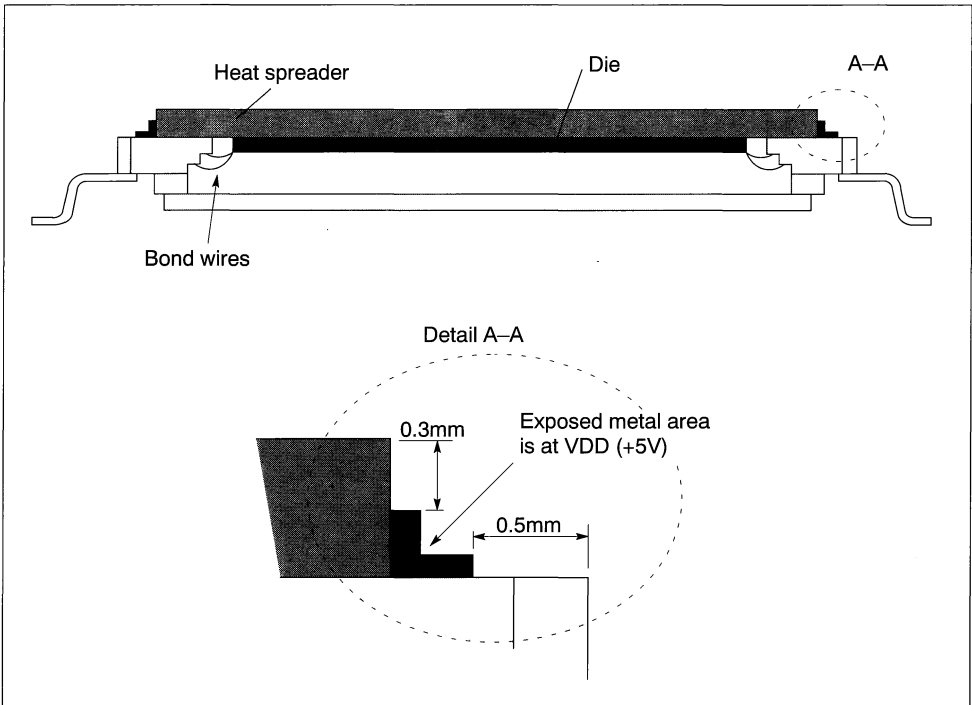


Figure 17.1 Cross-section of package showing die and heat spreader

17.1 Forced air flow cooling

The maximum operating ambient temperature can be increased with the use of forced air flow cooling. Table 17.2 gives the maximum allowable air temperature for various air flows when 7 W of power is being dissipated from within the package.

Power (Watts)	Air velocity (m/sec)	Air temperature (°C)
7	1	16
7	1.5	30
7	2	38
7	2.5	47.5
7	3	51
7	3.5	52.4

Table 17.2 Maximum air temperatures for forced air flow cooling

17.2 Heat sinks

To enable the IMS T9000 device to be used at higher ambient temperatures, further methods of heat removal, for example heat sinks, must be used. Heat sinks are available in various forms from external sources (for example, Redpoint, Thermalloy, EG&G Wakefield). Small light-weight heat sinks may be attached directly to the heat spreader on the upper surface of the package, after soldering to the board. Where larger heat sinks are necessary, these should be affixed to the board after applying a flexible, thermally conductive material to the underside of the heat sink.

WARNING

The exposed metal area around the heat spreader is at +5 V (VDD). If electrically conductive material is used to connect the heat sink it should not run over the edge of the package, see figure 17.1.

Table 17.3 shows typical air temperatures, when forced air flow cooling and a heat sink is used, for various types of heat sink.

Power (Watts)	Air velocity (m/s)	Air temperature (°C)			
		Heat sink 1 †	Heat sink 2	Heat sink 3	Heat sink 4
7	1	2	25	56	66
7	1.5	9.5	39	67	73
7	2	37	49	72	78
7	2.5	47.5	60	75	81
7	3	54.5	65	77	83
7	3.5	65	68	79	84

Table 17.3 Typical air temperatures for four types of heat sink each used in conjunction with forced air cooling

Where,

Heat sink 1 is a 28 mm diameter circular aluminium heat sink with 4 horizontal fins at 2 mm pitch.
 Heat sink 2 is a 38 mm diameter circular aluminium heat sink with 7 horizontal fins at 2 mm pitch.
 Heat sink 3 is a 45 mm square aluminium heat sink with vertical pins 10mm high at 4 mm pitch.
 Heat sink 4 is a 38 mm square aluminium heat sink with vertical fins 25 mm high at 1.3 mm pitch and air flow parallel to fins.

† Note that heat sink 1 has worse performance at air velocities of less than 2.5 m/s than if only air flow cooling is used.

As shown above a reasonable operating ambient temperature range can be achieved using a heat sink and/or forced air flow cooling.

17.3 Other thermal management techniques

In multiple component systems it may not be possible to remove the heat locally. In these situations some other, more efficient, method of heat transfer may be used to 'pipe' the heat away to a point where it can be removed more readily. These include copper cored boards, water cooled plates and immersion in fluoro-carbons. In all of these cases, the heat should be removed directly from the heat spreader on the top of the package to gain maximum benefit.

The above information is based on typical values obtained under ideal test conditions. Thermal performance will vary from one system to another as a function of the system design and method of assembly.

18 Electrical specifications

Inputs and outputs are TTL compatible.

18.1 Absolute maximum ratings

Symbol	Parameter	Min	Max	Units	Notes
VDD	DC supply voltage	0	7.0	V	1,2,3,4,5
V _I , V _O	Voltage on input and output pins	-0.5	VDD+0.5	V	1,3,4,5
I _I	Input current		±25	mA	6
t _{osc}	Output short circuit time (one pin)		1	s	4
T _s	Storage temperature	-65	150	°C	4

Table 18.1 Absolute maximum ratings

Notes

- All voltages are with respect to **GND**.
- Power is supplied to the device via the **VDD** and **GND** pins. Several of each are provided to minimize inductance within the package. All supply pins must be connected. The supply must be decoupled close to the chip by at least one 100 nF low inductance (e.g. ceramic) capacitor between **VDD** and **GND**. Four layer boards are recommended; if two layer boards are used, extra care should be taken in decoupling.
- Input voltages must not exceed specification with respect to **VDD** and **GND**, even during power-up and power-down ramping, otherwise *latchup* can occur. CMOS devices can be permanently damaged by excessive periods of latchup.
- This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operating sections of this specification is not implied. Stresses greater than those listed may cause permanent damage to the device. Exposure to absolute maximum rating conditions for extended periods may affect reliability.
- This device contains circuitry to protect the inputs against damage caused by high static voltages or electrical fields. However, it is advised that normal precautions be taken to avoid application of any voltage higher than the absolute maximum rated voltages to this high impedance circuit. Unused inputs should be tied to an appropriate logic level such as **VDD** or **GND**.
- The input current applies to any input or output pin and applies when the voltage on the pin is between **GND** and **VDD**.

18.2 Operating conditions

Symbol	Parameter	Min	Max	Units	Notes
VDD	DC supply voltage	4.75	5.25	V	1
V _I , V _O	Input or output voltage	0	VDD	V	1,2
CL	Load capacitance on any pin			pF	3

Table 18.2 Operating conditions

Notes

- All voltages are with respect to **GND**.
- Excursions beyond the supplies are permitted but not recommended.
- Maximum capacitance per address/ strobe/ data pin given in table 18.3, page 240.

18.3 Power rating

The internal power dissipation of the IMS T9000 depends on **VDD**, as shown in figure 18.1, and is substantially independent of temperature. It is dependent on operating frequency and program execution. The typical peak internal power dissipation (P_{INT}) for an IMS T9000 operating at 50 MHz is 3 W. Derating curves of maximum operating frequency against power dissipation will be included in the final datasheet.

The total power dissipation of the IMS T9000 is dependent on operating frequency, program execution, external memory configuration, and output pin loading.

The total peak power dissipation P_D of the chip is:

$$P_D = P_{INT} + P_{PMI}$$

The peak power dissipation of the PMI (P_{PMI}) can be determined for a given memory configuration from the following equation:

$$P_{PMI} = VDD^2 * ((n_{pA} * C_{pinA} * f_A) + (n_{pS} * C_{pinS} * f_S) + (n_{pD} * C_{pinD} * f_D))$$

where,

- n_p is the total number of active (address/ strobe/ data) pins
- C_{pin} is the actual capacitance per (address/ strobe/ data) pin
- f is the effective operating frequency per (address/ strobe/ data) pin

The maximum allowable capacitances that can be connected to each class of pins are:

Symbol	Parameter	Max	Units
C_{pinA}	Capacitance per address pin	250	pF
C_{pinS}	Capacitance per strobe pin	60	pF
C_{pinD}	Capacitance per data pin	60	pF
$n_{pA} * C_{pinA}$	Total address bus capacitance	2500	pF
$n_{pS} * C_{pinS}$	Total strobe pins capacitance	500	pF
$n_{pD} * C_{pinD}$	Total data bus capacitance	5000	pF

Table 18.3 Capacitance specifications

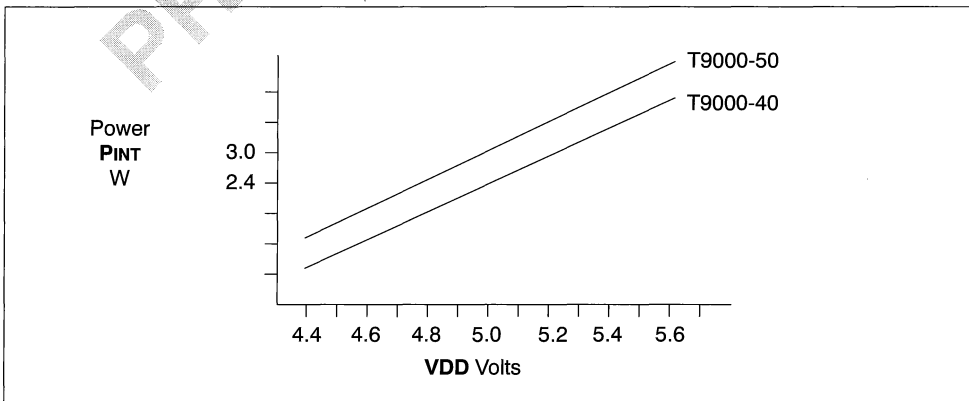


Figure 18.1 Internal power dissipation vs VDD

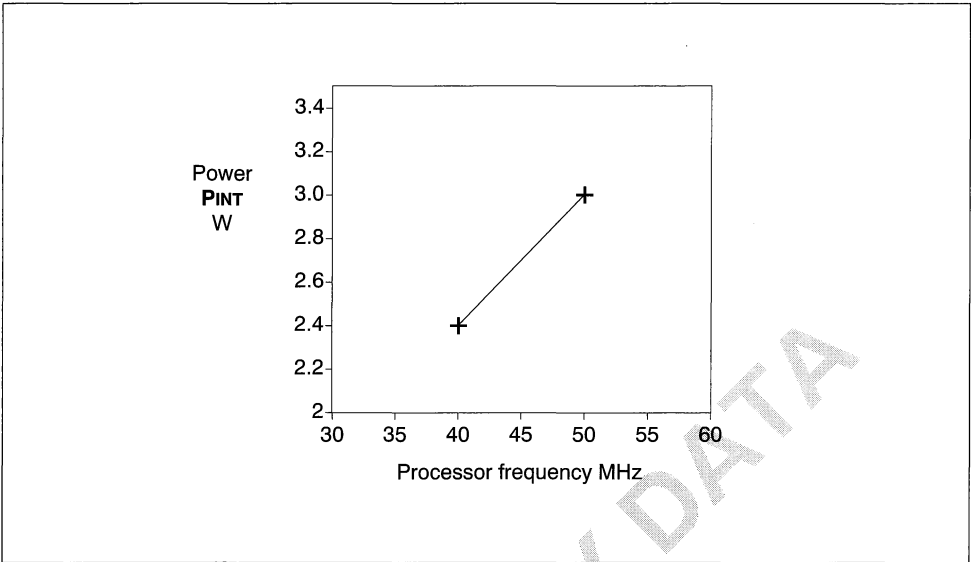


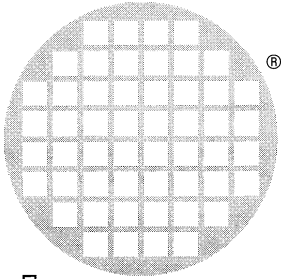
Figure 18.2 Typical peak internal power dissipation with processor speed

PRELIMINARY DATA



Part 3

Communications support devices



inmos[®]

IMS C100 system protocol converter

Preliminary Data

The information in this datasheet is subject to change

FEATURES

Communicates between T2xx/T4xx/T8xx and T9000 transputers

T2/T4/T8-series and T9-series data link protocol

T2/T4/T8-series and T9-series control protocol

Converts data and control protocols

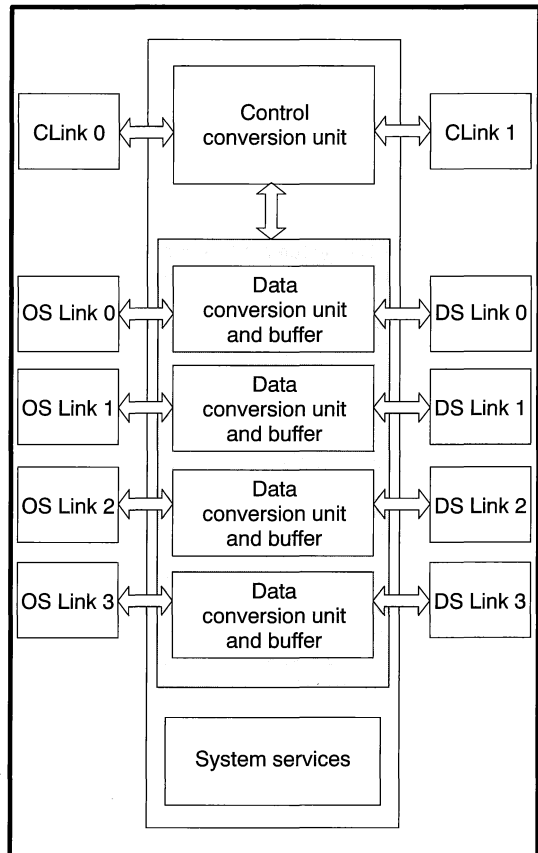
Four modes of operation:

Mode 0: Enables a single T9-series transputer to be used in a T2/T4/T8-series network

Mode 1: Enables a T2/T4/T8-series transputer system to use a T9-series subsystem

Mode 2: Enables a T9-series transputer system to use an existing T2/T4/T8-series subsystem without modifications to the T2/T4/T8 software

Mode 3: Enables a T9-series transputer system to use an existing T2/T4/T8-series subsystem, and enables a T2/T4/T8-series transputer to emulate a T9-series transputer



1.1 IMS C100 introduction

This document contains information on the IMS C100 system protocol converter.

The IMS C100 is part of a new product family based around the IMS T9000 transputer, referred to as the 'T9-series'. The current family of T2xx/T4xx/T8xx transputers are referred to as 'T2/T4/T8-series'.

T9-series transputers have different physical links and data protocols than T2/T4/T8-series transputers. The IMS C100 is a system protocol converter which converts between these protocols. It allows mixed systems, consisting of both T9-series and T2/T4/T8-series transputers, to be constructed. Enabling T2/T4/T8-series transputers to be added as low-cost peripherals to a T9-series network, or enabling T9-series transputers to be used to upgrade the performance of an existing transputer application.

T2/T4/T8-series transputer links consist of two wires, one in each direction, and use an asynchronous bit-serial (byte-stream) protocol. Each bit received is sampled five times and hence the links are referred to as oversampled links (OS-Links). Each link provides a pair of channels, one in each direction and can operate at up to 20 Mbits/sec, providing a bidirectional bandwidth of 2.4 Mbytes/sec.

T9-series transputer links consist of four wires, two in each direction, one carrying data and one carrying a strobe. The links are therefore referred to as data-strobe links (DS-Links). Each link can operate at up to 100 Mbits/sec, providing a bidirectional bandwidth of 20 Mbytes/sec. The DS-Link protocol supports *virtual channels* and *dynamic message routing*, and provides a high data bandwidth.

T2/T4/T8-series transputers are controlled by means of **Reset**, **Analyse** and **Error** pins on each device and are inspected and booted by means of a special protocol on their links. On T9-series transputers this is achieved by special links, called control links.

The IMS C100 provides an inter-networking solution for transputer systems, allowing systems to be constructed using the optimum mix of transputers, for processing power, communication bandwidth and system cost.

The IMS C100 converts both data and control protocols of T9-series transputer systems to those of T2/T4/T8-series, and vice versa. It is intended to be used in conjunction with software running on either T9-series or T2/T4/T8-series transputers and can operate in one of four modes.

This document describes the operation of the IMS C100 in detail, and summarizes the background information necessary to understand the full implications of each mode of operation.

The IMS Dx2xx toolsets refers to the C, OCCAM and FORTRAN toolsets written in C and supporting T2/T4/T8-series transputer networks. A set of C, C++ and OCCAM toolsets are available which incorporate T9-series transputer support. The tools provide support in the configuration and initialization of T9-series networks. The IMS T9000 configuration tools do not directly support the configuration of mixed T9-series and T2/T4/T8-series systems. Systems made up of T9-series networks and T2/T4/T8-series networks connected together via IMS C100s can be configured, with each network being configured and loaded separately using the appropriate toolset. Refer to *The T9000 Development Tools Preliminary Information* (document number 72 TRN 249 00) for further details.

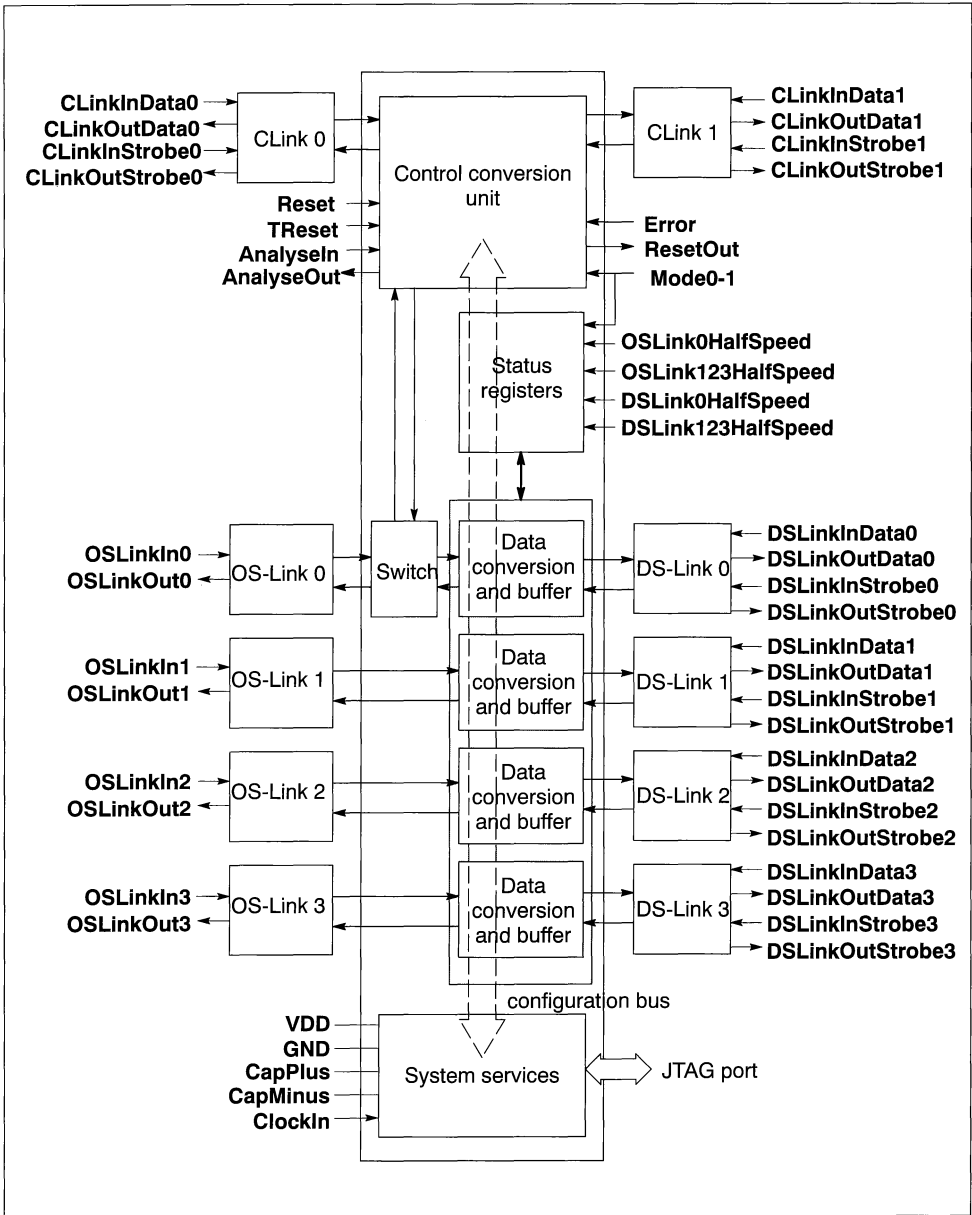


Figure 1.1 IMS C100 block diagram

1.2 IMS C100 modes of operation

This section describes the modes of operation of the IMS C100 and gives examples of its use in each mode. The figures given are included to illustrate how an IMS C100 may be used, they are not intended to provide accurate application designs. For a complete understanding of the implications of this section consult sections 1.3 to 1.6, which describe the link and control protocols of T2/T4/T8-series and T9-series components, and how the IMS C100 converts between these protocols.

The four modes of operation of the IMS C100 are listed below:

- Mode 0:** enables a T9-series transputer with ROM, from which the transputer boots, to emulate a T2/T4/T8-series transputer.
- Mode 1:** enables a T2/T4/T8-series system to use a T9-series subsystem.
- Mode 2:** enables a T9-series system to use an existing T2/T4/T8-series subsystem without any modification to the existing T2/T4/T8-series software.
- Mode 3:** enables a T9-series system to use an optimum T2/T4/T8-series subsystem and enables a T2/T4/T8-series transputer to emulate a T9-series transputer.

1.2.1 Mode pins

The IMS C100 has two mode pins (**Mode0-1**) which must be set at power-on. Table 1.1 details the mode pin settings. The mode of operation determines which type of conversion is to be performed between the data links, which system interface is regarded as master, and whether **OSLink0** has special initial behavior, see table 1.2. In modes 2 and 3 **OSLink0** is used to transmit the pre-boot protocol of the T2/T4/T8-series transputer until the transputer is booted (refer to section 1.6.2 for further information).

The OS-Link protocol synchronizes the communications of each byte of data, and hence the term *byte-stream* protocol has been adopted. DS-Links use a high level packet protocol and hence the term *packetized* protocol has been adopted. Each IMS T9000 transputer DS-Link may be set to operate in packetized mode or in byte-stream mode (see section 1.3.2). The IMS T9000 DS-Links operating in byte-stream mode, in conjunction with an IMS C100, convert the DS-Links to the byte-stream protocol.

Mode of operation	Mode pins	
	Mode1	Mode0
0	Low	Low
1	Low	High
2	High	Low
3	High	High

Table 1.1 **Mode0-1** pins

Note that the behavior of the IMS C100 is undefined if the mode pins are changed after reset.

Mode	Master	Subsystem	DS-Link mode	Control system master	OSLink0 (pre-boot)
0	T2/T4/T8-series	Single T9 + ROM	Byte-stream	Reset, Analyse, Error	Not special
1	T2/T4/T8-series	T9 network	Packetized	Control link 0	Not special
2	T9-series	T2/T4/T8 network	Byte-stream	Control link 0	Special
3	T9-series	T2/T4/T8 network	Packetized	Control link 0	Special

Table 1.2 Mode settings

1.2.2 Mode 0: Enables a single T9-series transputer to be used in a T2/T4/T8-series network

The purpose of this mode is to allow a single IMS T9000 transputer to operate as a fast T2/T4/T8-series transputer.

To convert a T9-series transputer to a T2/T4/T8-series interface, connect the four data DS-Links and the control link **CLink0** to the data DS-Links and **CLink0** of the IMS C100, as shown in figure 1.2, and set the IMS C100 to mode 0. The combination of the IMS C100 and the IMS T9000 transputer is controlled like a T2/T4/T8-series transputer by Reset, Analyse and Error signals.

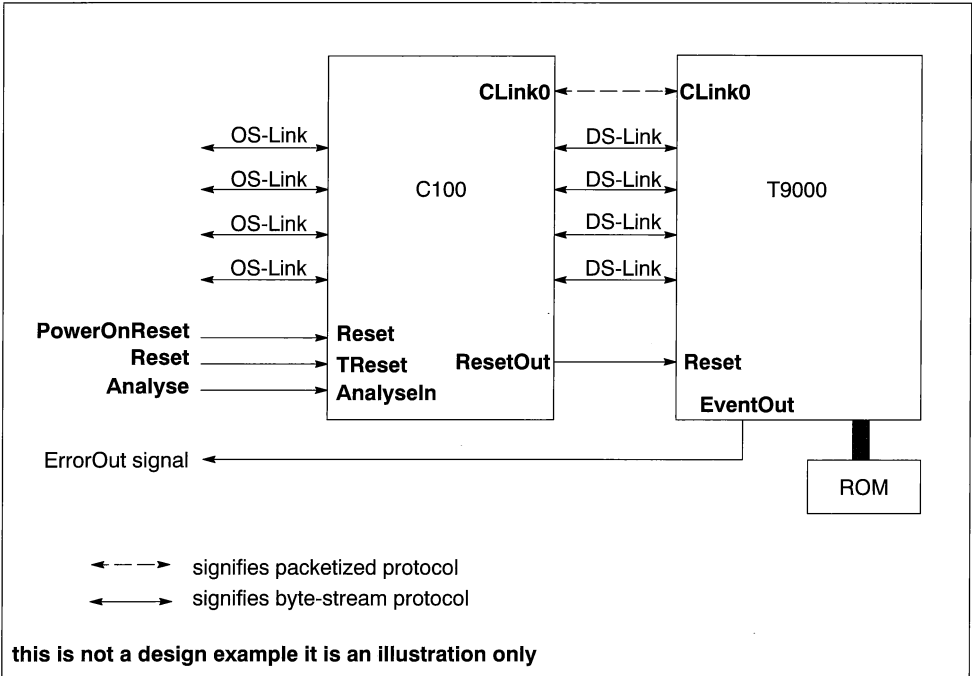


Figure 1.2 Mode 0 – converting an IMS T9000 transputer for use in a T2/T4/T8-series network

The **StartFromROM** pin on the IMS T9000 transputer must be set high so that the IMS T9000 transputer boots from ROM. Following power-on-reset, the IMS T9000 transputer runs with a workspace pointer (**Wptr**) and an instruction pointer (**lptr**) from two fixed locations near the top of memory (i.e. in the ROM). The ROM software configures the IMS T9000, and sets the IMS T9000 data links into byte-stream mode and starts them.

The IMS T9000 data links interact with the IMS C100 DS-Links operating in byte-stream conversion mode to generate the T2/T4/T8-series transputer protocol on the OS-Links of the IMS C100. The ROM software, as supplied to customers, also emulates the pre-boot protocol of T2/T4/T8-series transputers, and emulates the Error pin behavior of a T2/T4/T8-series transputer by performing an output on an event output channel whenever a T2/T4/T8-series transputer would assert its Error pin. The IMS C100 sends an initial message to program the label and return header of the IMS T9000 transputer's control link.

Analyse is used as a debugging aid on T2/T4/T8-series transputers. Reset can behave in two ways depending on the status of the **Analyse** signal, as described below.

The **TReset** pin indicates transputer reset of the connected T2/T4/T8-series transputer. Asserting the **TReset** pin of the IMS C100 with the **Analyseln** pin low, resets the IMS C100. The signal is reproduced

on **ResetOut** which causes the IMS T9000 to reset. When **TReset** is taken low this restarts the ROM code which repeats the above pre-boot sequence.

If the **AnalyseIn** pin of the IMS C100 is asserted, the IMS C100 sends a *Stop* message from control link **CLink0**. The IMS T9000 returns a *StopHandshake*. When the **TReset** pin of the IMS C100 is asserted, the IMS C100 sends a *Reset* message. When both **TReset** and **AnalyseIn** are deasserted, the IMS C100 sends a *Reboot* message. This restarts the ROM code. If this code executes a *testpranal* instruction it can take special action to assist the debugger before it repeats the above pre-boot sequence.

The **TReset** and **AnalyseIn** signals are used in this mode only and are ignored in modes 1, 2 and 3.

The **Reset** pin is provided in this case for systems which separate power-on-reset from transputer reset. When the **Reset** pin is asserted it always causes a reset of both the IMS C100 and the attached IMS T9000 (by being reproduced on **ResetOut**).

Note that, in this mode, the link speeds of the IMS C100 are set by the link speed pins, and cannot be changed by software as the system does not include a control link network.

1.2.3 Mode 1: Enables a T2/T4/T8-series system to use a T9-series subsystem

The purpose of this mode is to allow a T9-series subsystem to be connected to, and controlled from, a T2/T4/T8-series network. Communication is in the packetized protocol, and software must be run on the T2/T4/T8-series system to interface the packetized protocol, and to control the T9-series subsystem.

To enable a T2/T4/T8-series system to use a T9-series subsystem, set the IMS C100 to mode 1, and connect one or more OS-Links from the T2/T4/T8-series system to the OS data links of the IMS C100. Since T9-series systems are controlled entirely via links this enables T9-series subsystems to be configured, booted, reset and analyzed from a T2/T4/T8-series system. An example network is shown in figure 1.3. The RAE signals to the T2/T4/T8-series network are shown by the dotted line. The IMS C004 programmable link switch has 32 links, of which only six are shown in this example.

Note that, by 'looping back' through the control links of the IMS C100, the T2/T4/T8-series system obtains full control of the device. Note, however, that the IMS C100 must be labelled, and CLink1 started, before any of the devices in the T9-series subsystem can be accessed/controlled.

If an error occurs on the connection between the DS data link and control link CLink0, both links will reset. The DS data link will be restarted, and CLink0 will respond by starting automatically, thereby restoring control of the device.

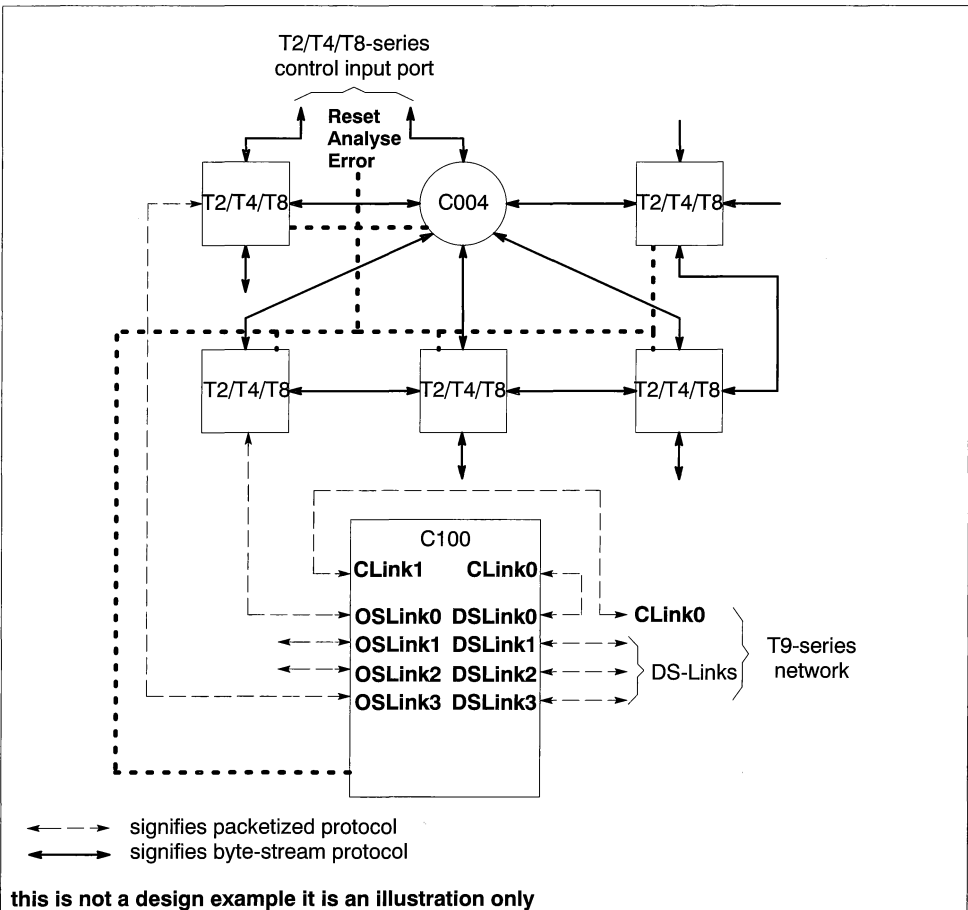


Figure 1.3 Mode 1 – T2/T4/T8-series system using T9-series subsystems

1.2.4 Mode 2: Enables a T9-series system to use an existing T2/T4/T8-series subsystem

The purpose of this mode is to allow T9-series systems to use an existing T2/T4/T8-series subsystem, without having to change either the hardware or the software of the T2/T4/T8-series subsystem. For example, a SCSI TRAM purchased as a functional subsystem from a third party supplier (including both hardware and the associated software drivers) can be used unmodified as a subsystem to a T9-series system. Thus this mode protects users existing investment in transputer-based equipment.

Figure 1.4 shows how a T9-series system can use an existing T2/T4/T8-series subsystem. Each IMS C104 packet routing switch has 32 data links, of which only seven are shown in this example. Note that the data DS-Links of the IMS C100 must be connected directly to IMS T9000 data links set into byte-stream mode, and **cannot** be connected to an IMS C104 packet routing switch.

The T2/T4/T8-series subsystem is controlled via **CLink0** of the IMS C100. After power-on, commands sent along **CLink0** are converted to the appropriate T2/T4/T8-series byte sequences which are sent along **OSLink0** of the IMS C100. This allows the memory of transputers in the T2/T4/T8-series subsystem to be peeked and poked, and for it to be booted.

Assertion of the **AnalyseOut** and **ResetOut** pins, which is achieved by sending appropriate messages on **CLink0**, results in the **Reset** and **Analyse** pins of the connected T2/T4/T8-series transputer being asserted, enabling it to be stopped and analyzed.

By setting the IMS C100 to mode 2, and connecting a link from a T9-series system to control link **CLink0** of the IMS C100, a T2/T4/T8-series subsystem can be reset, analyzed and monitored for errors. If one or more links, of an IMS T9000 set into byte-stream mode, are connected to the DS data links of the IMS C100, the subsystem can also be configured and booted.

Alternatively, provided the initial bootstrap of the T2/T4/T8-series device (which comes either from a ROM or via the booting mechanism described above) is capable of inputting further code in packetized form, setting the IMS C100 to mode 3 avoids the necessity of dedicating any specific IMS T9000 links to the control of the subsystem, see section 1.2.5. Note that in this case a *CPoke* is required to set the booted flag so that link 0 ceases to be connected to the control unit.

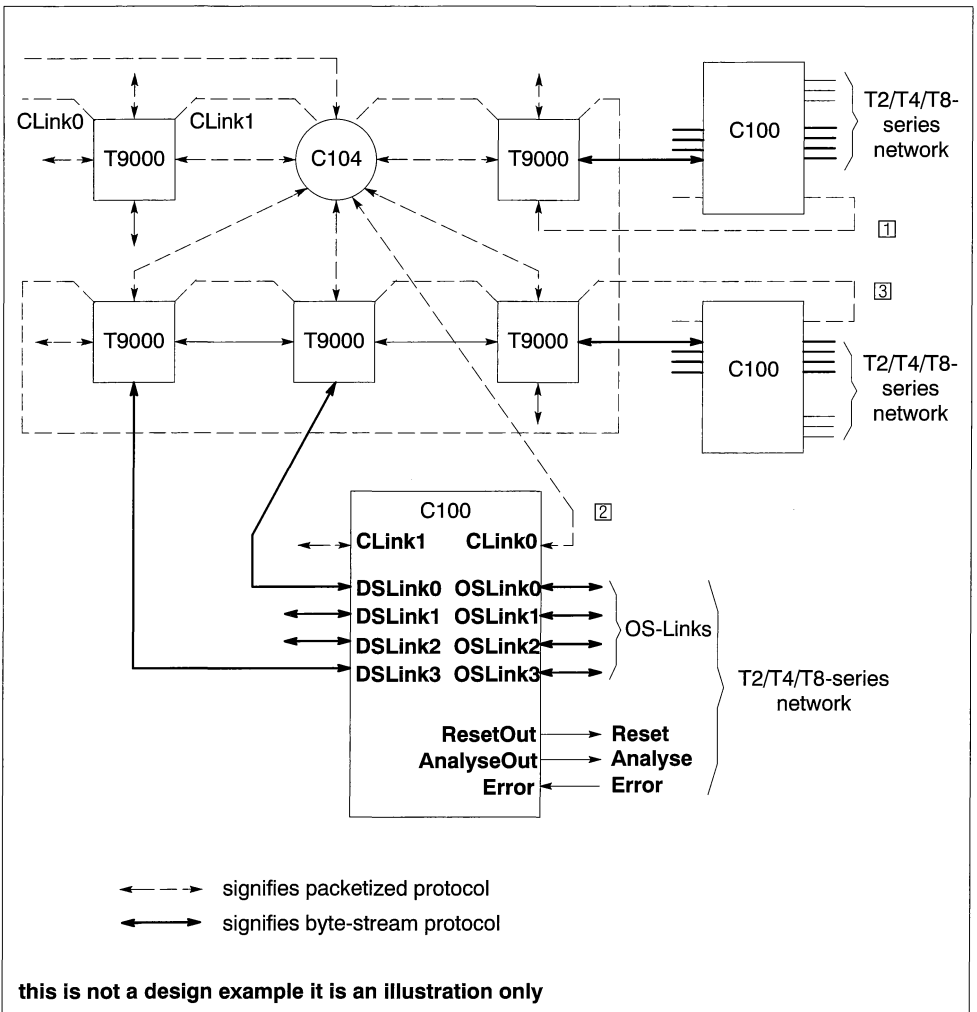


Figure 1.4 Mode 2 – T9-series system using existing T2/T4/T8-series subsystems

Figure 1.4 illustrates several possibilities in a system, which in practice may not be realistic. Three connection types are shown in this diagram.

- ① CLink0 of the IMS C100 connected to a data link of the IMS T9000. The IMS T9000 has full control of the IMS C100.
- ② CLink0 of the IMS C100 connected to the IMS C104. The IMS T9000 has full control of the IMS C100, but data link to control link is not dedicated and is routed via an IMS C104.
- ③ CLink0 of the IMS C100 connected to the control link of the IMS T9000. The IMS C100 is controlled from the controlling processor.

1.2.5 Mode 3: Enables a T9-series system to use a T2/T4/T8-series subsystem

The purpose of this mode is to allow T9-series systems to be built which use T2/T4/T8-series subsystems, enabling systems to use the optimum mix of transputers with regard to cost and performance.

Communication is in the packetized protocol. Thus the data DS-Links of the IMS C100 can be connected directly to an IMS C104 packet routing switch, as in figure 1.5.

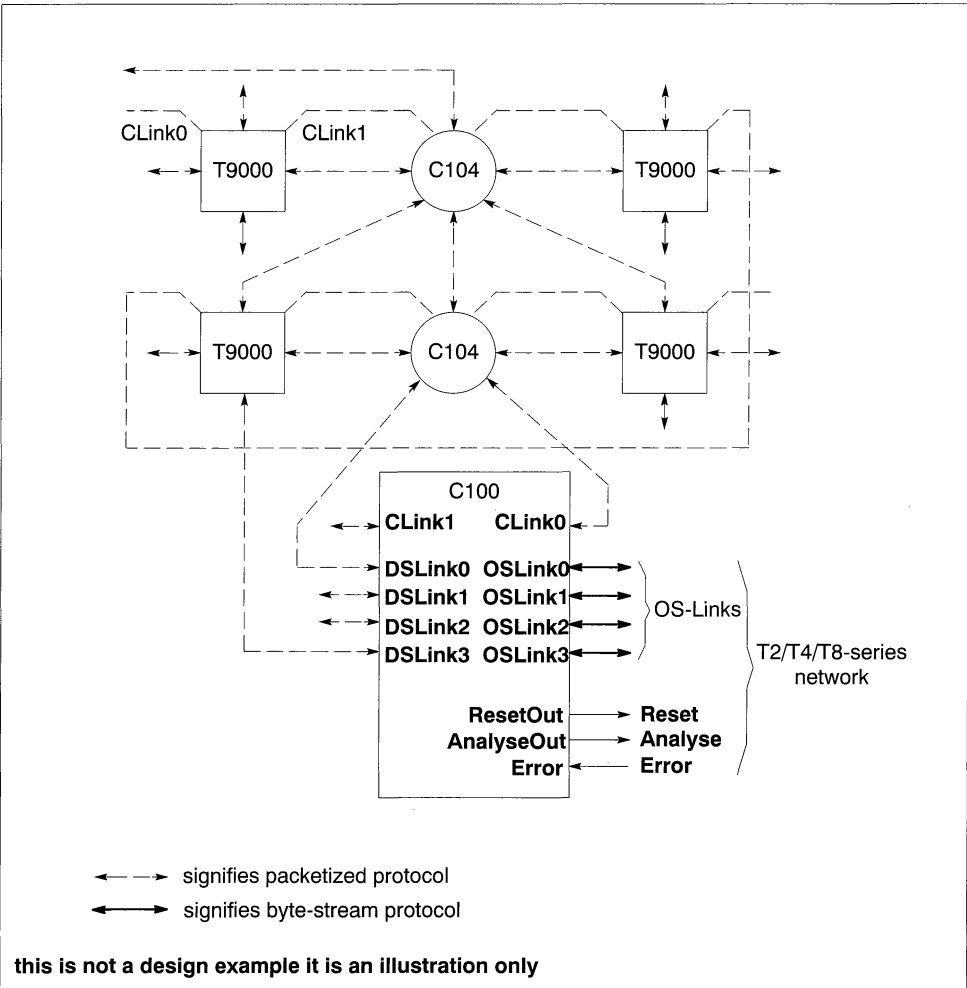


Figure 1.5 Mode 3 – T9-series system using optimum T2/T4/T8-series subsystems

The T2/T4/T8-series subsystem is controlled via **CLink0** of the IMS C100. Messages received on **CLink0** of the IMS C100 cause individual links to be reset, and the **ResetOut** and **AnalyseOut** pins to be toggled. Assertion of the **AnalyseOut** and **ResetOut** pins results in the **Reset** and **Analyse** pins of the connected T2/T4/T8-series transputer being asserted, enabling it to be stopped and analyzed. An error from within the IMS C100 and a signal on the **Error** pin both cause an **Error** message to be sent from **CLink0**.

The IMS C100 operating in mode 3 can be used to enable a T2/T4/T8-series transputer to emulate a T9-series transputer, as shown in figure 1.6. This is achieved by connecting the **ResetOut**, **AnalyseOut**,

and **Error** pins of the IMS C100 to the **Reset**, **Analyse**, and **Error** pins of the T2/T4/T8-series transputer and setting the IMS C100 into mode 3. This combination of the IMS C100 and the T2/T4/T8-series transputer has a control link 0 (**CLink0**), and a control link 1 (**CLink1**) for daisy-chaining. Figure 1.6 shows a T2/T4/T8-series transputer being converted to an IMS T9000 interface, with the T2/T4/T8-series transputer being booted from a link. In this mode, software must be run on the T2/T4/T8-series transputer to convert the OS-Links to the packetized protocol.

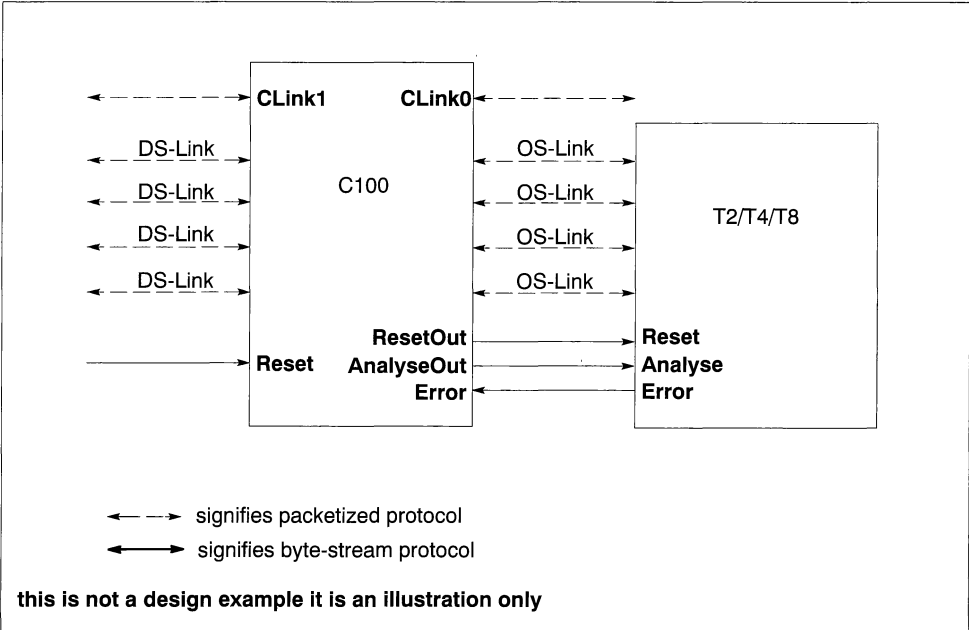


Figure 1.6 Mode 3 – converting a T2/T4/T8-series transputer for use in a T9-series network

Note that to emulate a T9-series device operating in **StartFromROM** and **standalone** mode, the T2/T4/T8-series device can have control over the DS data links by looping back one of them through **CLink0**, and setting the IMS C100 to mode 1, see figure 1.3.

1.3 Link protocols

This section describes the different link protocols used on T2/T4/T8-series and T9-series components. The two types of link protocol conversion are described in the following section.

1.3.1 T2/T4/T8-series oversampled links

T2/T4/T8-series transputer links consist of two wires, one in each direction, and use an asynchronous bit-serial protocol. Link inputs are sampled five times in each bit period, and hence the links are referred to as oversampled (OS) links.

Messages are transmitted as a sequence of single byte communications, each of which must be acknowledged. The acknowledge packets are used both to signal reception of the data bytes and to maintain flow control.

A link provides a pair of channels, one input and one output channel. Every byte of data sent on an output channel is acknowledged on the input channel of the same link, thus each signal line carries both data and control information.

Each data byte is transmitted as a high start bit followed by another high bit followed by eight data bits followed by a low stop bit, as shown in figure 1.7. The least significant bit of data is transmitted first. After transmitting a data byte the sender waits for the acknowledge, which consists of a high start bit followed by a zero bit. The acknowledge signifies that the receiving link is able to receive another byte.

The receiving transputer can send an acknowledge as soon as the data has been identified (provided there is sufficient buffer space for another data byte, and that an inputting process is ready to receive the data byte) so that communications can be continuous.

The link protocol synchronizes the communications of each byte of data, and hence the term *byte-stream* protocol has been adopted. As the protocol supports the transmission of an arbitrary sequence of bytes transputers of different word lengths can be connected together.

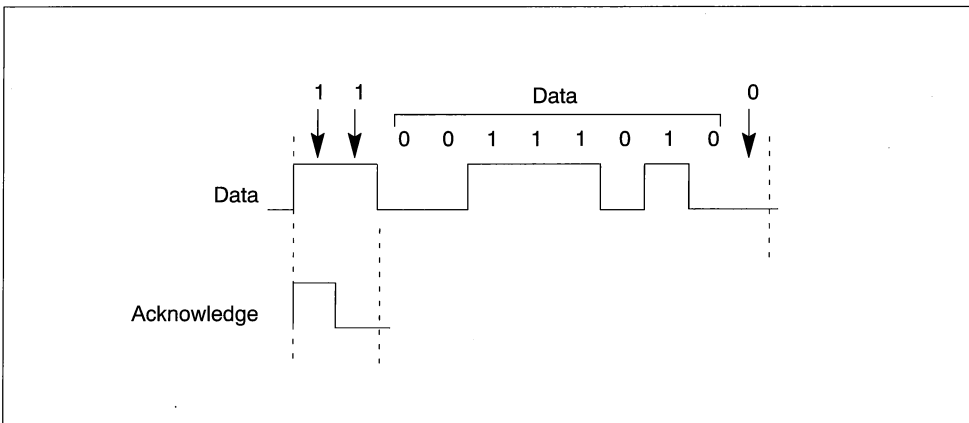


Figure 1.7 OS-Link data and acknowledge formats

The T2/T4/T8-series transputer family includes link adaptor devices, the IMS C011 and IMS C012, which enable OS-Links to interface with non-transputer devices.

1.3.2 T9-series data/strobe links

T9-series transputer links consist of four wires, two in each direction, one for data and one to carry a strobe signal. These links are therefore referred to as data/strobe (DS) links.

Communication between processes on one IMS T9000 transputer takes place over software channels. Communication between processes on different processors takes place over virtual channels. Virtual channels are multiplexed onto each physical link by a communications processor within the IMS T9000. The data links support a physical link protocol to support virtual channels and dynamic message routing, and to provide a high data bandwidth.

Each message is split into a sequence of packets, each of which has the structure shown in figure 1.8.

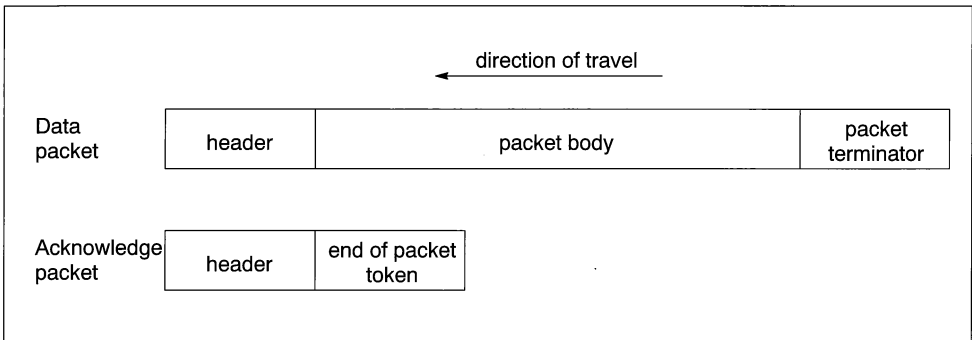


Figure 1.8 Structure of a packet on DS-Links

Packets from different messages may be interleaved over each physical link. Interleaving packets from different messages allows any number of processes to communicate simultaneously via each physical link. Communication channels can be established between any two processes regardless of where they are physically located, or whether the channels are routed through a network. Thus, programs can be independent of network topology.

In order that packets which are parts of different messages can be distinguished by the transputer which receives them, each packet contains a one or two byte header which identifies a virtual input channel of the receiving transputer. The packet header is also used to route the packet through a network. Bytes following the header are treated as the data section of the packet until a packet termination token is received. A packet termination token is either an end of packet (EOP) token or an end of message (EOM) token. The maximum length of data in each packet which the IMS T9000 can transmit is 32 bytes. All but the last packet of a message contains the maximum amount of data; the last contains the maximum amount of data or less.

The communications processor within the IMS T9000 enforces a high-level protocol on each virtual channel. To maintain synchronized communication, and to ensure that no data is lost, each packet of data sent along a virtual channel must be acknowledged before the next is sent. The last packet must be acknowledged before the outputting process is rescheduled. Data packets on a virtual channel are acknowledged by the communications processor by sending acknowledge packets on another virtual channel back to the processor which sent them. Acknowledge packets are packets containing no data and which are always terminated by an EOP token. The acknowledge packets perform packet-level flow-control and process synchronization.

Virtual channels always occur in pairs between pairs of communicating processors, with one virtual channel in each direction. If a message is being communicated in one direction the virtual channel in the opposite direction is used to return acknowledge packets to the sender. The associated pair of virtual channels is referred to as a *virtual link*. A virtual link can transfer messages in both directions at the same time with data packets and acknowledge packets being interleaved on both of the virtual channels. Because virtual channels are always paired in this way it is not necessary to include source information in the packets. Thus packet headers need only represent their destinations.

Each DS pair carries tokens and an encoded clock. The tokens can be data bytes or control tokens. Figure 1.9 shows the format of data and control tokens on the data and strobe wires. Data tokens are 10 bits long and contain a parity bit, a flag which is set to 0 to indicate a data token, and 8 bits of data. Control tokens are 4 bits long and contain a parity bit, a flag which is set to 1 to indicate a control token, and 2 bits to indicate the type of control token.

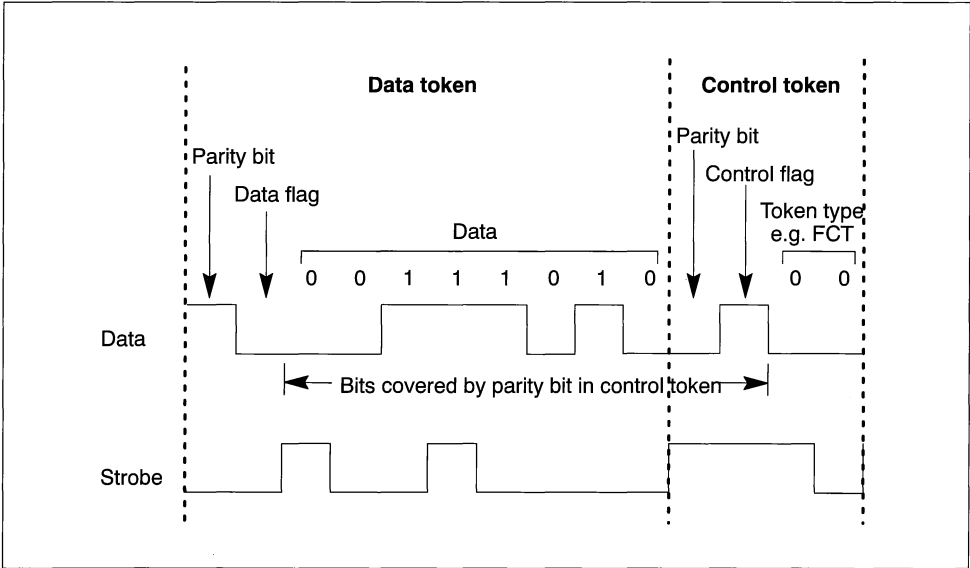


Figure 1.9 DS-Link data and strobe formats

The DS-Link protocol ensures that only one of the two wires of the data strobe pair has an edge in each bit time. The levels on the data wire give the data bits transmitted. The strobe signal changes whenever the data signal does not. These two signals encode a clock together with the data bits, permitting asynchronous detection of the data at the receiving end.

The data and control tokens are of different lengths, for this reason the parity bit in any token covers the parity of the data or control bits in the previous token, and the data/control flag in the same token, as shown in figure 1.9. This allows single bit errors in the token type flag to be detected. **Odd** parity checking is used. Thus the parity bit is set/unset to ensure that the bits covered, inclusive of the parity bit (see figure 1.9) always contain an odd number of 1's. To ensure the immediate detection of errors null tokens are sent in the absence of other tokens. The coding of the tokens is shown in table 1.3.

Token type	Abbreviation	Coding
Data token	–	P0DDDDDDDD
Flow control token	FCT	P100
End of packet	EOP	P101
End of message	EOM	P110
Escape token	ESC	P111
Null token	NUL	ESC P100

P = parity bit
D = data bit

Table 1.3 Token codings

The DS-link protocol separates the functions of flow control and process synchronization. Token-level flow control is performed in each link module, and the additional flow control tokens used are not visible to the higher-level packet protocol. The token-level flow control mechanism prevents a sender from overrunning the input buffer of a receiving link.

Each receiving link input contains a buffer for at least 8 tokens (more buffering than this is in fact provided). Whenever the link input has sufficient buffering available to consume a further 8 tokens (consisting of data and EOP or EOM tokens) a FCT is transmitted on the associated link output, and this FCT gives the sender permission to transmit a further 8 tokens. Once the sender has transmitted a further 8 tokens it waits until it receives another FCT before transmitting any more tokens. The provision of more than 8 tokens of buffering on each link input ensures that in practice the next FCT is received before the previous block of 8 tokens has been fully transmitted, so that the token-level flow control does not restrict the maximum bandwidth of the link.

DS-Links use a high level packet protocol and hence the term *packetized* protocol has been adopted.

Byte-stream mode

Each IMS T9000 data DS-Link (**Link0-3**) can be set to operate either in virtual channel mode or in byte-stream mode. Byte-stream mode is provided to allow IMS T9000 DS-Links to communicate, via an IMS C100, with OS-Links of a T2/T4/T8-series subsystem carrying the byte-stream protocol.

Byte-stream mode can be set independently for each IMS T9000 data link by setting the **ByteMode** bit in the associated VCP link mode configuration register (**VCPLink0-3Mode**). Refer to the *T9000 Transputer Datasheet – Communications chapter 11* for further information on setting an IMS T9000 data link to byte-stream mode. Setting the IMS T9000 links independently of each other, enables each IMS T9000 transputer to be connected to several different networks.

A DS data link operating in byte-stream mode is able to send and receive data. Data is transferred along the link in the form of packets each with a single byte header. Each packet is terminated with either an EOP or EOM token. The packets which can be sent and received on a DS-Link operating in byte-stream mode are given in section 1.4.1. Software on the IMS T9000 sends and receives messages normally via a pair of byte-stream channels.

For IMS T9000 links operating in byte-stream mode the packet headers are fixed and the byte-stream channels cannot be deactivated, stopped or put into resource mode. For compatibility with the T8xx transputer, the **Wdescriptors** of processes actually communicating on byte-stream channels are stored in the 8 words at the bottom of memory.

An IMS T9000 link set to byte-stream mode must be connected to an IMS C100 DS data link as it will not operate correctly if directly connected to another IMS T9000 link in byte-stream mode.

1.4 Link protocol conversion

This section describes the conversion between the two link protocols that the IMS C100 supports.

The IMS C100 is able to convert between the OS and DS-Link protocols in one of two ways depending on whether the higher message/packet level of the DS-Link protocol is added to the OS-Link or removed from the DS-Link.

Byte-stream conversion

The DS-Link of the connected T9-series transputer must be set to byte-stream mode. A pair of channels is then supported from the T9-series transputer, through the IMS C100, to a T2/T4/T8-series transputer. Software on the T2/T4/T8-series transputer sees the channels as being identical to that through a normal OS-Link. No modification to the T2/T4/T8-series transputer software is needed.

Packetized conversion

A process must be run on the connected T2/T4/T8-series transputer to impose a software packet protocol onto the OS-Link. This is converted by the IMS C100 to the hardware supported packet protocol on the DS-Link.

The IMS C100 data DS and OS-Links are paired, and all pairs perform one or other type of conversion, depending on the mode. In modes 1 and 3, all four link pairs convert the packetized protocol; in modes 0 and 2, all four convert the byte-stream protocol. The two types of conversion are described below for one pair of links.

1.4.1 Byte-stream conversion – modes 0 and 2

The OS-Link of the IMS C100 is identical to an OS-Link on a T2/T4/T8-series component. No modification to software running on a connected T2/T4/T8-series transputer is needed.

The DS-Link of the connected T9-series transputer must be set in byte-stream mode and connected to the DS-Link of the IMS C100, see figure 1.10. The IMS C100 cannot be directly connected to an IMS C104 when this type of conversion is being used. Software on the T9-series transputer will send and receive messages normally, via a pair of channels.

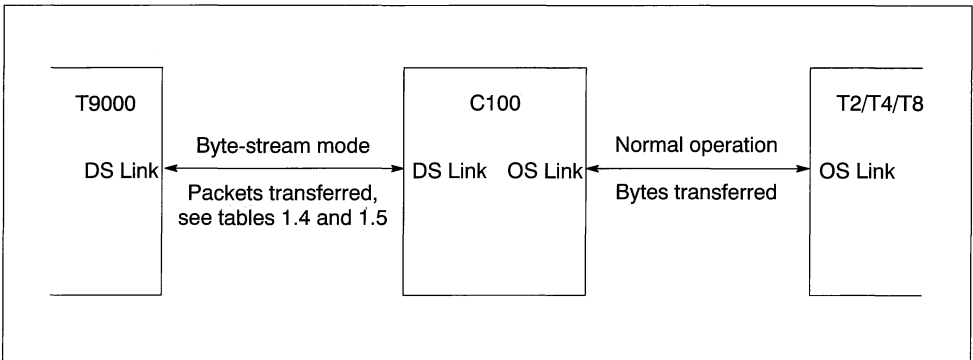


Figure 1.10 Byte-stream conversion

A special protocol is used between the IMS C100 and the T9-series transputer. This protocol is invisible to the user, and is described here for completeness. Data is transferred along the DS-Link in the form of packets each with a single byte header. Note that the DS-Link on the IMS T9000 must be set to expect single byte headers. Each packet is terminated with either an EOP or EOM token. The packets sent to/from a DS-Link in byte-stream mode are given in tables 1.4 and 1.5.

The IMS C100 interprets packets from the T9-series transputer as indicated in table 1.4. Note that the DS-Links of an IMS T9000 transputer which have been set into byte-stream mode generate this protocol automatically. The packet protocol ensures that communication between an attached IMS T9000 and the IMS C100 is never blocked, regardless of what data is or is not supplied and accepted along the OS-Link.

When a message is sent from an IMS T9000 to the IMS C100 on a DS-Link, the EOM token signals when the message is complete. However, when a message is sent from a T2/T4/T8 to the IMS C100 on an OS-Link there is no indication of when the complete message has been received. Therefore the T9000 must tell the C100 the length of the message it expects to receive from the T2/T4/T8. The IMS T9000 does this by sending an 'input count' packet which contains a count of the data bytes to be transferred from the OS-Link to the DS-Link by the IMS C100.

Header	Data	Terminator	Interpretation	Notes
0	32 bytes	EOP	Part of message	
0	1 to 32 bytes	EOM	End of message	
0	none	EOP	Acknowledgement	
1	1 to 4 bytes	EOM	Input count	1

Notes

- 1 The 'input count' packet contains the count of the data bytes the IMS T9000 expects to receive from the T2/T4/T8 transputer, via the IMS C100.

Table 1.4 Packets from IMS T9000 to IMS C100 – modes 0 and 2

The IMS C100 sends packets along the DS-Link, as shown in table 1.5. Note that DS-Links of an IMS T9000 transputer which have been set into byte-stream mode accept this protocol automatically.

When an unsolicited byte arrives from the T2/T4/T8 the IMS C100 informs the IMS T9000 by sending a zero length message.

Header	Data	Terminator	Interpretation	Notes
0	32 bytes	EOP	Part of message	
0	1 to 32 bytes	EOM	End of message	
0	none	EOP	Acknowledgement	1
0	none	EOM	Unsolicited byte	2

Notes

- 1 The acknowledgement packet is sent when the IMS C100 is ready to receive more data.
- 2 If a byte is received from the OS-Link whilst the output count is zero, the count is effectively reduced to -1 and an unsolicited packet is sent.

Table 1.5 Packets from IMS C100 to IMS T9000 – modes 0 and 2

Bytes are transferred normally on the OS-Link between the IMS C100 and the T2/T4/T8.

The IMS C100 can respond to both data bytes and acknowledges on the OS-Links immediately by buffering data from the IMS T9000 and holding a count of the input length, thus maintaining full bandwidth.

Messages from the T9000 to the T2/T4/T8

The IMS C100 is always ready to input data from the IMS T9000 on the DS-Link. Data following the header is placed in a FIFO. The contents of the FIFO are output to the T2/T4/T8 on the OS-Link immediately, subject to the receipt of acknowledges. If the packet is terminated with an EOP, the IMS C100 sends an

acknowledge packet on the DS-Link as soon as it has enough buffer capacity for another whole packet. If the packet is terminated with an EOM, the IMS C100 sends an acknowledge packet on the DS-Link as soon as the last byte of the packet has been sent and acknowledged on the OS-Link. Since an acknowledge packet is only sent when there is enough room in the buffer for a full-size data packet, a packet input can never be blocked, regardless of whether the OS-Link is able to transmit data or not.

Messages from the T2/T4/T8 to the T9000

A packet received from the DS-Link with a header of 1 is taken as the count of bytes to be transferred from the OS-Link to the DS-Link by the IMS C100. Thus the IMS C100 knows when a message arriving on the OS-Link from the T2/T4/T8 is complete.

If this count does not exceed the maximum amount of data permitted in a packet (32 bytes), the IMS C100 inputs that number of bytes from the OS-Link, and when there are no outstanding acknowledges, sends them on the DS-Link as a packet with header 0, terminated with an EOM. The count is then reduced to zero.

If the count exceeds the maximum amount of data permitted in a packet, the IMS C100 inputs the maximum allowed number of bytes, decrements the count by that amount, and when there are no outstanding acknowledges, sends the bytes on the DS-Link as a packet with header 0, terminated with an EOP. This process is repeated until the count is reduced to zero.

Each time such a packet is sent an acknowledge is outstanding until an acknowledge packet is received on the DS-Link from the IMS T9000.

If a byte is received from the OS-Link whilst the input count is zero, an unsolicited packet is sent. This is to enable the alternative mechanism of the transputer to operate correctly on byte-stream mode channels.

1.4.2 Packetized conversion – modes 1 and 3

This conversion type allows software on a connected T2/T4/T8-series transputer to use virtual channels to communicate with processes in the connected T9-series system. The IMS C100 can be directly connected to an IMS C104 when this type of conversion is being used.

With packetized conversion the DS-Links of the IMS C100 are operationally identical to the DS-Links of T9-series transputers.

Software must be run on the connected T2/T4/T8-series transputer to:

- 1 Packetize the messages output from the T2/T4/T8-series transputer, according to the protocol described below.
- 2 Interpret the packetized messages arriving at the T2/T4/T8-series transputer.

The IMS C100 converts packets between the software supported protocol on the OS-Link and the hardware supported packetized protocol on the DS-Link.

The length of packets in T9-series DS-Links is indicated by terminator codes EOP or EOM. In order for the IMS C100 to determine the length of the packet on OS-Links, the length must be given explicitly as an unsigned 7 bit value. This is contained in the 'count byte' at the start of the packet.

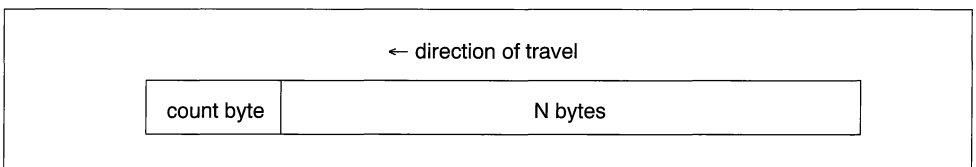


Figure 1.11 Structure of a software supported packet on OS-Links

The first byte of a (software supported) packet on an OS-Link is defined to be a count byte as shown in figure 1.12.

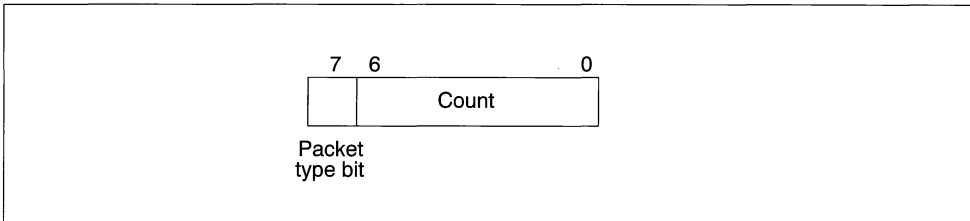


Figure 1.12 Structure of a count byte in an OS-Link packet.

If the packet type bit in the count byte is 0 then the packet is equivalent to a DS-Link packet terminated by an EOP token. If the packet type bit is 1 then the packet is equivalent to a DS-Link packet terminated by an EOM token.

The packet level of the DS-Link protocol is represented in the OS-Link protocol by transformations, as shown in tables 1.6 and 1.7. The transformations are given for different representations of a packet containing N bytes, including the header, see figure 1.13. Note that these transformations are independent of details of the structure of the packet, such as the header length.

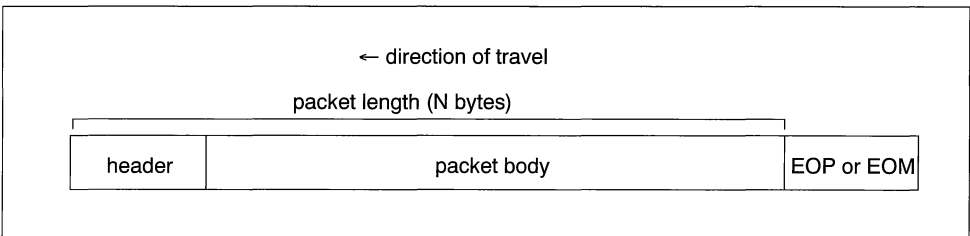


Figure 1.13 Structure of a packet on DS-Links

Messages from the T2/T4/T8 to the T9000

Received on OSLink of C100 ← direction of travel	N	Count byte		Transmitted on DSLink of C100 ← direction of travel
		bit 7	bits 0-6	
[Count byte] [N bytes]	1 to 127	0	N	[N bytes] EOP
[Count byte] [N bytes]	1 to 127	1	N	[N bytes] EOM

Table 1.6 Packets from T2/T4/T8 via the IMS C100 to T9000 – modes 1 and 3

For each packet, the IMS C100 first inputs a byte along the OS-Link. The IMS C100 masks the top bit of the byte to give the count of the number of bytes to follow. A count of zero is an error, which is signalled on control link 0; see table 1.8, page 278. Otherwise, the IMS C100 inputs that number of bytes into a 37 byte FIFO. If the top bit of the count byte was zero, it places an EOP token in the FIFO; if it was one, an EOM is inserted instead. The cycle then repeats.

As soon as the last byte is received, or the FIFO becomes full, the IMS C100 starts transmitting the contents of the FIFO from the DS-Link. It stops transmitting when it sends an EOP or EOM and the next packet has not been completely received. Note that, since the process synchronization is performed at a higher level, every byte received from the OS-Link can be acknowledged immediately provided there is room in the FIFO for another byte. In this way the maximum data rate can be attained on the OS-Link.

Messages from the T9000 to the T2/T4/T8

To improve the efficiency of software running on the T2/T4/T8-series transputer, packets of less than 7 bytes received on the DS-Link have extra bytes added. The extra bytes are all zero.

Received on DSLink of C100 ← direction of travel	N	Transmitted on OSLink of C100 ← direction of travel	Count byte		Note
			bit7	bits0-6	
[N bytes] EOP	< 7	[Count byte] [N bytes] [(7-N) zero bytes]	0	N	1, 2
[N bytes] EOP	≥ 7	[Count byte] [N bytes]	0	N	
[N bytes] EOM	< 7	[Count byte] [N bytes] [(7-N) zero bytes]	1	N	2
[N bytes] EOM	≥ 7	[Count byte] [N bytes]	1	N	

Table 1.7 Packets from T9000 via the IMS C100 to T2/T4/T8 – modes 1 and 3

Notes

- 1 A packet received on the DS-Link containing less than 32 bytes of data and ending in an EOP is an acknowledge packet.
- 2 Packets received on the DS-Link, from the connected T9-series component, which are less than 7 bytes have zero bytes added. This improves the efficiency of software running on T2/T4/T8-series transputers.

Data received on the DS-Link cannot immediately be output on the OS-Link, since the packet length (which must be sent first) is not known until the EOP or EOM has been received. Thus there is a 34-byte FIFO buffer to hold the bytes and a 6-bit counter to count them. When the EOP/EOM has been received, the count byte is sent, followed by the contents of the buffer; the counter is then reset to zero ready for the next packet. Buffering and counting of the next packet begins immediately. If the packet is less than 7 bytes in length it is padded out to 7 bytes (making eight bytes in all including the length byte) with extra trailing bytes which must be discarded by the inputting software. This is done to enable short bytes to be received with a single 8-byte input, which improves performance overall.

The size of the FIFO buffer allows a full-size packet to have 2 bytes of header. If the packet exceeds this size, this is an error, which causes an error message to be sent from control link 0 (see table 1.8, page 278).

Note that, if packets are not received continuously, the buffering in the chip can be used to match different data rates on the two types of link. Data can be received from the DS-Link at a high rate and re-transmitted more slowly from the OS-Link. If packets are sent to the DS-Link at a higher average rate than they can be re-transmitted from the OS-Link, the FIFO will become full, after which the flow-control mechanism of the DS-Links will allow further data to be received only as fast as it can be sent from the OS-Link.

1.5 Control protocols

This section describes the different control protocols used on T2/T4/T8-series and T9-series components. The two types of control protocol conversion that the IMS C100 supports are described in the following section.

1.5.1 T2/T4/T8-type control

T2/T4/T8-series transputers are controlled by means of **Reset**, **Analyse** and **Error** pins (RAE) on each device, and are inspected and booted by means of a special protocol across the transputer links (T2/T4/T8-type control).

The falling edge of **Reset** initializes the transputer, triggers the memory configuration sequence and starts the bootstrap routine.

The processor and the OS-Links start after reset. The transputer obeys a bootstrap program which can either be in off-chip ROM or can be received from one of the links.

A software error, such as arithmetic overflow, array bounds violation or divide by zero, causes an error flag to be set in the transputer processor. The flag is directly connected to the **Error** pin. Both the flag and the pin can be ignored, or the transputer stopped. Stopping the transputer on an error means that the error cannot cause further corruption.

As well as containing the error in this way it is possible to determine the state of the transputer and its memory at the time the error occurred.

If **Analyse** is taken high when the transputer is running, the transputer will halt at the next descheduling point. From **Analyse** being asserted, the processor will halt within three time slice periods plus the time taken for any high priority process to complete. **Reset** may then be asserted. When **Reset** comes low again the transputer will be in its reset state, but the registers contain information on the state of the machine when it was halted by the assertion of **Analyse**, permitting analysis of the halted machine.

Input links will continue with outstanding transfers. Output links will not make another access to memory for data but will transmit only those bytes already in the link buffer. Providing there is no delay in link acknowledgement, the links will be inactive within a few microseconds of the transputer halting.

1.5.2 T9-type control

T9-series transputers are controlled by a pair of control links on each device (T9-type control). The control links on all T9-series transputer family products allow a separate control network to be used to assist in configuring, booting, error handling, resetting and analysing processors and other components connected in a system, even in the presence of errors on the data communications links in the network. Many of these functions can also be performed directly by software running on an IMS T9000 transputer.

Each IMS T9000 transputer has two bidirectional control links, **CLink0** and **CLink1**, which use the same electrical and packet level protocols as the DS data links. **CLink0** will be connected via a control link network to one of the data links of a controlling IMS T9000 transputer, or to a different host via a link adaptor. All communications with the controlling processor are via **CLink0**. **CLink1** is provided to allow T9-series product family components to be connected in a daisy-chain. This allows a simple physical connectivity to be used for the controlling network, as shown in figure 1.14.

The controlling network provides each device with a virtual link connected to the control process. When the network is initialized the first communication with each device programs the label and return addresses to establish the virtual link between the control process and that device. The label address determines whether a packet arriving on **CLink0** is for that device (i.e. when the header matches the label), and if not, the packet is forwarded along **CLink1** until it reaches its destination. The message header (label address) and return address are both two bytes.

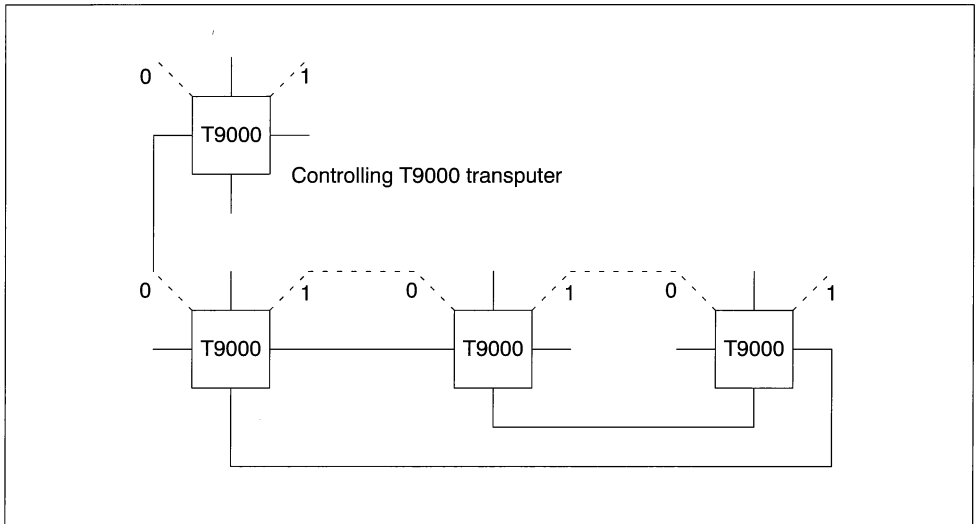


Figure 1.14 A daisy-chained control link network

A high level protocol is defined for the controlling network to allow the control process to issue commands to, and receive responses from, devices in the network. Commands are sent as normal packets but with the first byte after the header containing a command code, which may be followed by additional data.

Control link protocols

The control link implements one end of a virtual link, along which the specific message protocol is enforced. There are two layers of protocol, as follows:

- **Message level handshakes**

Each message from the controlling process is handshaken before another message is sent (the exceptions to this are *Reset* and *RecoverError* messages, which can be sent even if no handshake for the previously transmitted command has been received). Similarly, an *Error* message from the controlling process must be handshaken before another *Error* message can be sent.

- **Packet level acknowledgements**

Each data packet sent on a DS-Link is acknowledged before the next is sent. The only exception to this is the *RecoverError* command. A *RecoverError* command can be sent even if no acknowledgement for the previously sent packet has been received. This enables the control process to handle error conditions in the network.

1.6 Control protocol conversion

To enable T2/T4/T8-series subsystems to be easily incorporated into T9-series transputer systems (and vice versa) the IMS C100 converts between the two control systems described above. The subsystem to be connected can be controlled either through **Reset**, **Analyse** and **Error** signals, or through control link (**CLink0**) of the IMS C100.

RAE master

The **TReset** and **AnalyseIn** pins act as master of the IMS C100 and errors are reported by the **EventOut** pin (acting as an error pin) of the connected IMS T9000 transputer.

The IMS T9000 transputer is booted from ROM. The ROM code sets the IMS T9000 DS-Links in byte-stream mode and emulates the boot time behaviour of the T2/T4/T8-series transputer. That is, it allows code to be booted down the data links of the IMS T9000 transputer in the same way as for T2/T4/T8-series transputers.

CLink0 master

The **CLink0** acts as master of the IMS C100. Commands received on **CLink0** are converted either to signals on the **Reset** and **Analyse** pins, or into T2/T4/T8-series *Boot*, *Peek* and *Poke* messages transmitted along **OSLink0**. Signals on the **Error** pin are converted to *Error messages* transmitted along **CLink0**.

The mode determines which conversion is to be carried out. The IMS C100 has two control links, one for issuing and receiving commands (**CLink0**) and one for daisy-chaining (**CLink1**). In mode 0 **CLink0** of the IMS C100 generates commands. In modes 1, 2 and 3 it is receptive to commands.

The two types of conversion are described in more detail below.

1.6.1 RAE master control (mode 0)

In mode 0, T2/T4/T8-type control is the master control and the IMS C100 translates **Reset** and **Analyse** pin signals from the T2/T4/T8-series transputer to control link commands of T9-series transputers.

In this mode the IMS C100 cannot receive commands from an IMS T9000 transputer, but can issue commands via **CLink0**. In effect the IMS C100 acts like the control process in a network of IMS T9000 transputers, see figure 1.15. The commands generated are the same as those received by an IMS T9000.

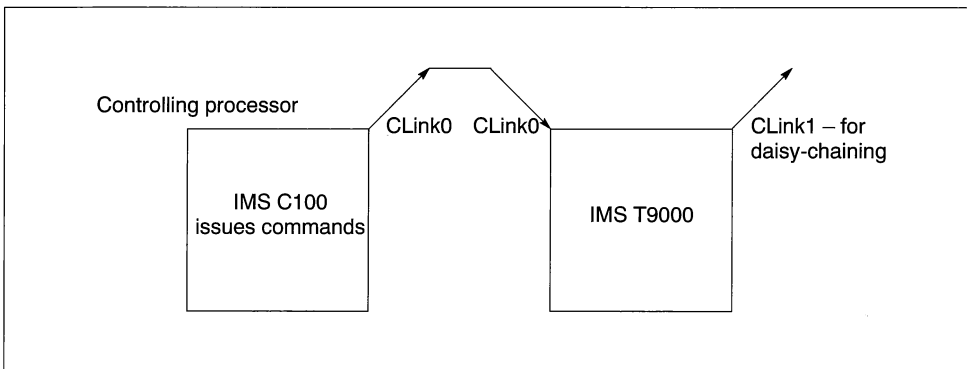


Figure 1.15 RAE master control

Figure 1.16 illustrates mode 0 in which the IMS C100 converts a T2/T4/T8-series interface to a T9-series interface by translating the **Reset** and **Analyse** pin signals from the T2/T4/T8-series transputer into commands sent via **CLink0** to the IMS T9000 transputer.

Note that in this mode **CLink0** of the IMS C100 is connected to **CLink0** of the IMS T9000 transputer. The standard connection of control links is to connect **CLink0** to **CLink1**.

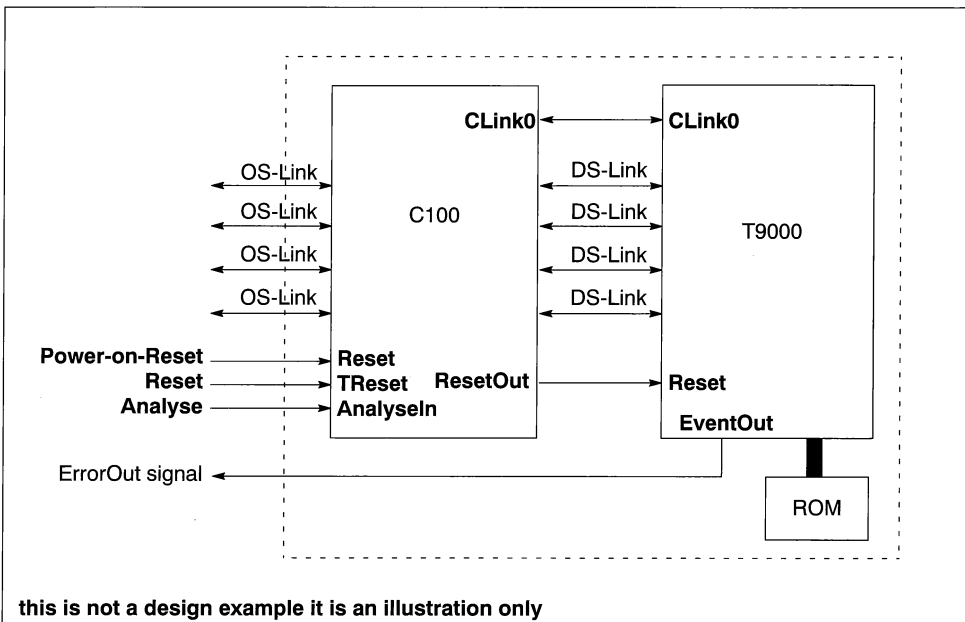


Figure 1.16 RAE master control – mode 0

Control commands sent by the IMS C100 in RAE master control mode

The following section details the command messages which can be sent from the IMS C100 to a connected IMS T9000 and describes the effect of the commands on the IMS T9000.

Each command package is acknowledged by an acknowledge packet which is a packet containing no data and terminated by an EOP token. The exception to this is the *RecoverError* command which can be sent even if no acknowledge for the previously sent packet has been received. In addition the higher level control protocol requires that all command messages are acknowledged by a response message before the control process can send another command message to the same device, so appropriate responses must be generated by the IMS C100 in this mode of operation. However, the exception to this is the *Reset* and *command message* which may be sent before the handshake for the previous command has been received.

The response message is a handshake code corresponding to the command message. Each message is preceded by the return header and followed by an EOM token. Command response codes are the same as the command codes except with the top bit inverted. Some of the handshake messages include a status byte which indicates whether the received command was valid as defined below.

- Status byte has value 0 if command is valid.
- Status byte has value 1 if command is invalid or has failed for some reason.

Figures 1.17 and 1.18 show the command packets sent by the IMS C100 and the handshake packets received by the IMS C100.

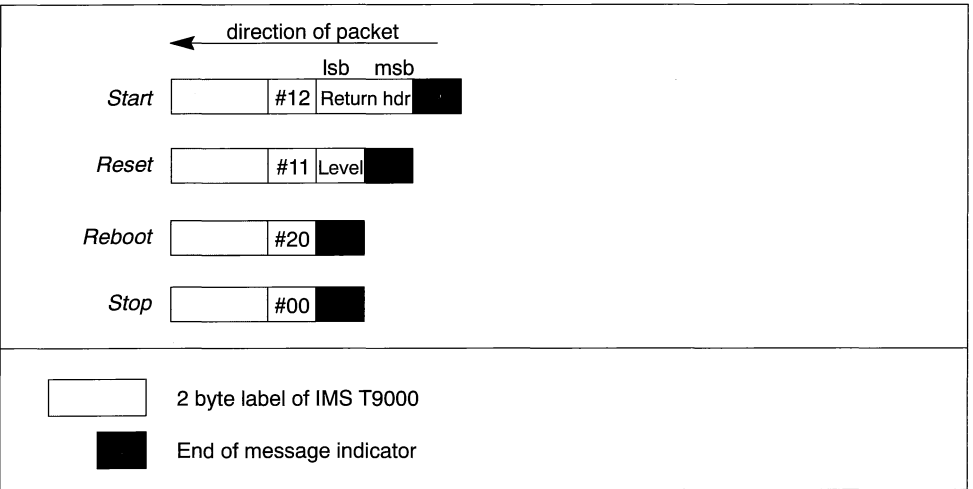


Figure 1.17 Control link command messages sent by the IMS C100 in mode 0

Start

Programs the T9000's **CLink0** by allocating a label and return header.

The header of the first *Start* command after power-up is taken as the 'label' for the device and all subsequent messages with the same header are interpreted by that device. Messages with a different header are forwarded (if possible) via **CLink1**.

The *Start* command programs the return header of the IMS T9000. The return header is 2 bytes long, with the least significant byte (lsb) being the first byte transmitted following the command code.

Reset

The *Reset* command message causes some or all of the subsystems of the IMS T9000 to be reset. The level of reset is encoded in the 'level' byte of the command message. There are three different levels of

reset to which the IMS T9000 responds. Reset 1 is equivalent to a hard reset except that the control system is not affected; reset 2 resets all subsystems of the IMS T9000 except the control system, and leaves the configuration and the PMI activity unchanged; reset 3 simply halts the processor. Refer to the *T9000 Datasheet* for more information on IMS T9000 reset levels.

Reset3 is the only reset level sent from an IMS C100.

Note that a *Reset* command may cause a handshake for a previously transmitted command to be: terminated prematurely (with an EOM token); completed with a failure status; or suppressed entirely.

A *Reset* command with an invalid level is handshaken with a failure status.

Stop

The *Stop* command message stops the processor 'cleanly' so that register values are preserved for debugging. It acts like the **Analyse** pin on the T2/T4/T8-series transputer.

Reboot

The *ReBoot* command causes the processor to reboot from ROM. It starts executing, with a workspace pointer (**Wptr**) (which must be word-aligned) and instruction pointer (**Iptr**) read from memory locations at the top of the address space.

The IMS T9000 reads a **Wptr** and an **Iptr** from two fixed locations near the top of memory:

Boot from ROM **Iptr** address is #7FFFFFF8

Boot from ROM **Wptr** address is #7FFFFFFC

Handshake and Error messages received by the IMS C100 from the IMS T9000

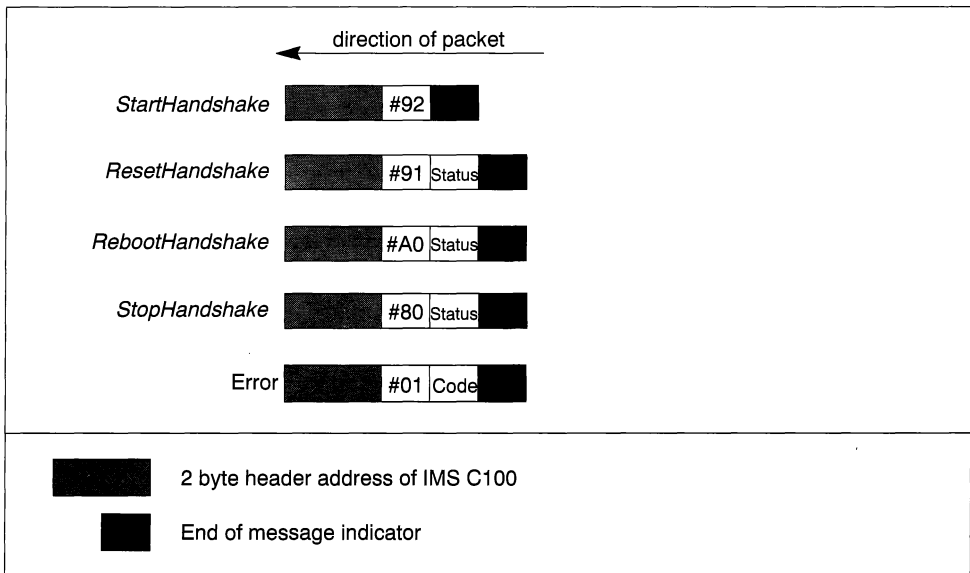


Figure 1.18 Handshake messages sent by the IMS T9000 to the IMS C100 in mode 0

Behavior of the control system in RAE master mode

After power-on reset a *Start* command is sent from the IMS C100 to **CLink0** of the attached IMS T9000 transputer. The IMS T9000 returns a *StartHandshake*. This forms the virtual link between the controlling process (IMS C100) and the node (IMS T9000). The attached IMS T9000 boots from ROM.

The IMS C100 sends *Reset*, *Stop* and *Reboot* commands via **CLink0** in response to reset and analyse pins being toggled in given sequences, as described below.

Note that the IMS C100 does not interpret the 'pre-boot protocol' of T2/T4/T8-series transputers. This is done by software, loaded from the boot ROM, running on the attached IMS T9000.

Reset

The **TReset** pin indicates transputer reset of the connected T2/T4/T8-series transputer. If the **TReset** pin is asserted with the **AnalyseIn** pin low, the IMS C100 is reset. The signal is reproduced on **ResetOut** which causes the IMS T9000 to reset.

The **Reset** pin is provided in this case for systems which separate power-on-reset from transputer reset. When the **Reset** pin is asserted it resets both the IMS C100 and the attached IMS T9000 (by being reproduced on **ResetOut**).

TReset asserted (with **AnalyseIn** low) and/or **Reset** asserted is seen by both the IMS C100 and the attached IMS T9000 as a power-on reset.

Reboot

The *Reboot* command causes the attached IMS T9000 to boot from ROM using a **Wptr** and **lptr** from a fixed location in ROM. The ROM code, configures the IMS T9000, sets the links into byte-stream mode, starts them, and then emulates the T2/T4/T8-series pre-boot protocol.

Analyse

In response to the **AnalyseIn** pin being asserted the IMS C100 will send a *Stop* command from **CLink0** to the IMS T9000. The *Stop* command causes the processor to be stopped whilst preserving register values. The IMS T9000 returns a *StopHandshake*. When the **TReset** pin of the IMS C100 is asserted, the IMS C100 sends a *Reset* message (reset level 3 - to stop the CPU). When both **TReset** and **AnalyseIn** are deasserted, the IMS C100 sends a *Reboot* message. This restarts the ROM code. If this code executes a *testpranal* instruction it can take special action to assist the debugger before it repeats the above-mentioned pre-boot sequence.

The **TReset** and **AnalyseIn** signals are used in this mode only and are ignored in CLink0 master control modes (modes 1, 2 and 3).

Errors

Software, supplied to customers, running on the IMS T9000 emulates the pre-boot protocol of T2/T4/T8-series transputers. It also emulates the Error pin behavior of a T2/T4/T8-series transputer by performing an output on an event output channel whenever a T2/T4/T8-series transputer would assert its Error pin.

If an error occurs on the IMS T9000, this is signalled by the **EventOut** pin. It also causes an *Error* message to be sent from **CLink0** of the IMS T9000, which is received by **CLink0** of the IMS C100 and ignored.

In this mode there is nowhere for the IMS C100 to send *Error* messages.

If an error occurs on the control link, the link is reset, then restarted and a *RecoverError* command sent.

If an error occurs on one of the DS data links, the link is reset, any incomplete packets received from the IMS T9000 are discarded and the data protocol converter and OS-Link are reset. If the error occurred during a packet transmission, data in one direction will stop, as either the IMS C100 or the IMS T9000 will be waiting to receive a packet from the other. If no packet was being transmitted, data transfer can continue. If transfer is halted, software on the IMS T9000 may eventually execute a *resetch* (reset channel) instruction, which causes a link error to be seen by the IMS C100, resulting in the reset of the link conversion unit and the OS-Link. Communication may then be re-started.

1.6.2 CLink0 master control (modes 1, 2 and 3)

In modes 1, 2 and 3, T9-type control is the master control. In these modes **CLink0** and **CLink1** act as a daisy chain (see figure 1.19) with **CLink0** saving the header of the first packet it receives, and only inputting subsequent packets with the same header. Packets with a different header are relayed out of **CLink1**. All packets received on **CLink1** are relayed out of **CLink0**. There is a fair arbiter to deal with the case that the IMS C100 needs to send a packet at the same time as a packet arrives on **CLink1**. Note this is identical to the daisy-chaining behavior of the IMS T9000 (as described in the *Control System Chapter of The T9000 Datasheet*).

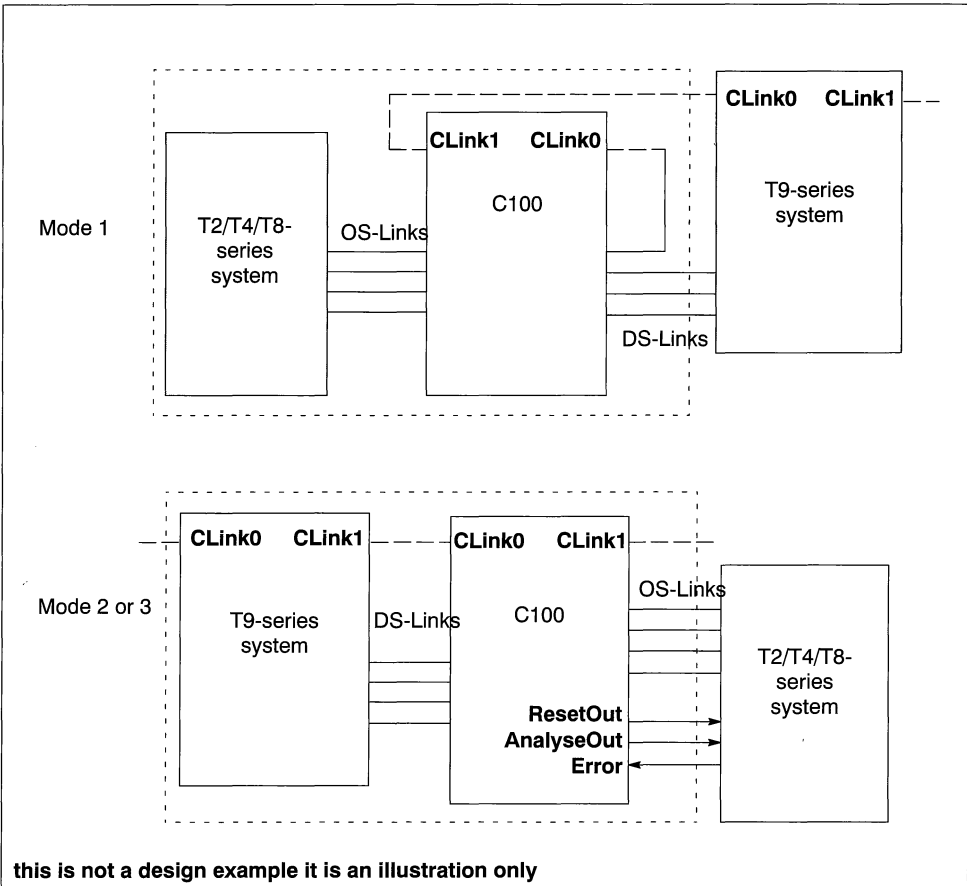


Figure 1.19 **CLink0** master control – modes 1, 2 and 3

All packets received on **CLink0** with the same header as the first packet received are input by the IMS C100 and decoded as either acknowledge packets (which allow further messages to be sent by the IMS C100), or as messages. Messages are further subdivided into commands and handshakes. A handshake indicates that a previously sent message has been received, and so another can be sent.

Control commands sent by the controlling processor (IMS T9000) to the IMS C100

The commands recognized by the IMS C100 in modes 1, 2 and 3 include those accepted by an IMS T9000 transputer. However, execution of the commands is adapted to T2/T4/T8-series behavior.

The following section details the command messages which can be sent from the T9-series control process to the IMS C100.

Each command packet is acknowledged by an acknowledge packet which is a packet containing no data and terminated by an EOP token. The exception to this is the *RecoverError* command which can be sent even if no acknowledge for the previously sent packet has been received. In addition the higher level control protocol requires that all command messages are acknowledged by a response message before the control process can send another command message to the same device, so appropriate responses must be generated by the IMS C100 in this mode of operation. However, the exceptions to this are the *Reset* and *RecoverError* command messages which may be sent before the handshake for the previous command has been received.

The response message can contain the result of a *Peek* or *Identify* command, or it may be simply a handshake code corresponding to the command message. Each such message is preceded by the return header and followed by an EOM token. Command response codes are the same as the command codes except with the top bit inverted. Some of the handshake messages include a status byte which indicates whether the received command was valid as defined below.

- Status byte has value 0 if command is valid.
- Status byte has value 1 if command is invalid or has failed for some reason.

Commands sent which cannot be converted to T2/T4/T8-series actions, or commands which are illegal in certain states, are handshaken with status set to 1.

Figures 1.20 and 8.7 which show the command packets received by the IMS C100 and the handshake packets returned to the controlling process respectively.

Start

This command programs or re-programs the return header of the IMS C100. The return header is 2 bytes long, with byte 0 being the first byte transmitted following the command code. Note that if this command is used to re-program the return header, the acknowledge for the command message packet will be sent with the *old* header, whilst the handshake will be sent with the *new* header.

The *Start* command must be the first command received. If an error occurs before the first *Start* command is received, the *StartHandshake* will be returned before the *Error* message is sent.

Reset

This command resets the IMS C100 and consequently the connected T2/T4/T8-series transputer.

The *Reset* command message includes a 'level' parameter. The level of reset is encoded in the 'level' byte of the command message. There are two different levels of reset to which the IMS C100 responds. Levels 2 and 3 set the **ResetOut** pin high. A *ResetHandshake* with a success status indicator (0) is sent on completion. Any other value of the level parameter causes the status of the *ResetHandshake* to be a failure (1).

Note that a *Reset* command must be followed by a '*CPoke*-end-Reset' to de-assert **ResetOut**. A *CPoke*-end-Reset is a *CPoke* to the IMS C100 **Command** register with bit 5 set (see table 1.16).

See section 1.8 for more information on reset levels.

Note that a *Reset* command may cause a handshake for a previously transmitted command to be: terminated prematurely (with an EOM token); completed with a failure status; or suppressed entirely.

Note that *Reset level 1* is not supported directly by the IMS C100. It can be achieved by performing a *Reset2*, followed by *CPoke* commands to set the configuration of the DS-Links to their default values and force the data and strobe outputs low.

Identify

The *Identify* command message causes the IMS C100 to respond with a handshake containing an identifier unique to the device type. This can be used to check the contents of a network. The lower 16 bits of the identifier are the same as the contents of the **DeviceID** register (see section 1.9.3); the upper 16 bits are zero.

Note, the *Identify* command code is identical for all IMS T9000 family devices.

Stop

The *Stop* command message is used to cause the attached transputer to come into a quiescent state whilst preserving information for debugging.

This command sets the **AnalyseOut** pin high (resulting in the **Analyse** pin on the connected T2/T4/T8-series transputer being asserted) and generates a *StopHandshake* message. To stop a T2/T4/T8-series transputer the **Analyse** and **Reset** pins must be asserted and deasserted in the correct order (see page 280). In order to achieve this a *Reset3* command must be sent after the *Stop* command to assert the **Reset** pin. *CPoke* commands to the IMS C100 **Command** register must be used to de-assert the pins.

Note that a *Stop* command must be followed by a '*CPoke*-end-Analyse' to de-assert **AnalyseOut**. A *CPoke*-end-Analyse is a *CPoke* to the IMS C100 **Command** register with bit 6 set (see table 1.16).

RecoverError

This command is used in error recovery on the control links (see section 1.8.5). It restores the protocol after a link error in the control link system. Note that if there is an unhandshaken error, the *RecoverError* handshake will be returned before the error message is sent.

CPeek

The *CPeek* command includes a 2 byte address which points to a register in the configuration address space. The handshake message returns the value stored at the given address. If the address is invalid the handshake message returns an invalid status.

CPoke

The *CPoke* command includes a 2 byte address and 4 bytes of data. It is used to set the value of a configuration register. It writes the data to the configuration space register at the given address. If the address contained in the command message was invalid the status byte of the handshake message indicates failure.

Note, some registers do not have a value, but writing to them causes some action to occur.

Peek16 and Peek32

The *Peek16* command determines the value of a memory location of the attached 16 bit (T2) transputer. It includes a 2 byte address which points to the memory location. The address location must be word-aligned. The handshake message returns the value of the location, unless the status byte indicates that the command failed. If the status byte of the handshake message indicates failure, this means that a reset or recover was received before the command completed.

The *Peek32* command determines the value of a memory location of the attached 32 bit (T4/T8) transputer. It includes a 4 byte address which points to the memory location. The address location must be word-aligned. The handshake message returns the value of the location, unless the status byte indicates that the command failed.

Poke16 and Poke32

The *Poke16* command writes data to a memory location of an attached 16 bit transputer. The 2 byte address of the location (which must be word-aligned) and the value to be written (2 bytes of data) are included in the command message. If the status byte of the handshake message indicates failure, this means that a reset or recover was received before the command completed.

The *Poke32* command writes data to a memory location of an attached 32 bit transputer. The 4 byte address of the location (which must be word-aligned) and the value to be written (4 bytes of data) are included in the command message. If the status byte of the handshake message indicates failure, this means that a reset or recover was received before the command completed.

Boot16 and Boot32

The *Boot* command starts a 'booting' sequence. The booting sequence makes the loading of code via the control link more efficient than performing a series of pokes. The *Boot* command message contains the length in bytes (the length value must be a multiple of 4) of the boot code to be loaded, and the address (which must be word-aligned) to where in memory the code is to be loaded from.

When the IMS C100 is attached to a T2/T4/T8-series transputer, the address is discarded and the length is restricted to be less than 256 bytes.

BootData

The *BootData* command loads code as part of a booting sequence. Each *BootData* command message contains 16 bytes of code. A handshake is generated after transferring a complete packet. The boot length is decremented each transfer and on the final packet only the outstanding number of bytes from the 16 are forwarded. The remaining padding is input and ignored before the handshake is generated.

Run16 and Run32

The *Run16* and *Run32* commands are not applicable to a T2/T4/T8-series transputer and are ignored and handshaken with a failure indication.

Reboot

The *Reboot* command causes a reboot from ROM.

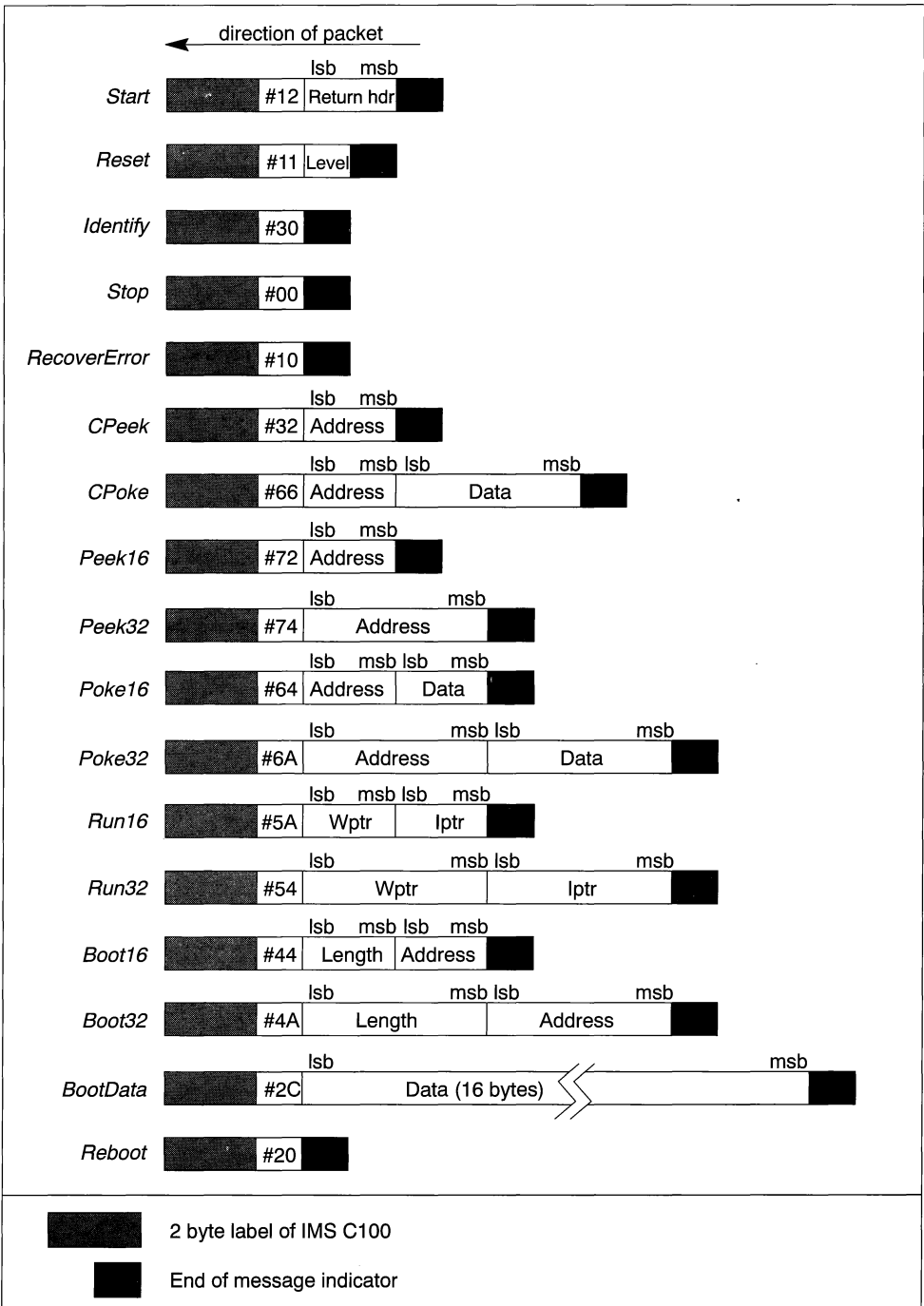


Figure 1.20 Control link command messages received by IMS C100 in modes 1, 2 and 3

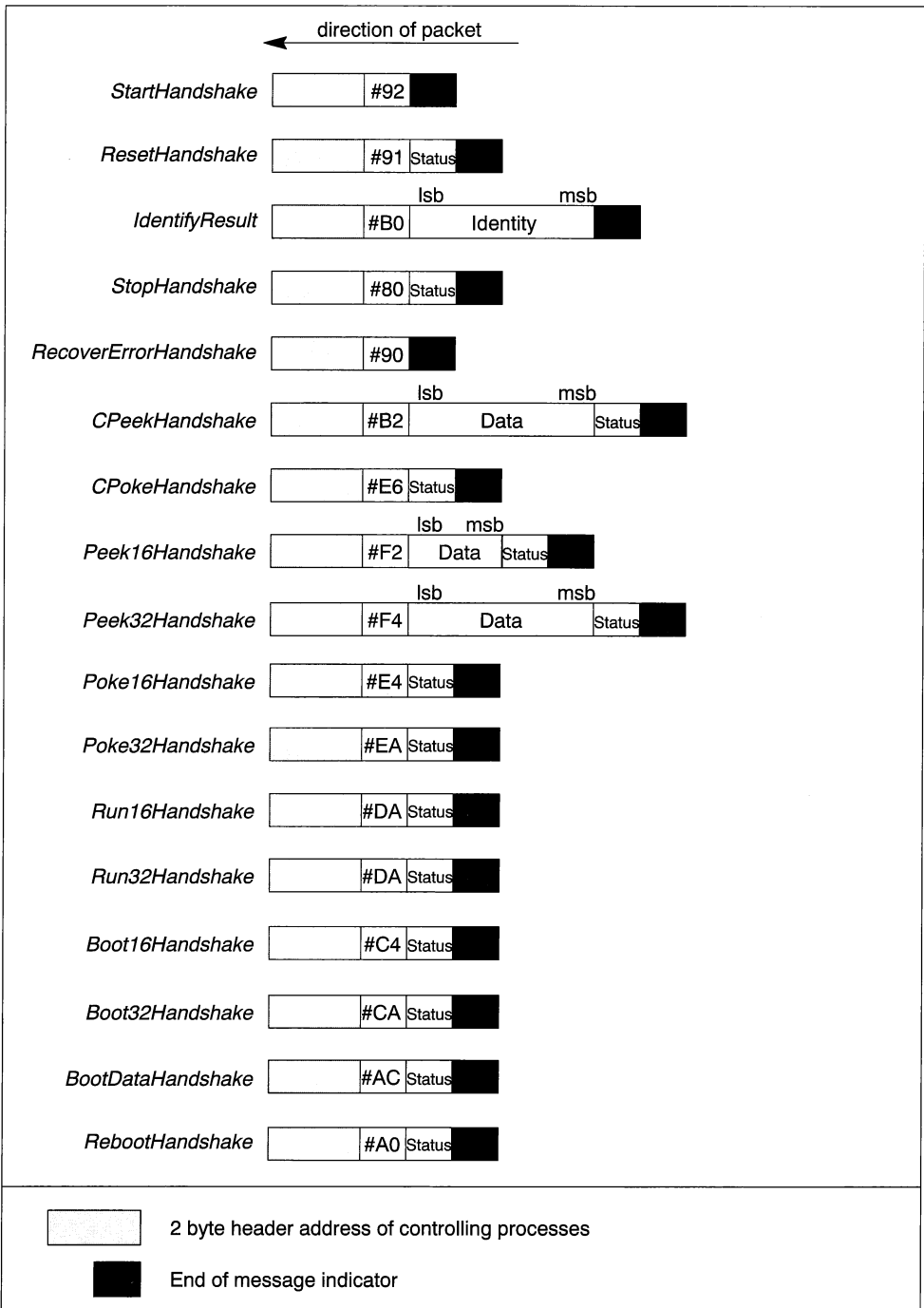


Figure 1.21 Handshake messages sent by IMS C100 to the controlling process in modes 1, 2 and 3

Errors

The IMS C100 can send an *Error* message to the controlling process to indicate that an error has occurred. The *Error* message contains an error code which determines the type of error, as given in table 1.8. All the error codes must be handshaken by the control processor with the *ErrorHandshake* command.

Errors are reported by sending an *Error* message with the corresponding code or in the case of an error on **CLink0**, by causing a disconnection. Software at the control processor can then take appropriate action.

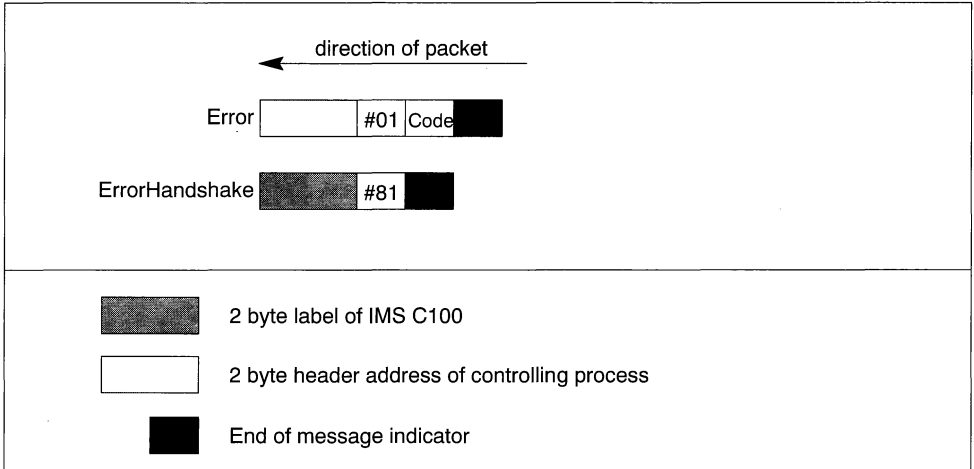


Figure 1.22 Error message

Error code	Cause of error	Priority
#C2	Parity or disconnect error on CLink1	Highest
#C1	Protocol error on CLink0 e.g. bad command length, unexpected acknowledge	
#C0	Unrecognized command code on CLink0	Lowest
#01	Signal on Error pin	
#80	Parity or disconnect error on DSLlink0	
#81	Parity or disconnect error on DSLlink1	
#82	Parity or disconnect error on DSLlink2	
#83	Parity or disconnect error on DSLlink3	
#10	Overlong packet on OSLink0	
#11	Overlong packet on OSLink1	
#12	Overlong packet on OSLink2	
#13	Overlong packet on OSLink3	
#20	Invalid count (i.e. count = 0) on OSLink0	
#21	Invalid count (i.e. count = 0) on OSLink1	
#22	Invalid count (i.e. count = 0) on OSLink2	
#23	Invalid count (i.e. count = 0) on OSLink3	

Table 1.8 Error codes

OS-Link 0 special function – modes 2 and 3

In modes 2 and 3, the control links are the system master, and the default assumption is that at least **OSLink0** is connected to an unbooted T2/T4/T8-series transputer. In this case **OSLink0** is used to transmit the pre-boot protocol of the T2/T4/T8-series transputer until the transputer is booted. Commands to peek, poke and boot the T2/T4/T8-series transputer, arriving down **CLink0**, are converted to T2/T4/T8-series protocol and sent down **OSLink0**.

The default assumption of **OSLink0** being connected to an unbooted T2/T4/T8-series transputer can be controlled by the **booted** flag, which when set indicates that the connected transputer has booted. The **booted** flag can be set by means of a *CPoke* command. The **booted** flag is set automatically by the booting sequence (*Boot16* or *Boot32* command followed by *BootData* commands) or by the *Reboot* command. The **booted** flag is reset by the *Reset* command.

Commands which correspond to the protocol of an unbooted T2/T4/T8 transputer

There are four control link commands which correspond to the special protocol of an unbooted T2/T4/T8-series transputer. These cause messages to be generated from **OSLink0** in modes 2 and 3, for which the behavior of **OSLink0** is defined to be special. The assumed length (N) of addresses and data can be either 16 or 32 bits depending on whether a command is being sent to a 16 bit (T2) or 32 bit (T4/T8) transputer.

Peek

On receipt of a *PeekN* command, and the associated peek address, the IMS C100 sends from **OSLink0** the following sequence of bytes, where the address is a 2 or 4 byte array depending on the word length of the command:

```
1(BYTE);address[0];...address[N]
```

When the last byte has been sent and acknowledged the IMS C100 awaits an associated response. A *PeekNHandshake* is returned with 2 or 4 bytes of data and a status byte of 0.

If the communication does not complete (for example if there is no transputer connected), the peeking process will not receive a *PeekNHandshake* and can time-out, and if required, reset the IMS C100 with a *Reset* command.

PeekN commands are only permitted if the **booting** and **booted** flags are not set (i.e. 0) and the **ResetOut** and **AnalyseOut** pins are low.

Poke

On receipt of a *PokeN* command, and the associated poke address and data, the IMS C100 sends from **OSLink0** the following sequence of bytes. Where the address and data are each 2 or 4 byte arrays, depending on the word length of the command.

```
0(BYTE);address[0];...address[N];data[0];...data[N]
```

The command is acknowledged immediately, and if and when the last byte of the above communication is acknowledged, a *PokeNHandshake* is returned with a status of 0.

If the communication does not complete (for example because there is no transputer connected after all), the poking process will not receive a *PokeNHandshake* and can time-out, and reset the IMS C100.

PokeN commands are only permitted if the **booting** and **booted** flags are not set and the **ResetOut** and **AnalyseOut** pins are low.

Boot

On receipt of a *BootN* command, and the associated boot address and length, the IMS C100 sends the first byte of the length from **OSLink0** and discards the address.

The *BootN* command is acknowledged immediately, and if and when the length byte is acknowledged by a connected transputer, a *BootNHandshake* response is sent with a status of 0.

The value of the length byte is kept by the IMS C100 as a count, and that number of bytes are then received by **CLink0**, as a sequence of *BootData* messages. The bytes are sent out on the OS-Link. Each arriving *BootData* message is acknowledged immediately, but not handshaken until all its data bytes have been sent and acknowledged on the OS-Link. Once all bytes have been sent and acknowledged a *BootDataHandshake* is sent with a status byte of 0. If more bytes are received than were allowed for in the count, the extra bytes are discarded. If a *BootData* command is received when the count is reduced to zero, all its data is discarded and a *BootDataHandshake* with a failure status indicator is returned.

The boot count must be a multiple of 4 bytes and the last *BootData* command padded out to 16 bytes with 0, 4, 8 or 12 zero bytes.

BootN commands are only permitted if both the **booting** and **booted** flags are not set.

After a *BootN* command has been received, the **booting** flag is set, and any further *PeekN*, *PokeN* or *BootN* commands are invalid. Once the number of bytes as allowed for in the count of the *BootN* command have been received, the **booting** flag is unset, and the **booted** flag is set; *BootData* commands are then also invalid. All such invalid commands are acknowledged and handshaken immediately, but with a status byte of 1 indicating failure. No other action is taken. The **booting** and **booted** flags are reset by any *Reset* command.

Resetting and Analyzing

Reset

On receipt of a *Reset* command on **CLink0**, the IMS C100 asserts its **ResetOut** pin, resets the **booting** and **booted** flags, and may also reset its internal buffers and OS-Links (see section 1.8.5 for details on the effects of the different levels of reset). The **ResetOut** pin is deasserted by a *CPoke* command to bit 5 in the IMS C100 **Command** register (see table 1.16).

Analyse pin

On receipt of a *Stop* command on **CLink0**, the IMS C100 asserts its **AnalyseOut** pin. It acknowledges the *Stop* command and returns a *StopHandshake* message. The controlling software dealing with the T9-series device waits for a certain time and then sends a *Reset* message. When the IMS C100 receives this message, it asserts the **ResetOut** pin. It deasserts the **AnalyseOut** and **ResetOut** pins by *CPoke* commands to the corresponding bits in the IMS C100 **Command** register (see table 1.16). The minimum timings for asserting and deasserting these pins are given in section 1.13.1.

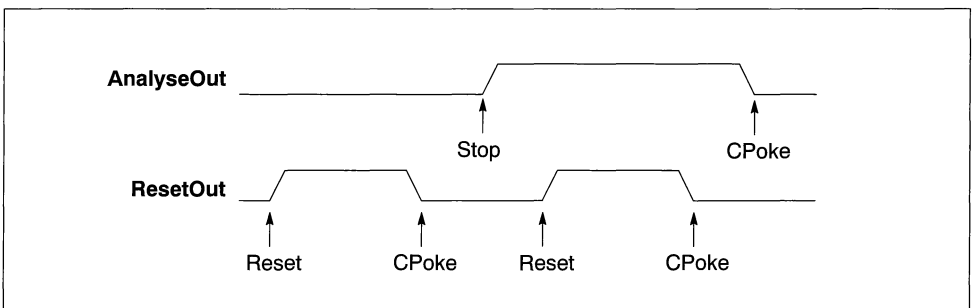


Figure 1.23 Resetting and analyzing in modes 1, 2 and 3

Note that **ResetOut** and **AnalyseOut** must be de-asserted using *CPokes* to the **Command** register (see table 1.16) before continued operation.

1.7 Links

1.7.1 Data links

The IMS C100 has eight data links. **OSLink0-3** are oversampled links and **DSLlink0-3** are data/strobe links. Each OS-Link is paired with a DS-Link, for data protocol conversion. All pairs of links perform one or other type of conversion, depending on mode.

Each pair of links is joined by a conversion unit. **OSLink0** can be diverted into the control conversion unit by a switch which is controlled by the **booted** flag. Refer to section 1.6.2 for description of **OSLink0** special function in modes 2 and 3.

The OS and DS-Links are TTL compatible.

Links are not synchronized with **ClockIn** and are insensitive to their phases. Thus links from independently clocked systems may communicate, providing only that the clocks are nominally identical and within specification.

Data link speed pins

There are four pins to set the operating speed of the links. **OSLink0HalfSpeed** and **OSLink123HalfSpeed** pins set the operating speed of the OS-Links and **DSLlink0HalfSpeed** and **DSLlink123HalfSpeed** pins set the speed of the DS-Links. The **OSLink0HalfSpeed** pin enables the speed of **OSLink0** to be set independently of OS-Links 1, 2, 3 and the **DSLlink0HalfSpeed** pin enables the default speed of **DSLlink0** to be set independently of DS-Links 1, 2, 3.

OSLink0-3 support a communication speed of 10 Mbits/s. In addition they can be used at 20 Mbits/s which is determined by the **OSLink0HalfSpeed** and **OSLink123HalfSpeed** pins. The **DSLlink0HalfSpeed** and **DSLlink123HalfSpeed** pins can be used to set the speed of the DS-Links to 25 Mbits/s or 50 Mbits/s, (see table 1.9).

Note that if these pins are changed after power-on the IMS C100 is not guaranteed to function correctly until it has been reset.

OSLink0HalfSpeed	0	OSLink0 runs at 20 Mbits/s
	1	OSLink0 runs at 10 Mbits/s
OSLink123HalfSpeed	0	OSLink1-3 runs at 20 Mbits/s
	1	OSLink1-3 runs at 10 Mbits/s
DSLlink0HalfSpeed	0	DSLlink0 runs at 50 Mbits/s
	1	DSLlink0 runs at 25 Mbits/s
DSLlink123HalfSpeed	0	DSLlink1-3 runs at 50 Mbits/s
	1	DSLlink1-3 runs at 25 Mbits/s

Table 1.9 **LinkHalfSpeed** pins

DS-Link speeds in mode 0

The DS data links of the IMS C100 in mode 0 can be set to run at 25 Mbits/s or 50 Mbits/s depending on the setting of the **DSLlink0HalfSpeed** and **DSLlink123HalfSpeed** pins.

DS-Link speeds in modes 1, 2 and 3

The DS-Links of the IMS C100 in modes 1, 2 and 3 can support a range of communication speeds which can be programmed by writing to registers in the IMS C100 configuration space using the **CPoke** command via **CLink0**.

Only the transmission speed of a DS-Link is programmed as reception is asynchronous. This means that DS-Links running at different speeds can be connected, provided that each device is capable of receiving at the speed of the connected transmitter.

The transmission speed of all of the DS-Links (data and control links) on the IMS C100 are related to the speed of the 10 MHz base clock. This 10 MHz clock is multiplied by a programmable value to provide the root clock for all the DS-Links. The multiplication factor is programmed by writing to the **SpeedMultiply** bit in the **DSLlinkPLL** register in System Services, see section 1.9.3. This root clock is then optionally divided (by programming the **SpeedDivide** bits in the associated **DSLlinkMode** register) by 1, 2, 4 or 8 independently for each DS-Link, giving a range of speeds. This arrangement allows each link to be run at one of four transmission speeds, as shown in table 13.3.

SpeedMultiply	SpeedDivide1:0				BaseSpeed
	0:0	0:1	1:0	1:1	
	/ 1	/ 2	/ 4	/ 8	
8	80	40	20	10.0	10
10	100	50	25	12.5	10
12	Reserved	60	30	15.0	10
14	Reserved	70	35	17.5	10
16	Reserved	80	40	20.0	10
18	Reserved	90	45	22.5	10
20	Reserved	100	50	25.0	10

Table 1.10 DS-Link transmission speed in Mbits/sec

Note also that each DS-Link can be programmed to use a base rate clock of 10 MHz. At reset all DS-Links are configured to run at the **BaseSpeed** of 10 Mbits/sec. The **SpeedSelect** bit in the associated **DSLlink-Mode** register when set to 1 sets the respective DS-Link to the speed selected by the **SpeedMultiply** and **SpeedDivide** bits, as opposed to the default base speed of 10 Mbits/s.

Errors on DS-Links

DS-Link inputs can detect parity and disconnection conditions as errors. A single bit odd parity system detects single bit errors at the link token level. The protocol to transmit NUL tokens in the absence of other tokens enables disconnection of a DS-Link to be detected. A disconnection error indicates one of two things:

- the DS-Link has been physically disconnected;
- an error has occurred at the other end of the DS-Link, which has then stopped transmitting.

The **LinkError** bit in the **DSLlink0-3Status** registers flags that a parity and/or disconnection error has occurred on the **DSLlink0-3**. The bit fields **ParityError** and **DiscError** indicate when parity and disconnect errors occur respectively.

When a DS-Link detects a parity error on its input it halts its output. This is detected as a disconnect error at the other end of the DS-Link, causing this to halt its output also. Detection of an error causes the DS-Link to be reset. Thus, the disconnect behavior ensures that both ends are reset. Each end can then be re-started.

Note, when one end of a DS-Link is started up before the other end of a DS-Link, a disconnect error does not occur as no tokens have been received and once the other end of the DS-Link is started communication can commence. A disconnect error is only flagged once a token has been received on a DS-Link and transmission is subsequently interrupted.

The DS-Links are designed to be highly reliable within a single subsystem and can be operated in one of two environments, 'reliable' or 'unreliable' determined by the **LocalizeError** bit (set in **DSLlink0-3Mode** register) in each DS-Link. The **LocalizeError** bit is set on a per link basis, therefore it is possible to have some DS-Links in a system marked as reliable and others as unreliable. The consequence of a DS-Link error depends on which environment the DS-Link is in.

	LocalizeError bit	Link response to error	C100 control system	External device
Reliable links	0	Halt, signal error	Report error on CLink0	Reset
Unreliable links	1	Halt, auto restart, no error reported	—	—

Table 1.11 Effect of reliable and unreliable links

Reliable links

In the majority of applications, the communications system should be regarded as being totally reliable. In this environment errors are considered to be very rare, but are treated as being catastrophic if they do occur. This environment is the default on power-on reset, with all DS-Links having their **LocalizeError** bit set to 0. If an error occurs it will be detected and reported via a message sent along **CLink0**. Normal practice will then be to reset the subsystem in which the error has occurred and to restart the application.

Unreliable links

For some applications, for instance when a disconnect or parity error may be expected during normal operation, an even higher level of reliability is required. This level of fault tolerance is supported by localizing errors to the DS-Link on which they occur. This is achieved by setting the **LocalizeError** bit in the **DSLInk0-3Mode** register to 1. In this mode the link is considered 'unreliable'.

When in unreliable mode, processes must communicate using defensive software which can detect errors at the message level. These processes are responsible for establishing and maintaining a higher level flow control, using time-out to detect that a message has not completed. If an error occurs, packets in transit at the time of the error will be discarded or truncated, and the DS-Link will be reset without the error being reported via the control link. Code to implement error recovery must be run on each virtual channel. This application software is provided for the user in libraries contained in INMOS toolset products.

A DS-Link error in unreliable mode results solely in packets in transit at the time of the error being discarded or truncated.

Link connections

OS-Link connections

OS-Links are not synchronized with **ClockIn** and are insensitive to its phases. Thus OS-Links from independently clocked systems may communicate.

OS-Links are TTL compatible and intended to be used in electrically quiet environments, between devices on a single printed circuit board or between two boards via a backplane. Direct connection may be made between devices separated by a distance of less than 300 millimeters. For longer distances a matched 100 ohm transmission line should be used (with series matching resistors R_M of 56 ohms on the incoming link connected to the T2/T4/T8-series transputer), see figure 1.25. When this is done the line delay should be less than 0.4 bit time to ensure that the reflection returns before the next data bit is sent.

Note that IMS C100 OS-Links have a 100 ohm driver as for DS-Links.

Buffers may be used for very long transmissions. If so, their overall propagation delay should be stable within the skew tolerance of the OS-Link, although the absolute value of the delay is immaterial.

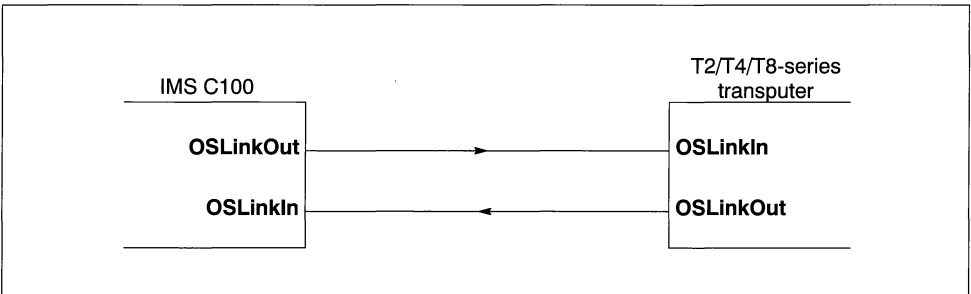


Figure 1.24 OS-Links directly connected

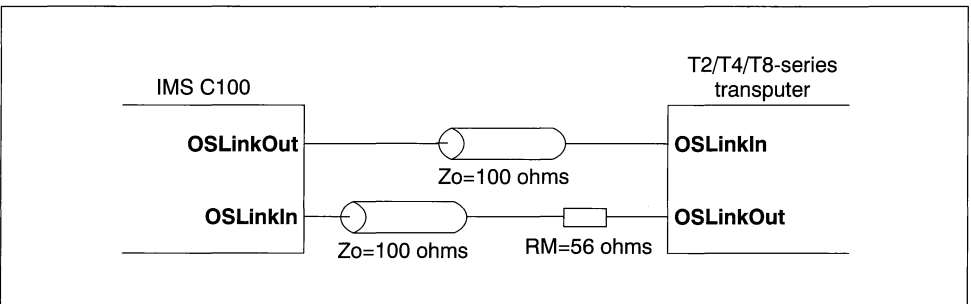


Figure 1.25 OS-Links connected by transmission line

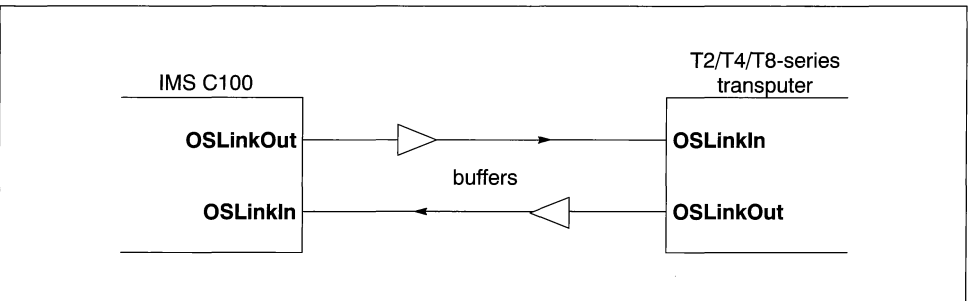


Figure 1.26 OS-Links connected by buffers

DS-Link connections

DS-Links are not synchronized with **ClockIn** and are insensitive to its phases. Thus, DS-Links from independently clocked systems may communicate.

DS-Links are TTL compatible and intended to be used in electrically quiet environments, between devices on a single printed circuit board or between two boards via a backplane. Direct connection may be made between devices separated by a distance of less than 300 millimeters. For longer distances a matched 100 ohm transmission line should be used, see figure 13.5.

The inputs and outputs have been designed to have minimum skew at the 1.5 V TTL threshold.

Buffers may be used for very long transmissions. If so, their overall propagation delay should be stable within the skew tolerance of the DS-Link.

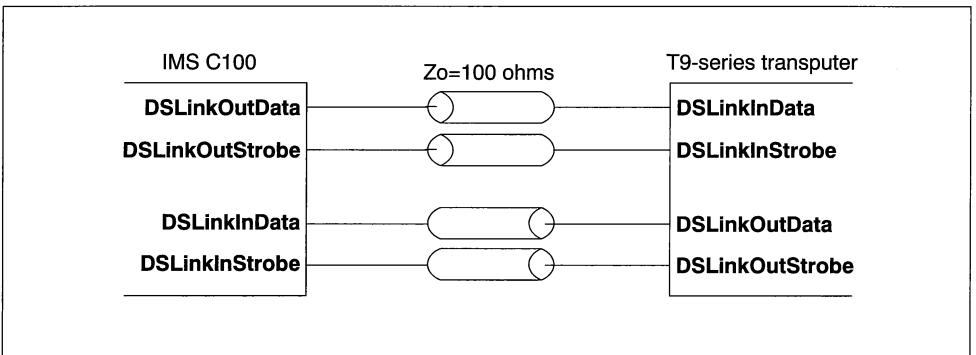


Figure 1.27 DS-Links connected by transmission line

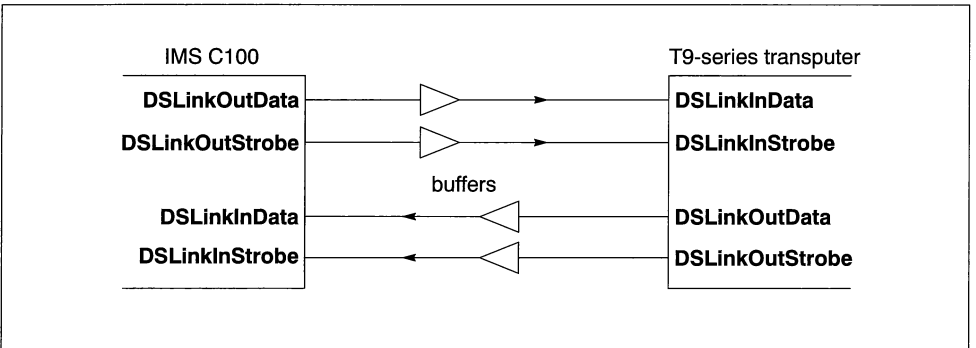


Figure 1.28 DS-Links connected by buffers

1.7.2 Control links

The IMS C100 has two bidirectional control links; **CLink0** and **CLink1**. They use the same electrical and packet level protocols as the data DS-Links (refer to section 1.3.2).

All communications with the controlling processor are via **CLink0**. **CLink1** provides a daisy-chain link, allowing a simple physical connectivity to be used for controlling networks.

The behavior of **CLink0** depends on the mode as detailed in section 1.6.

Control link speeds

After power-on the control links run at a default speed of 10 Mbits/s; this can be changed by setting the **SpeedMultiply** bit field in the **DSLInkPLL** register and the **SpeedDivide** bit field in the **CLink0-1Mode** register.

1.7.3 Starting and resetting links

The OS and DS data links on the IMS C100 start automatically in all modes following the reset signal going low. When the IMS C100 is set to mode 0 control link 0 also starts automatically following the reset signal going low. When the IMS C100 is set to modes 1, 2 or 3 control link 0 starts automatically following receipt of a token.

Following reset in all modes the OS-Links run at the speed determined by the **OSLink0HalfSpeed** and **OSLink123HalfSpeed** pins. Following reset, in mode 0 the DS-Links run at the speed determined by the **DSLInk0HalfSpeed** and **DSLInk123HalfSpeed** pins, in modes 1, 2 and 3 the DS-Links run at the base speed of 10 Mbits/s.

There are two basic mechanisms for resetting the data links. An error on the DS-Link (caused for example by resetting the DS-Link of an attached IMS T9000) causes the OS-Link of the pair and the internal state of the data conversion unit to be reset. The other mechanism is via the configuration bus. The configuration bus can be used to reset any individual DS-Link (via the **ResetLink** bit in the **DSLInk0-3Command** register) or any link pair (via the **Command** register).

1.8 Levels of reset

The IMS C100 can be reset to a given level using the *Reset* command or **Reset** pin. The *Reset* command is accompanied by a 'level' parameter. The IMS C100 directly supports levels 2 and 3 of the *Reset* command. The different levels of reset are described below.

Note that any level of reset may abort the command which was executing when the *Reset* command was applied. An illegal level of *Reset* will also result in a handshake with a failure status being returned.

1.8.1 Level 0 – hardware reset

In all modes the IMS C100 is reset by asserting the **Reset** pin high. The **ResetOut** pin follows the **Reset** pin. In mode 0 the IMS C100 is also reset by a similar transition on **TReset**, providing **AnalyseIn** is low.

After a hardware reset has been deasserted each IMS C100 is in the following state. All the links are in Wait state: with the data OS-Links operating at their default speed set by the **LinkHalfSpeed** pins; the data DS-Links of the IMS C100 in mode 0 operating at the speed set by the **DSLInkHalfSpeed** pins; the data DS-Links of the IMS C100 in modes 1, 2 and 3 operating at the base speed of 10 MHz; and the control links operating at their default speed of 10 MHz. The label and return headers for the control links are undefined. All registers contain their default values. All buffers are cleared; all latched error signals are cleared; and the **AnalyseOut** pin is low.

1.8.2 Level 1 – labelled control network

Reset to level 1 is not directly supported by the IMS C100. However, a reset to level 1 can be achieved by sending a *Reset2*, followed by *CPoke* commands to set the configuration of the DS data links to their default values and force their data and strobe outputs low.

This level of reset leaves the label and return headers unaltered and all connected control links remain operational. All the data links are in the Wait state. All registers are reset to their default values. All buffers are cleared.

The **ResetOut** pin is set high and must be de-asserted by a *CPoke* to the IMS C100 **Command** register with bit 5 set, see table 1.16.

1.8.3 Level 2 – configured network

The network can be reset to level 2 by sending a *Reset2* command message to each IMS C100.

At this level of reset the label and return headers are unaltered and register contents are unaffected. All buffers are cleared. The data links are reset and returned to the Wait state. The **booted** and **booting** flags are reset.

The **ResetOut** pin is set high and must be de-asserted by a *CPoke* to the IMS C100 **Command** register with bit 5 set, see table 1.16.

1.8.4 Level 3

The **ResetOut** pin is set high and must be de-asserted by a *CPoke* to the IMS C100 **Command** register with bit 5 set, see table 1.16. The **booted** and **booting** flags are reset.

1.8.5 Effects of different levels of reset

The effect of different levels of reset on various aspects of the IMS C100 state is summarized in the following table.

The handshake state indicates whether the IMS C100 expects a handshake message; the acknowledge state indicates whether it expects to receive an acknowledge packet; and the error state is the latched error signals which would otherwise cause *Error* messages to be sent. When the handshake/acknowledge state is cleared any outstanding handshakes/acknowledges will be ignored.

The corresponding effects of the *RecoverError* message are also shown. The *RecoverError* command resets the acknowledge state so that acknowledges are neither expected nor pending, and causes the re-transmission of any unhandshaken error message.

State	Reset level		Recover Error
	2	3	
OS-Links 0-3	Reset	no effect	no effect
DS-Links 0-3	Reset	no effect	no effect
Control links 0-1	Reset	no effect	no effect
Handshake state	Cleared	Cleared	Cleared
Acknowledge state	no effect	no effect	Cleared
Error state	Cleared	no effect	no effect
AnalyseOut pin	De-asserted	no effect	no effect
ResetOut pin	Asserted	Asserted	no effect

Table 1.12 Effect of the different levels of reset

1.9 Configuration

1.9.1 Configuration space

The IMS C100 can be controlled via the configuration address space. The registers in this address space are accessed by *CPeek* and *CPoke* command messages received along **CLink0**.

1.9.2 Configuration register addresses

Table 1.13 gives the addresses of the configuration registers. The **Read/Write** column indicates whether the register is read-only (R), write-only (W), or read-write (R/W). All registers are 32 bits long, and 32 bits are always read or written. The column labelled **Bit size** indicates the number of valid bits in the register. The lowest bits in the register are significant. This is shown in the **Significant bits** column. When writing to a register, any non-significant bits must always be zero. When reading from a register, any non-significant bits are undefined. For registers listed as containing 0 bits, any write to the corresponding address causes some action to occur.

Address	Register	Bit size	Significant bits	Read/Write	Reset value
#1001	DeviceID	16	0-15	R	
#1002	DeviceRevision	16	0-15	R	
#1003	Command	7	0-6	W	see table 1.16
#1004	ErrorCode	8	0-7	R	see table 1.8
#1005	DSLinkPLL	6	0-5	R/W	10
#1006	Status	8	0-7	R	see table 1.19
#8001	DSLink0Mode	5	0-4	R/W	
#8002	DSLink0Command	4	0-3	W	
#8003	DSLink0Status	6	0-5	R	see Reset section 1.8
#8101	DSLink1Mode	5	0-4	R/W	
#8102	DSLink1Command	4	0-3	W	
#8103	DSLink1Status	6	0-5	R	see Reset section 1.8
#8201	DSLink2Mode	5	0-4	R/W	
#8202	DSLink2Command	4	0-3	W	
#8203	DSLink2Status	6	0-5	R	see Reset section 1.8
#8301	DSLink3Mode	5	0-4	R/W	
#8302	DSLink3Command	4	0-3	W	
#8303	DSLink3Status	6	0-5	R	see Reset section 1.8
#FD01	CLink0Mode	5	0-4	R/W	
#FD02	CLink0Command	4	0-3	W	
#FD03	CLink0Status	6	0-5	R	see Reset section 1.8
#FE01	CLink1Mode	5	0-4	R/W	
#FE02	CLink1Command	4	0-3	W	
#FE03	CLink1Status	6	0-5	R	Depends on mode
#8004, #8104, #8204, #8304, #FD04, #FE04,	DSLink0WriteLock†, DSLink1WriteLock†, DSLink2WriteLock†, DSLink3WriteLock†, CLink0WriteLock†, CLink1WriteLock†	1	0	R/W	
#FF01‡,	DSLink0-3Mode,	5	0-4	R/W	
#FF02‡,	DSLink0-3Command,	4	0-3	W	
#FF04‡	DSLink0-3WriteLock	1	0	R/W	

† This register is not used on the IMS C100.

‡ The subsystem set of all four data DS-Links has a unique address #FF. This address can be used when writing (poking) to the associated register for each of the four DS-Links.

Table 1.13 Configuration register addresses

1.9.3 Configuration registers

This section describes the functionality of the IMS C100 to be controlled by the configuration registers and the associated bit fields.

The configuration bus can be used to reset any individual DS-Link (via the **DSLInk0-3Command** register) or any link pair (via the **Command** register) in packetized conversion mode, modes 1 and 3.

Note: All undefined and INMOS reserved bits must be written with 0 unless otherwise stated.

System services configuration registers

The **DeviceID** register contains a 16 bit device identification code unique to the device. The value of the device identification code for the IMS C100 is in the range 320 to 335. The device identification code can also be read using the *Identify* command. This register is read only.

DeviceID		#1001	Read only
Bit	Bit field	Function	
15:0	DeviceID	Device identification code.	
31:16		Undefined	

Table 1.14 Bit fields in the **DeviceID** register

The **DeviceRevision** register contains the revision of the device. It is a 16 bit read-only register.

DeviceRevision		#1002	Read only
Bit	Bit field	Function	
15:0	DeviceRev	Device revision.	
31:16		Undefined	

Table 1.15 Bit fields in the **DeviceRevision** register

A bit set in the **Command** register effects the indicated function. The IMS C100 **Command** register, which is write-only, has the structure shown in table 1.16 below.

Command		#1003	Write only
Bit	Function	Bit description	
0	Link0 pair reset	Set to 1 to cause reset of OSLink0 , DSLInk0 and conversion unit	
1	Link1 pair reset	Set to 1 to cause reset of OSLink1 , DSLInk1 and conversion unit	
2	Link2 pair reset	Set to 1 to cause reset of OSLink2 , DSLInk2 and conversion unit	
3	Link3 pair reset	Set to 1 to cause reset of OSLink3 , DSLInk3 and conversion unit	
5	End Reset	Set to 1 to de-assert the ResetOut pin	
6	End Analyse	Set to 1 to de-assert the AnalyseOut pin	
4, 31:7		Undefined	

Table 1.16 Bit fields in the **Command** register

The **ErrorCode** register is an 8 bit register used for debugging after a crash. This register is read only.

ErrorCode		#1004	Read only
Bit	Bit field	Function	
7:0	ErrorCode	Contains an error code which can be used for debugging after a crash. Refer to table 1.8, page 278 for the error code definitions.	
31:8		Undefined	

Table 1.17 Bit fields in the **ErrorCode** register

The **DSLlinkPLL** register contains the **SpeedMultiply** bit field and is used to program the DS-Link speeds. This takes the 10 MHz clock and multiplies it by a programmable value to provide the root clock for all the DS-Links. Refer to section 1.7.1 for further details.

DSLlinkPLL		#1005	Read/Write
Bit	Bit field	Function	
5:0	SpeedMultiply	Sets DS-Link master clock to required value (see table 13.3).	
31:6		Undefined	

Table 1.18 Bit fields in the **DSLlinkPLL** register

A bit set in the **Status** register indicates the current status (high or low) of a pin or flag. The status register is read only.

Status		#1006	Read only
Bit	Status of pin/flag	Bit description	
0	Mode0	Set if the Mode0 pin has been asserted.	
1	Mode1	Set if the Mode1 pin has been asserted.	
2	OSLink0HalfSpeed	Set if the OSLink0HalfSpeed pin has been asserted.	
3	OSLink123HalfSpeed	Set if the OSLink123HalfSpeed pin has been asserted.	
4	DSLlink0HalfSpeed	Set if the DSLlink0HalfSpeed pin has been asserted.	
5	DSLlink123HalfSpeed	Set if the DSLlink123HalfSpeed pin has been asserted.	
7	Booting	Set by the <i>Boot16</i> , <i>Boot32</i> commands. This flag is reset by the <i>Reset</i> command or by the last <i>BootData</i> command.	
8	Booted	Set by the last <i>BootData</i> command or the <i>ReBoot</i> command. This flag is reset by the <i>Reset</i> or <i>Stop</i> commands.	
6, 31:9		Undefined	

Table 1.19 Bit fields in the **Status** register

Data DS-Link configuration registers

Each DS-Link has three registers, the **DSLlinkMode** register, **DSLlinkCommand** register and **DSLlink-Status** register.

In addition the configuration space contains the **DSLlinkPLL** register which contains the **SpeedMultiply** bit, see table 1.18.

The **DSLlink0-3Mode** registers power up into a default state and may be reprogrammed before or after the link has been started.

DSLlinkMode		#8001, #8101, #8201, #8301	Read/Write
Bit	Bit field	Function	
1:0	SpeedDivide	Sets transmit speed of the DS-Link (see table 1.10). 00 = /1, 01 = /2, 10 = /4, 11 = /8	
2	SpeedSelect	Sets the DS-Link to transmit at the speed determined by the Speed-Divide bits as opposed to the base speed of 10 Mbits/s.	
3	LocalizeError	When set errors are no longer reported to the control link. Packets in transit at the time of an error will be discarded or truncated.	
4	1 (RESERVED)	This bit should be written as 1.	
31:5		Undefined	

Table 1.20 Bit fields in the **DSLlink0-3Mode** registers

The **DSLInk0-3Command** registers are write only and contain four bits which when set cause a specific action to be taken by the DS-Link.

DSLInkCommand		#8002, #8102, #8202, #8302	Write only
Bit	Bit field	Function	
0	ResetLink	Resets the link engine of the DS-Link. The token state is reset, the flow control credit is set to zero, the buffers are marked as empty, and the parity state is reset.	
1	StartLink	When a transition from 0 to 1 occurs the DS-Link will be initialized and commence operation.	
2	ResetOutput	Sets both outputs of the DS-Link low.	
3	WrongParity	The DS-Link output will generate incorrect parity. This may be used to force a parity error on a transputer at the other end of the DS-Link.	
31:4		Undefined	

Table 1.21 Bit fields in the **DSLInk0-3Command** registers

The **DSLInk0-3Status** registers are read only and contain six bits which contain information about the state of the DS-Link.

DSLInkStatus		#8003, #8103, #8203, #8303	Read only
Bit	Bit field	Function	
0	LinkError	Flags that an error has occurred on the DS-Link.	
1	LinkStarted	Flags that the output DS-Link has been started and no errors have been detected.	
2	ResetOutputComplete	Flags that ResetOutput has completed on the DS-Link.	
3	ParityError	Flags that a parity error has occurred on the DS-Link.	
4	DiscError	Flags that a disconnect error has occurred on the DS-Link.	
5	TokenReceived	Flags that a token has been seen on the DS-Link since ResetLink .	
31:6		Undefined	

Table 1.22 Bit fields in the **DSLInk0-3Status** registers

All data links

The subsystem set of all four data DS-Links has a unique address #FF. This address referring to the set of DS-Links should be used when writing (poking) to the associated register for each of the four DS-Links. For example, to simultaneously write to all four DS-Link command registers (**DSLInk0-3Command**), address #FF02 should be used.

Control link configuration registers

The link module hardware in each control link is identical to that in each data DS-Link. An equivalent set of configuration bit fields is provided for each control link, as for the data DS-Links.

Write lock registers

This register is not used on the IMS C100.

CPUWriteLock		#8004, #8104, #8204, #8304, #8FD4, #8FE4,	Read/Write
Bit	Bit field	Function	
0	WriteLock	Not used on the IMS C100.	

Table 1.23 Bit fields in the **CPUWriteLock** register

1.10 Electrical specifications

Inputs and outputs are TTL compatible.

1.10.1 Absolute maximum ratings

Symbol	Parameter	Min	Max	Units	Notes
VDD	DC supply voltage	0	7.0	V	1,2,3,4,5
V _i , V _o	Voltage on input and output pins	-0.5	VDD+0.5	V	1,3,4,5
I _i	Input current		±25	mA	6
t _{osc}	Output short circuit time (one pin)		1	s	4
T _s	Storage temperature	-65	150	°C	4

Table 1.24 Absolute maximum ratings

Notes

- All voltages are with respect to **GND**.
- Power is supplied to the device via the **VDD** and **GND** pins. Several of each are provided to minimize inductance within the package. All supply pins must be connected. The supply must be decoupled close to the chip by at least one 100 nF low inductance (e.g. ceramic) capacitor between **VDD** and **GND**. Four layer boards are recommended; if two layer boards are used, extra care should be taken in decoupling.
- Input voltages must not exceed specification with respect to **VDD** and **GND**, even during power-up and power-down ramping, otherwise *latchup* can occur. CMOS devices can be permanently damaged by excessive periods of latchup.
- This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operating sections of this specification is not implied. Stresses greater than those listed may cause permanent damage to the device. Exposure to absolute maximum rating conditions for extended periods may affect reliability.
- This device contains circuitry to protect the inputs against damage caused by high static voltages or electrical fields. However, it is advised that normal precautions be taken to avoid application of any voltage higher than the absolute maximum rated voltages to this high impedance circuit. Unused inputs should be tied to an appropriate logic level such as **VDD** or **GND**.
- The input current applies to any input or output pin and applies when the voltage on the pin is between **GND** and **VDD**.

1.10.2 Operating conditions

Symbol	Parameter	Min	Max	Units	Notes
VDD	DC supply voltage	4.75	5.25	V	1
V _i , V _o	Input or output voltage	0	VDD	V	1,2

Table 1.25 Operating conditions

Notes

- All voltages are with respect to **GND**.
- Excursions beyond the supplies are permitted but not recommended.

1.11 Recommended decoupling

1.11.1 Power decoupling

Power is supplied to the device via the **VDD** and **GND** pins. Several of each are provided to minimize inductance within the package. All supply pins must be connected. The supply must be decoupled close to the chip by at least one 0.1 μF ceramic capacitor between **VDD** and **GND**.

1.11.2 Phase locked loop decoupling

The internally derived power supply for internal clocks requires an external low leakage, low inductance 2 μF capacitor to be connected between **CapPlus** and **CapMinus**. A surface mounted ceramic capacitor should be used. In order to keep stray inductances low, the total PCB track length should be less than 20 mm, thus the capacitor should be no more than 10 mm from the chip. The connections must not touch power supplies or other noise sources.

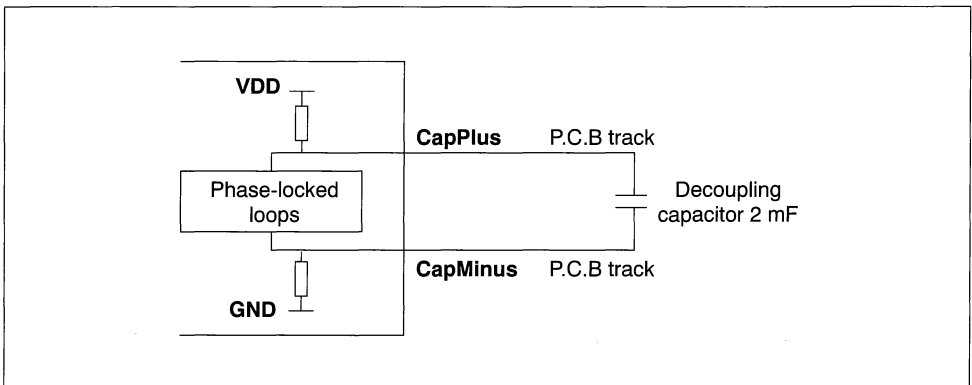


Figure 1.29 Recommended PLL decoupling

1.12 Clocks

1.12.1 Clock input

The high frequency internal clocks are derived from the clock frequency supplied by the user. The user supplies the clock frequency for input to the PLL's via the **ClockIn** input. The nominal frequency of this clock for all transputer family components is 5 MHz, regardless of device type, transputer word length or processor cycle time.

A number of transputer devices may be connected to a common clock, or may have individual clocks providing each one meets the specified stability criteria. In a multi-clock system the relative phasing of **ClockIn** clocks is not important, due to the asynchronous nature of the links. Mark/space ratio is unimportant provided the specified limits of **ClockIn** pulse widths are met.

Oscillator stability is important. **ClockIn** must be derived from a crystal oscillator; RC oscillators are not sufficiently stable. **ClockIn** must not be distributed through a long chain of buffers. Clock edges must be monotonic and remain within the specified voltage and time limits.

The timing requirements for **ClockIn** are given in section 1.13.2 of the Timing Specifications section.

1.13 Timing specifications

1.13.1 Reset and Analyse timings

ResetOut and AnalyseOut timings

Symbol	Parameter	Min	Nom	Max	Units	Notes
tROHROL	ResetOut pulse width high	8			ClockIn	
tAOHROH	AnalyseOut setup before ResetOut	3			ms	
tROLAOL	AnalyseOut hold after ResetOut end	1			ClockIn	

Table 1.26 **ResetOut** and **AnalyseOut** timings

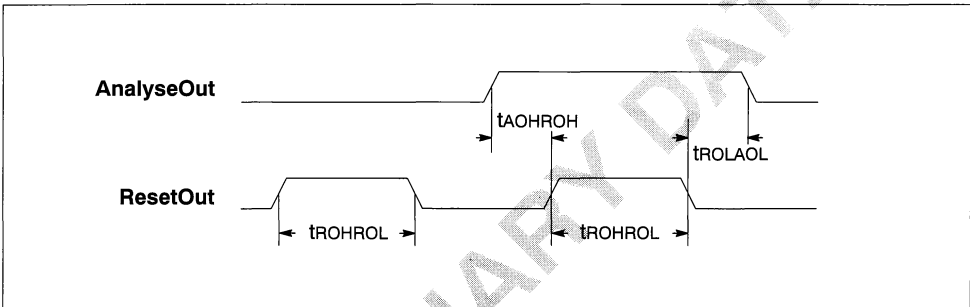


Figure 1.30 **ResetOut** and **AnalyseOut** timing

TReset and AnalyseIn timings

Symbol	Parameter	Min	Nom	Max	Units	Notes
tTRHTRL	TReset pulse width high	8			ClockIn	
tAIHTRH	AnalyseIn setup before TReset	3			ms	
tTRLAIL	AnalyseIn hold after TReset end	1			ClockIn	

Table 1.27 **TReset** and **AnalyseIn** timings

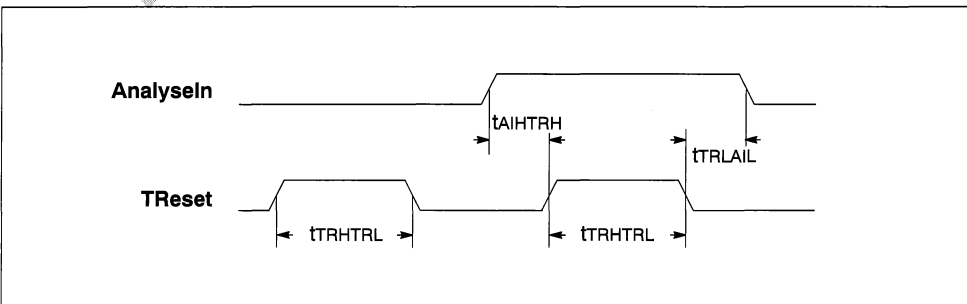


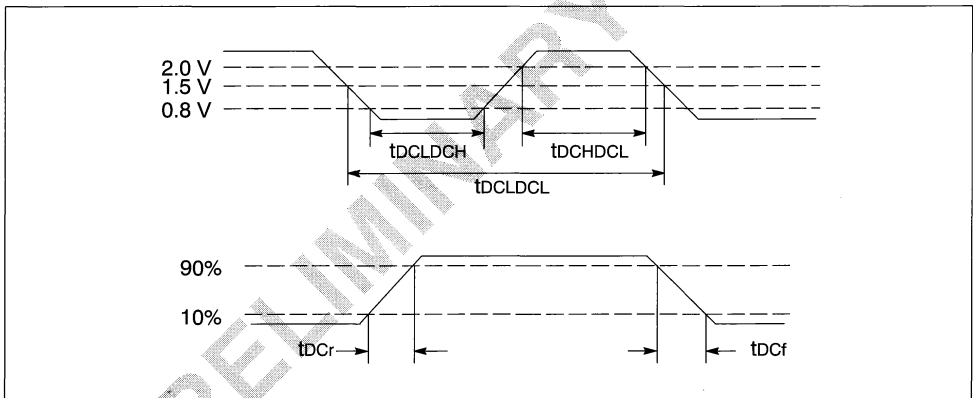
Figure 1.31 **TReset** and **AnalyseIn** timing

1.13.2 ClockIn timings

Symbol	Parameter	Min	Nom	Max	Units	Notes
tDCLDCH	ClockIn pulse width low	40			ns	
tDCHDCL	ClockIn pulse width high	40			ns	
tDCLDCL	ClockIn period		200		ns	1, 2
tDCr	ClockIn rise time			10	ns	3
tDCf	ClockIn fall time			8	ns	3

Table 1.28 **ClockIn** timings**Notes**

- 1 Measured between corresponding points on consecutive falling edges.
- 2 This value allows the use of 200 ppm crystal oscillators for two devices connected together by a link.
- 3 Clock transitions must be monotonic within the range V_{IH} to V_{IL} (refer to Electrical specifications section 1.10).

Figure 1.32 **ClockIn** timing

1.13.3 DS-Link timings

Symbol	Parameter	Min	Nom	Max	Units
tLODSr	DSLinkOut rise time		4		
tLODSf	DSLinkOut fall time		4		
tLIDSr	DSLinkIn rise time		4		
tLIDSf	DSLinkIn fall time		4		
tLIHL	Input edge resolution	2			ns
tSDS	Bit period	10		100	ns
Δ tDSO	Data / strobe output skew			1	ns
CLIZ	DSLinkIn capacitance		7		pF

Table 1.29 DS-Link timings

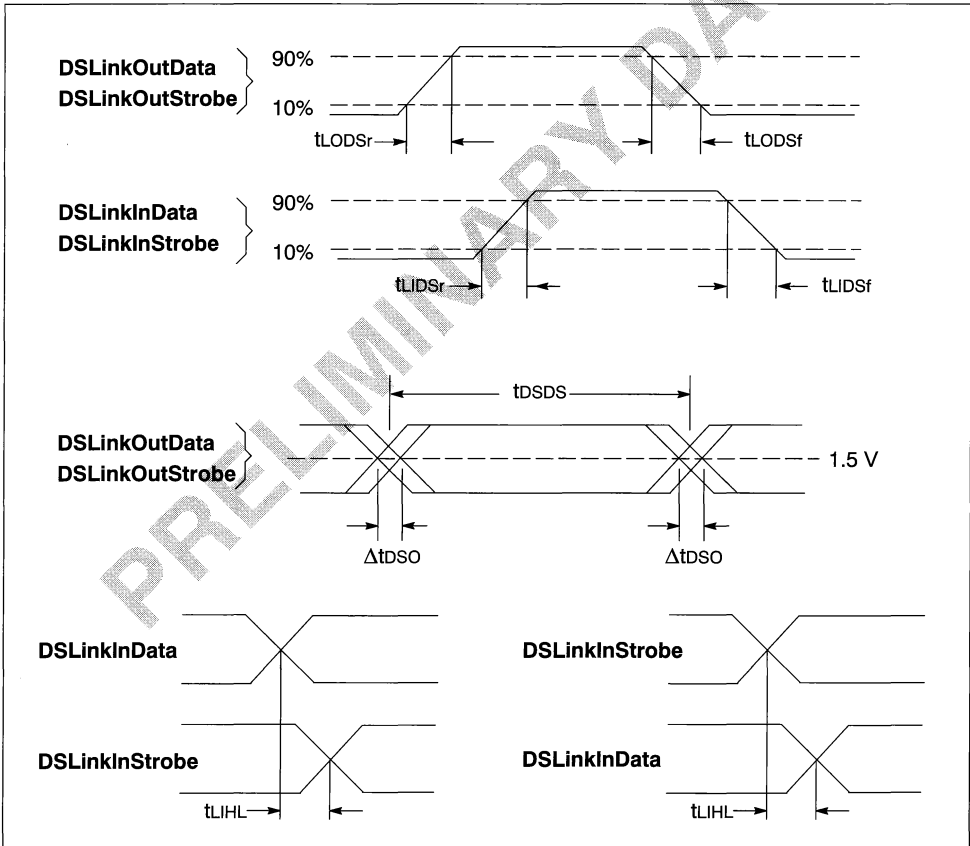


Figure 1.33 DS-Link timing

1.13.4 OS-Link timings

Symbol	Parameter	Min	Nom	Max	Units	Notes
tJQr	OSLinkOut rise time			20	ns	1
tJQf	OSLinkOut fall time			10	ns	1
tJDr	OSLinkIn rise time			20	ns	1
tJdf	OSLinkIn fall time			20	ns	1
tJQJD	Buffered edge delay	0			ns	
ΔtJB	Variation in tJQJD	20 Mbits/s		3	ns	2
		10 Mbits/s		10	ns	2
		5 Mbits/s		30	ns	2
CLIZ	OSLinkIn capacitance @ f=1MHz			7	pF	1
CLL	OSLinkOut load capacitance			50	pF	

Table 1.30 OS-Link timings

Notes

- 1 Guaranteed, but not tested.
- 2 This is the variation in the total delay through buffers, transmission lines, differential receivers etc., caused by such things as short term variation in supply voltages and differences in delays for rising and falling edges.

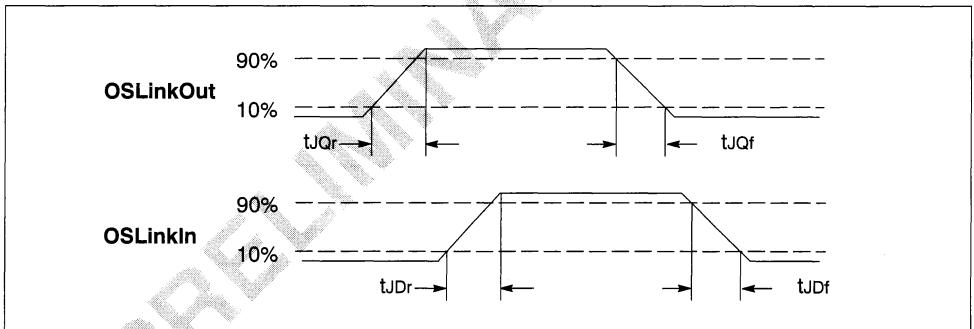


Figure 1.34 OS-Link timing

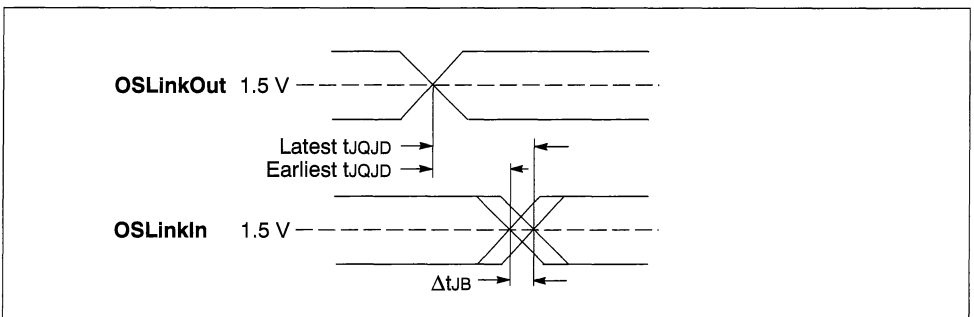


Figure 1.35 Buffered OS-Link timing

1.14 Pin designations

The following tables outline the function of each of the pins. Pinout details are given in section 1.15.

Signal names are prefixed by **not** if they are active low, otherwise they are active high.

Supplies

Pin	In/Out	Signal description
VDD		Power supply
GND		Ground

Table 1.31 IMS C100 supplies

Clocks

Pin	In/Out	Signal description
CapPlus, CapMinus		External capacitor for internal clock power supply
ClockIn	in	5 MHz input clock

Table 1.32 IMS C100 clocks

Links

Pin	In/Out	Signal description
OSLinkIn0-3	in	OS-Link input data channels
OSLinkOut0-3	out	OS-Link output data channels
DSLinkInData0-3	in	DS-Link input data channels
DSLinkInStrobe0-3	in	DS-Link input strobes
DSLinkOutData0-3	out	DS-Link output data channels
DSLinkOutStrobe0-3	out	DS-Link output strobes
CLinkInData0-1	in	Control link input data channels
CLinkInStrobe0-1	in	Control link input strobes
CLinkOutData0-1	out	Control link output data channels
CLinkOutStrobe0-1	out	Control link output strobes
OSLink0HalfSpeed	in	OS-Link 0 speed selection
OSLink123HalfSpeed	in	OS-Link 1, 2, 3 speed selection
DSLink0HalfSpeed	in	DS-Link 0 speed selection
DSLink123HalfSpeed	in	DS-Link 1, 2, 3 speed selection

Table 1.33 IMS C100 links

Control unit

Pin	In/Out	Signal description
Reset	in	System reset
ResetOut	out	Asserts the Reset pin on any connected T9-series or T2/T4/T8-series device.
TReset	in	Mode 0 T2/T4/T8-series transputer reset. In modes 1–3 this pin must be tied to GND
Error	in	Modes 1–3 error indicator – message sent from CLink0 . In mode 0 this pin must be tied to GND
AnalyseIn	in	Mode 0 error analysis In modes 1–3 this pin must be tied to GND
AnalyseOut	out	Mode 1–3 error analysis – a <i>Stop</i> message received on CLink0 causes this pin to be set high. In mode 0 it is low.
Mode0-1	in	Mode of operation

Table 1.34 IMS C100 control unit

JTAG support

The IMS C100 supports the IEEE 1149.1 test standard which has been agreed by the Joint Test Action Group (JTAG). There are five pins which support the JTAG standard.

Pin	In/Out	Signal description
TDI	in	Test data input
TDO	out	Test data output
TMS	in	Test mode select
TCK	in	Test clock
notTRST	in	Test logic reset

Table 1.35 IMS C100 JTAG support pins

Miscellaneous

Pin	In/Out	Signal description
HoldToGND		Must be connected to GND
DoNotWire		Must not be wired

Table 1.36 IMS C100 miscellaneous pins

1.15 Package specifications

The IMS C100 is available in a 100 pin ceramic quad flatpack (CQFP) package and a 100 pin plastic quad flatpack (PQFP) package. It is intended for cavity up assembly and the dimensions detailed below apply to cavity up installation.

1.15.1 IMS C100 100 pin cavity-up CQFP package pinout

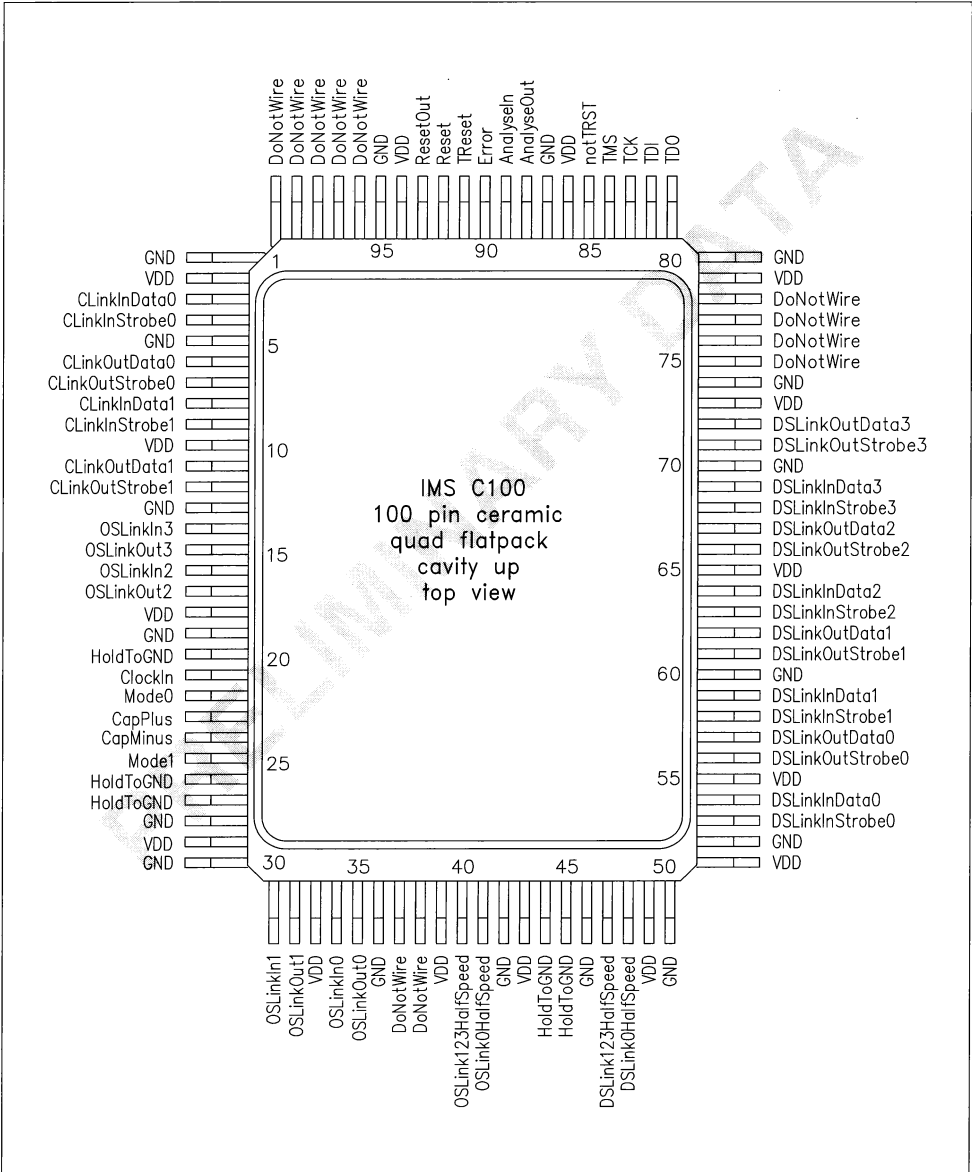
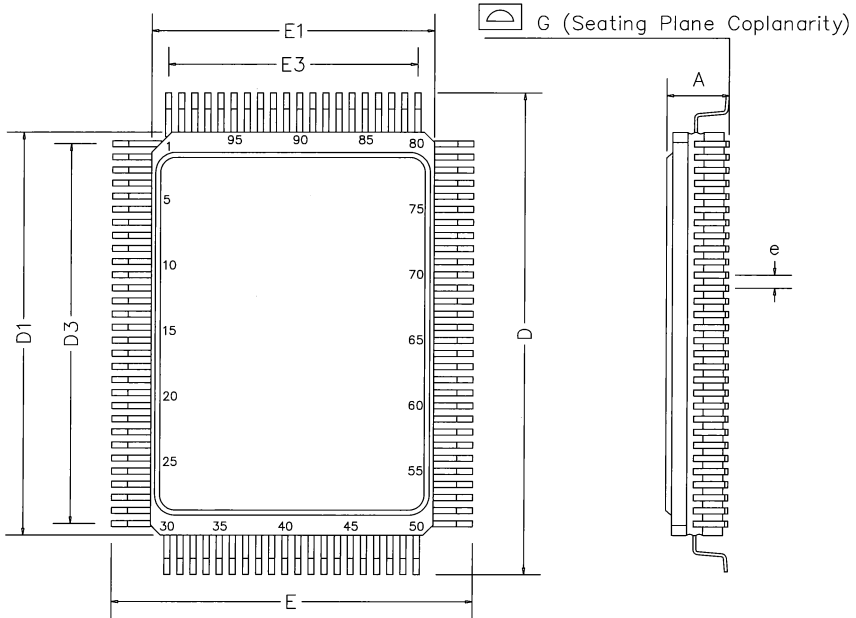
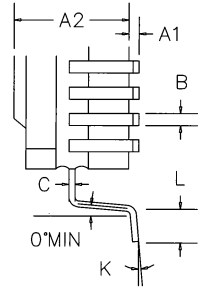


Figure 1.36 IMS C100 100 pin cavity-up CQFP package pinout

1.15.2 100 pin CQFP package dimensions

DIM	CONTROL DIMENSIONS mm			ALTERNATIVE DIMENSIONS INCH		
	MIN	NOM	MAX	MIN	NOM	MAX
A	—	—	3.400	—	—	0.134
A1	0.250	—	—	0.010	—	—
A2	—	—	3.073	—	—	0.121
B	0.220	—	0.380	0.009	—	0.15
C	0.130	—	0.230	0.005	—	0.009
D	23.650	—	24.150	0.931	—	0.951
D1	19.800	20.000	20.200	0.780	0.787	0.795
D3	—	18.850REF	—	—	0.742REF	—
E	17.650	—	18.150	0.695	—	0.715
E1	13.840	14.000	14.150	0.545	0.551	0.557
E3	—	12.350REF	—	—	0.486REF	—
e	—	0.650BSC	—	—	0.026BSC	—
G	—	—	0.100	—	—	0.004
K	0*	—	7*	0*	—	7*
L	0.650	0.800	0.950	0.026	0.031	0.037



Notes;

1. Maximum lead displacement from notional centre line = $\pm 0.125\text{mm}$

Figure 1.37 100 pin CQFP package dimensions

1.15.3 IMS C100 100 pin cavity-up PQFP package pinout

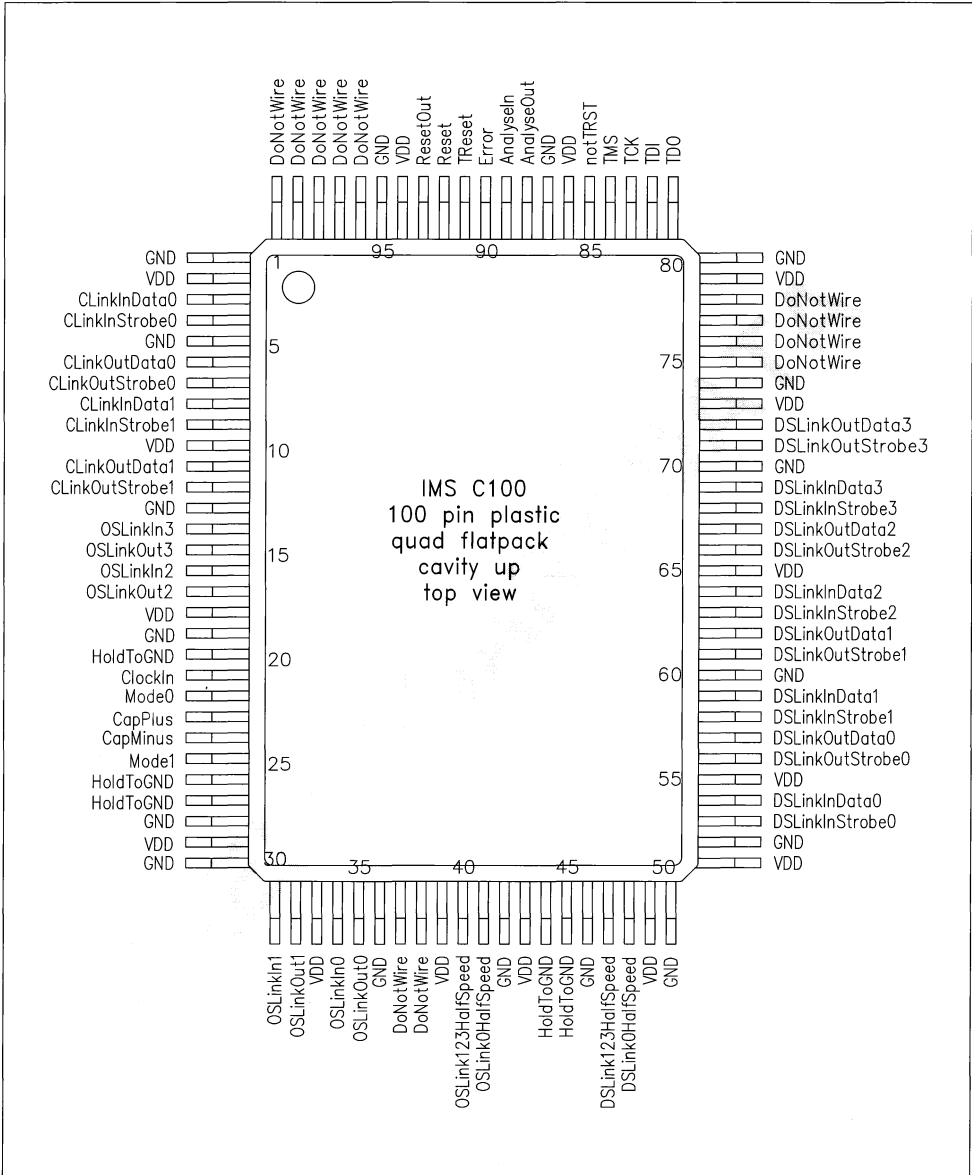
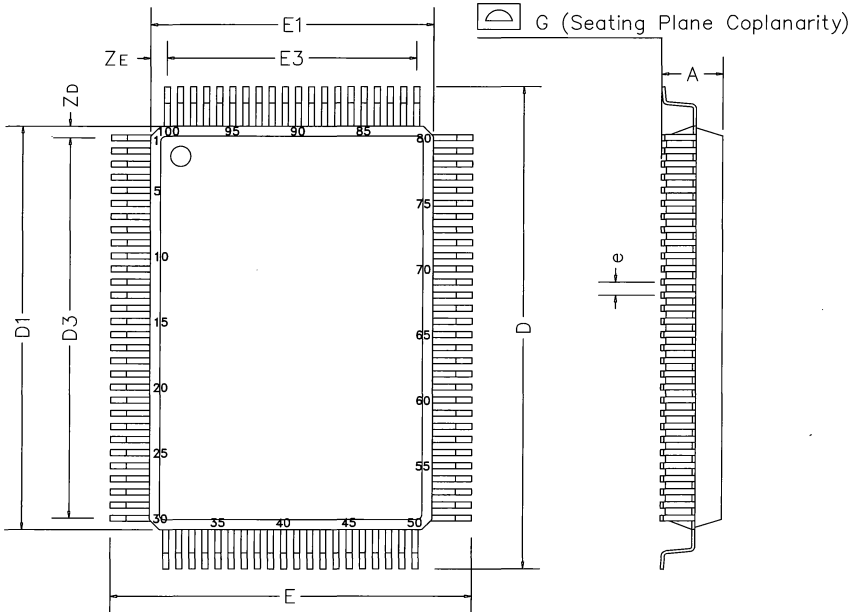
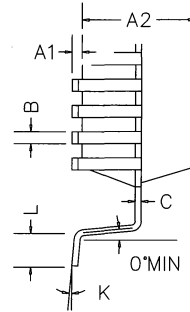


Figure 1.38 IMS C100 100 pin cavity-up PQFP package pinout

1.15.4 100 pin PQFP package dimensions

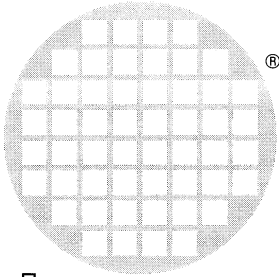
DIM	CONTROL DIMENSIONS mm			ALTERNATIVE DIMENSIONS INCH		
	MIN	NOM	MAX	MIN	NOM	MAX
A	—	—	3.400	—	—	0.134
A1	0.100	—	0.360	0.004	—	0.014
A2	2.450	2.800	3.050	0.096	0.110	0.120
B	0.220	—	0.380	0.009	—	0.15
C	0.130	—	0.230	0.005	—	0.009
D	22.950	—	24.150	0.904	—	0.951
D1	19.900	20.000	20.100	0.783	0.787	0.791
D3	—	18.850REF	—	—	0.742REF	—
E	16.950	—	18.150	0.667	—	0.715
E1	13.900	14.000	14.100	0.547	0.551	0.555
E3	—	12.350REF	—	—	0.486REF	—
e	—	0.650BSC	—	—	0.026BSC	—
G	—	—	0.100	—	—	0.004
K	0°	—	7°	0°	—	7°
L	0.650	0.800	0.950	0.026	0.031	0.037
Zp	—	0.580REF	—	—	0.023REF	—
Ze	—	0.830REF	—	—	0.033REF	—



Notes;

1. Maximum lead displacement from notional centre line = ±0.125mm.

Figure 1.39 100 pin PQFP package dimensions



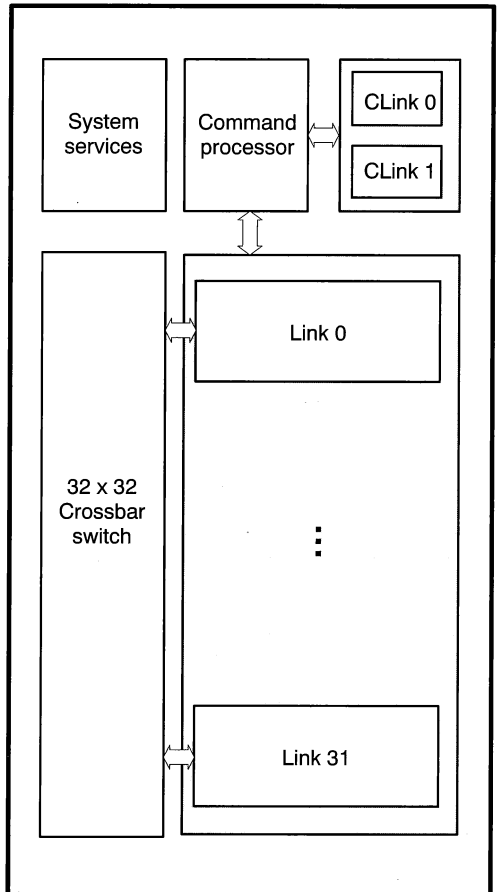
inmos®

IMS C104 packet routing switch

Product preview

FEATURES

- 32 way programmable packet router
- 100 Mbits/s serial bi-directional links
- 640 Mbytes/s bandwidth
- Concurrent processing of packets
- High rate of packet processing
 - up to 200 M packets/s
- Less than 1 μ second packet latency
- Non-blocking crossbar
- Separate control system
- Wormhole interval routing algorithm
- Cascadable to any depth
- No loss of signal integrity
- Partitioning
- Grouped adaptive routing



This is preliminary information on a product under development and product details may change.

2.1 IMS C104 introduction

The IMS C104 is a complete, low latency, packet routing switch on a single chip. It connects 32 high bandwidth serial communication links to each other via a 32 by 32 way non-blocking crossbar switch, enabling messages to be routed from any of its links to any other link. The links operate concurrently and the transfer of a packet between one pair of links does not affect the data rate for another packet passing between a second pair of links. Each link can operate at up to 100 Mbits/s, providing a bidirectional bandwidth of 20 Mbytes/s, with the IMS C104 supporting a rate of packet processing of up to 200 Mpackets/s.

The IMS C104 allows communication between IMS T9000 transputers that are not directly connected. A single IMS C104 can be used to connect up to 32 IMS T9000 transputers. The IMS C104 can also be connected to other IMS C104s to make larger and more complex switching networks, linking any number of IMS T9000 transputers, link adaptors, and any other devices that use the link protocol. Another member of the IMS T9000 product family, the IMS C101 flexible link adaptor, will allow links to be interfaced to peripheral buses and devices.

The IMS C104 enables networks to be built which effectively emulate a direct connection between each of the devices in the system. In the absence of any contention for a link output, the packet latency will be less than 1 μ second.

A message on a IMS C104 communication system is transmitted as a sequence of packets. To ensure that packets which are parts of different messages can be routed, each packet contains a header. The IMS C104 uses the header of each packet arriving to determine the link to be used to output the packet. Anything after the header is treated as the packet body until the packet terminator is received. This enables the IMS C104 to transmit packets of arbitrary length.

In most packet switching networks complete packets are stored internally, decoded, and then routed to the destination node. This causes relatively long delays due to high latency at each node. To overcome this limitation, the IMS C104 uses *wormhole routing*, in which the routing decision is taken as soon as the routing information, which is contained in the packet header, has been input. Therefore the packet header can be received, and the routing decision taken, before the whole packet has been transmitted by the source. A packet may be passing through several nodes at any one time. Thus, latency is minimized and transmission can be continuous.

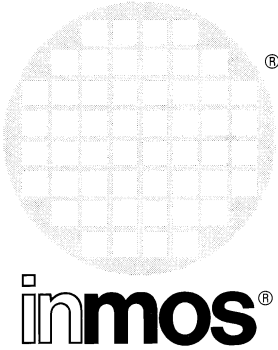
The term *wormhole routing* comes from the analogy of a worm crawling through soil, creating a hole that closes again behind its tail. Wormhole routing is invisible as far as the senders and receivers of packets are concerned, its only effect is to minimize the latency in message transmission.

The routing algorithm which makes the routing decision is called *interval labeling*, which is complete, deadlock free, inexpensive and fast. Each destination in a network is labeled with a number, and this number is used as the destination address in a packet header. Each link in a routing switch is labeled with an interval of possible header values, and only packets whose header value falls within that interval are output via that link.

The IMS C104 contains a hardware mechanism to allow independently programmed networks to be connected together. It also has additional circuitry to reduce the impact of message congestion on worst-case latency and bandwidth, in heavily loaded networks.

The IMS C104 is controlled and programmed via a control link. The IMS C104 has two separate control links, one for receiving commands and one to provide daisy chaining. The control links enable networks of IMS T9000 transputers and IMS C104s to be controlled and monitored for errors. The control links can be connected into a daisy chain or tree, with a controlling processor, such as an IMS T9000, at the root.

A set of tools is available to support the configuration of IMS T9000 systems. The tools provide support in the configuration and initialization of networks consisting of IMS T9000 processors and IMS C104 routing switches. Refer to *The T9000 Development Tools Preliminary Datasheets (document number 72-TRN-249-00)* for further details.



IMS C101 parallel DS-Link adaptor

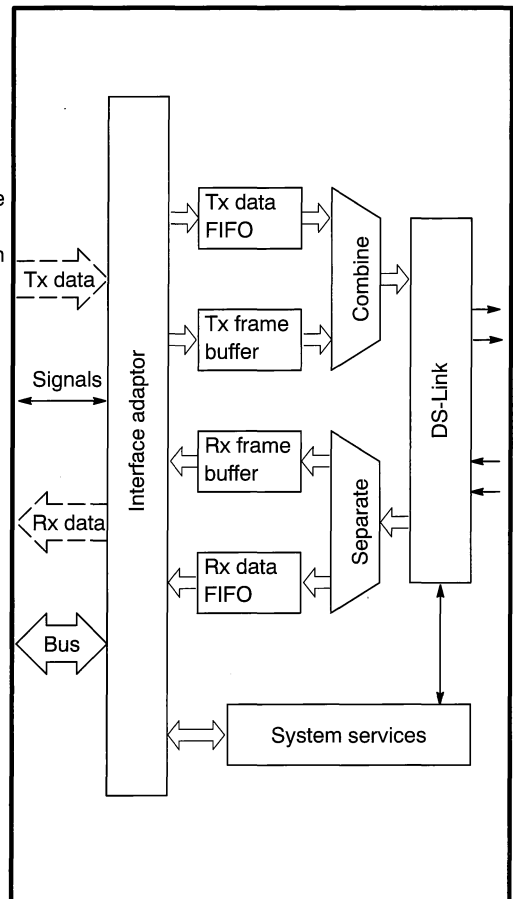
Product preview

FEATURES

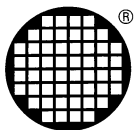
- Standard DS-Link protocol
- Converts between parallel bus and DS-Links
- Optional FIFO interfaces
- Interfaces to packet routing networks
- Performs DS-Link packetization
- Variable packet length capability
- Packetization function can be disabled to provide simple point to point connection
- Provides point-to-point bi-directional handshaken FIFO function
- Supports T9000 virtual links and virtual routing
- 64 byte Tx and Rx FIFOs optimize packet processing performance
- Programmable parallel bus interface (8, 16 or 32 bit)
- Optional bus parity checking
- 100 Mbits/s unidirectional bandwidth
- Interrupt capability
- Independent clock systems
- 100 pin quad flat pack package
- Single +5V \pm 5% power supply

APPLICATIONS

- Connecting microprocessors/peripherals to T9000 transputers
- Connecting microprocessors/peripherals to C1xx communications family devices
- High speed link between microprocessors
- Inter-family microprocessor interfacing
- Allow ATM, Fibrechannel and other networking nodes to take advantage of INMOS' virtual routing technology



This is preliminary information on a product under development and product details may change.



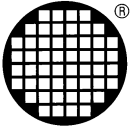
IMS T9000 special values

A1 IMS T9000 special values

A number of values are used by the IMS T9000 transputer to indicate the state of a process and other special conditions. These are shown in table A1.1.

Name	Value	Meaning
<i>DeviceId</i>	Depends on transputer type.	A value used to identify the type and revision of transputer. Each product is allocated a unique group of numbers. The value of the device id for the IMS T9000 is in the range 300 to 319. The device id is returned by the <i>IddevId</i> instruction.
<i>NotProcess.p</i>	#80000000	Used, wherever a process descriptor is expected, to indicate that there is no process.
<i>NoneSelected.o</i>	#FFFFFFF	Stored in the pw.Temp slot of a process' workspace while no branch of an alternative has been selected during the waiting and disabling phases.
<i>Disabling.p</i>	#80000003	Stored in the pw.State location while an alternative is being disabled.
<i>Enabling.p</i>	#80000001	Stored in the pw.State location while an alternative is being enabled.
<i>Ready.p</i>	#80000003	Stored in the pw.State location during the enabling phase of an alternative, to indicate that a guard is ready.
<i>Waiting.p</i>	#80000002	Stored in the pw.State location by <i>altwt</i> (alt wait) and <i>taltwt</i> (timer alt wait) to indicate that the alternative is waiting.
<i>LengthError.p</i>	#FFFFFFF	Stored in the pw.Length slot of a process' workspace to indicate that the number of bytes received by a <i>vin</i> (variable input) instruction was more than the maximum specified.
<i>NullHeader</i>	#FFFFFFF	Used where a null virtual channel header is needed.
<i>NullOffset</i>	#FFFFFFF	Used where a null virtual channel header offset is needed.
<i>Deactivated.p</i>	#80000001	Stored in a channel word to indicate that the channel is deactivated.
<i>ResChan.p</i>	#80000002	Stored in a channel word to indicate that it is in resource channel mode.
<i>Stopping.p</i>	#80000003	Stored in a channel word to indicate that the channel is stopping.
<i>TimeNotSet.p</i>	#80000002	Stored in pw.TLink location during enabling of a timer alternative after a time to wait for has been encountered.
<i>TimeSet.p</i>	#80000001	Stored in pw.TLink location when a timer alternative has been enabled.

Table A1.1 Constants used within the IMS T9000



IMS T9000 quick reference guide

B1 IMS T9000 quick reference guide

This section is intended to provide a quick reference guide to all the key data required to run the device. The following is a listing of all the information contained in this section.

- IMS T9000 electrical specifications
- IMS T9000 timings
- IMS T9000 processor speed select table
- IMS T9000 link speed select table
- IMS T9000 package details

B1.1 Electrical specifications

B1.1.1 Absolute maximum ratings

Symbol	Parameter	Min	Max	Units	Notes
VDD	DC supply voltage	0	7.0	V	1,2,3,4,5
V _I , V _O	Voltage on input and output pins	-0.5	VDD+0.5	V	1,3,4,5
I _I	Input current		±25	mA	6
t _{osc}	Output short circuit time (one pin)		1	s	4
T _s	Storage temperature	-65	150	°C	4

Table B1.2 Absolute maximum ratings

Notes

- All voltages are with respect to **GND**.
- Power is supplied to the device via the **VDD** and **GND** pins. Several of each are provided to minimize inductance within the package. All supply pins must be connected. The supply must be decoupled close to the chip by at least one 100 nF low inductance (e.g. ceramic) capacitor between **VDD** and **GND**. Four layer boards are recommended; if two layer boards are used, extra care should be taken in decoupling.
- Input voltages must not exceed specification with respect to **VDD** and **GND**, even during power-up and power-down ramping, otherwise *latchup* can occur. CMOS devices can be permanently damaged by excessive periods of latchup.
- This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operating sections of this specification is not implied. Stresses greater than those listed may cause permanent damage to the device. Exposure to absolute maximum rating conditions for extended periods may affect reliability.
- This device contains circuitry to protect the inputs against damage caused by high static voltages or electrical fields. However, it is advised that normal precautions be taken to avoid application of any voltage higher than the absolute maximum rated voltages to this high impedance circuit. Unused inputs should be tied to an appropriate logic level such as **VDD** or **GND**.
- The input current applies to any input or output pin and applies when the voltage on the pin is between **GND** and **VDD**.

B1.1.2 Operating conditions

Symbol	Parameter	Min	Max	Units	Notes
VDD	DC supply voltage	4.75	5.25	V	1
V _I , V _O	Input or output voltage	0	VDD	V	1,2
CL	Load capacitance on any pin			pF	3

Table B1.3 Operating conditions

Notes

- All voltages are with respect to **GND**.
- Excursions beyond the supplies are permitted but not recommended.
- Maximum capacitance per address/ strobe/ data pin given in table B1.4.

B1.1.3 Power rating

Symbol	Parameter	Max	Units
C_{pinA}	Capacitance per address pin	250	pF
C_{pinS}	Capacitance per strobe pin	60	pF
C_{pinD}	Capacitance per data pin	60	pF
$n_{pA} * C_{pinA}$	Total address bus capacitance	2500	pF
$n_{pS} * C_{pinS}$	Total strobe pins capacitance	500	pF
$n_{pD} * C_{pinD}$	Total data bus capacitance	5000	pF

Table B1.4 Capacitance specifications

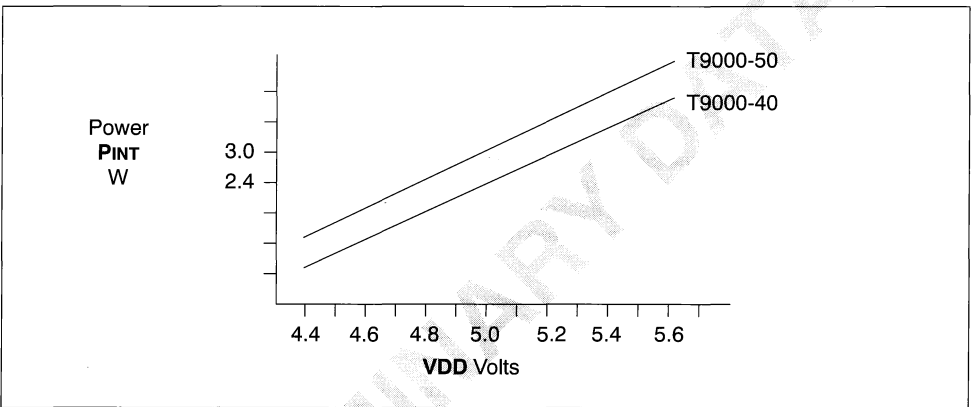


Figure B1.1 Internal power dissipation vs VDD

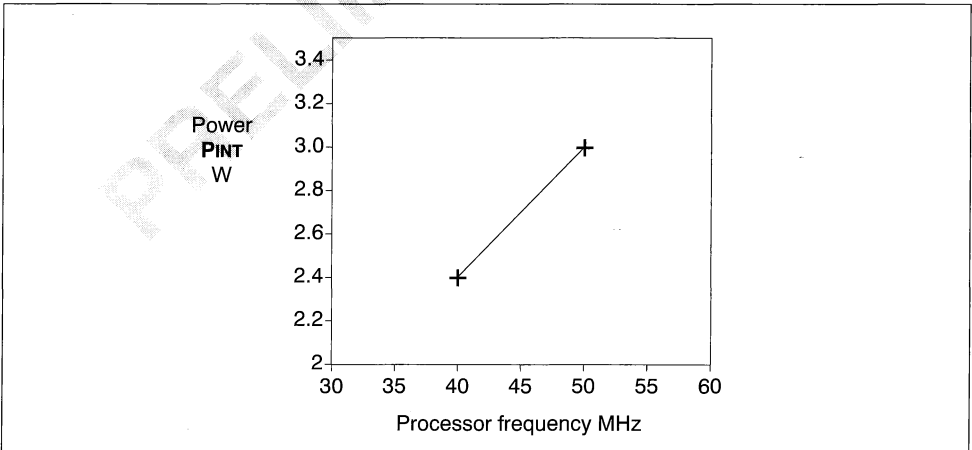


Figure B1.2 Typical peak internal power dissipation with processor speed

B1.2 Timing specifications

The following timings are based on simulations of the 50 MHz version of the IMS T9000 chip, and may change when full characterization is completed. The simulations were run under a loading of 75 pF unless otherwise stated.

B1.2.1 ClockIn timings

Symbol	Parameter	Min	Nom	Max	Units	Notes
tDCLDCH	ClockIn pulse width low	40			ns	
tDCHDCL	ClockIn pulse width high	40			ns	
tDCLDCL	ClockIn period		200		ns	1, 2
tDCr	ClockIn rise time			10	ns	3
tDCf	ClockIn fall time			8	ns	3

Table B1.5 **ClockIn** timings

Notes

- 1 Measured between corresponding points on consecutive falling edges.
- 2 This value allows the use of 200 ppm crystal oscillators for two devices connected together by a link.
- 3 Clock transitions must be monotonic within the range V_{IH} to V_{IL} (refer to Electrical specifications section B1.1).

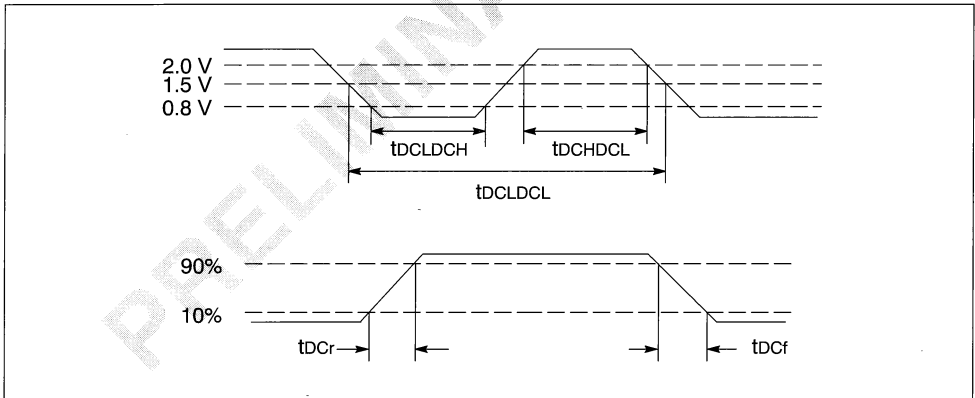


Figure B1.3 **ClockIn** timing

B1.2.2 ProcClockOut timings

The simulations were run under a range of output loads from 10 pF to 70 pF (capacitive only).

Symbol	Parameter	Min	Nom	Max	Units	Notes
tPCLPCL	ProcClockOut period		20		ns	
tPCHPCL	ProcClockOut pulse width high	9.0	10	11.4	ns	
tPCLPCH	ProcClockOut pulse width low	8.6	10	11.0	ns	
tPCstab	ProcClockOut stability				%	1

Table B1.6 ProcClockOut timings

Notes

- 1 Stability is the variation of cycle periods between two consecutive cycles, measured at corresponding points on the cycles.

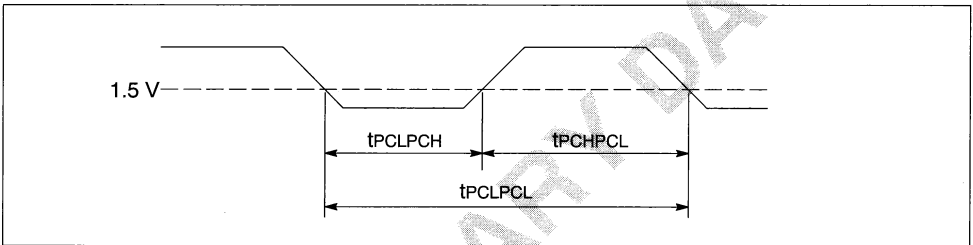


Figure B1.4 ProcClockOut timing

B1.2.3 Programmable memory interface timings

The IMS T9000 PMI has three types of pins; address, data and strobe pins. The falling and rising edges of the strobe pins are programmable through the configuration registers. The following specification is given either as an absolute timing value or as a skew (Δt_{SPEC}) from the nominal programmed value (t_n). The relationship between the parameter (t_{SPEC}) as specified on the timing diagrams, the skew and the programmed value is as follows: $t_{SPEC} = t_n + \Delta t_{SPEC}$.

Symbol	No.	Parameter	Min	Max	Units	Notes
Δt_{AVSV}	1	Address setup to strobe valid	-6	+2	ns	1,2
Δt_{SVAV}	2	Address hold after strobe valid	-2	+6	ns	1,3
Δt_{SVSV}	3	Strobe valid to strobe valid	-4	+4	ns	1,4
Δt_{SLSH}	4	Strobe low to strobe high	-4	+4	ns	1,5
Δt_{SHSL}	9	Strobe high to strobe low	-4	+4	ns	1,5
t_{RDS}	5	Read data valid after notMemCAS low		$5((N*4)-e1)-18$	ns	9
t_{RDH}	6	Read data hold after notMemCAS high	$5((N*4)-e2)-10$		ns	10
Δt_{WDS}	7	Write data setup before related strobe low	-4	+4	ns	6, 8
Δt_{WDH}	8	Write data hold after related strobe high	-4		ns	7, 8
t_{SVVW}	10	Wait setup time		$5((m*4)-e1)-8$	ns	11, 12
t_{SVWI}	11	Wait hold time	$5((m*4)-e1)+7$		ns	11, 12

Table B1.7 PMI AC specifications

Notes

- The timings are based on the following loading conditions: address pin loaded with 75 pF and strobe pins loaded with 75 pF.
- The nominal value is given by the programmed position of the falling edge (**RASEdgeTime** or **E1Time**) of the **RAS**, **CAS** or **PS** strobe.
- The nominal value is given by the programmed length of the sub-cycle (**RASTime** or **CASTime**) minus the programmed position of the falling strobe edge (**RASEdgeTime** or **E1Time**).
- The nominal value is given by the absolute difference between any pair of falling edges (**RASEdgeTime** or **E1Time**) or any pair of rising edges (**E2Time**) or any falling to rising pair of strobes.
- The nominal strobe pulse width is given by the difference between the programmed falling (**RASEdgeTime** or **E1Time**) and the programmed rising edge (**E2Time**) of a **RAS**, **CAS** or **PS** strobe. This is applicable to a positive or negative pulse.
- The nominal value is given by the programmed **E1Time** for **notMemCAS0-3** or **notMemWrB0-3**.
- The nominal value is given by the programmed **CASTime** minus the programmed position of the rising edge of **notMemCAS0-3** or **notMemWrB0-3**. Output data may be held indefinitely if there are no subsequent external cycles.
- Timings are for all four byte write strobes **notMemWrB0-3**.
- Where, **N** is the number of **CASTime** cycles and **e1** is the programmed falling edge position (**E1Time**).
- Where, **N** is the number of **CASTime** cycles and **e2** is the programmed rising edge position (**E2Time**).
- A wait state will be inserted between cycle **m** and cycle **m+1** in an **n** cycle access if the specified relationship is met.
- In an **n** cycle access a wait cycle cannot be added after cycle **n**. Also, a wait cycle cannot be added before cycle 1.

Read cycle

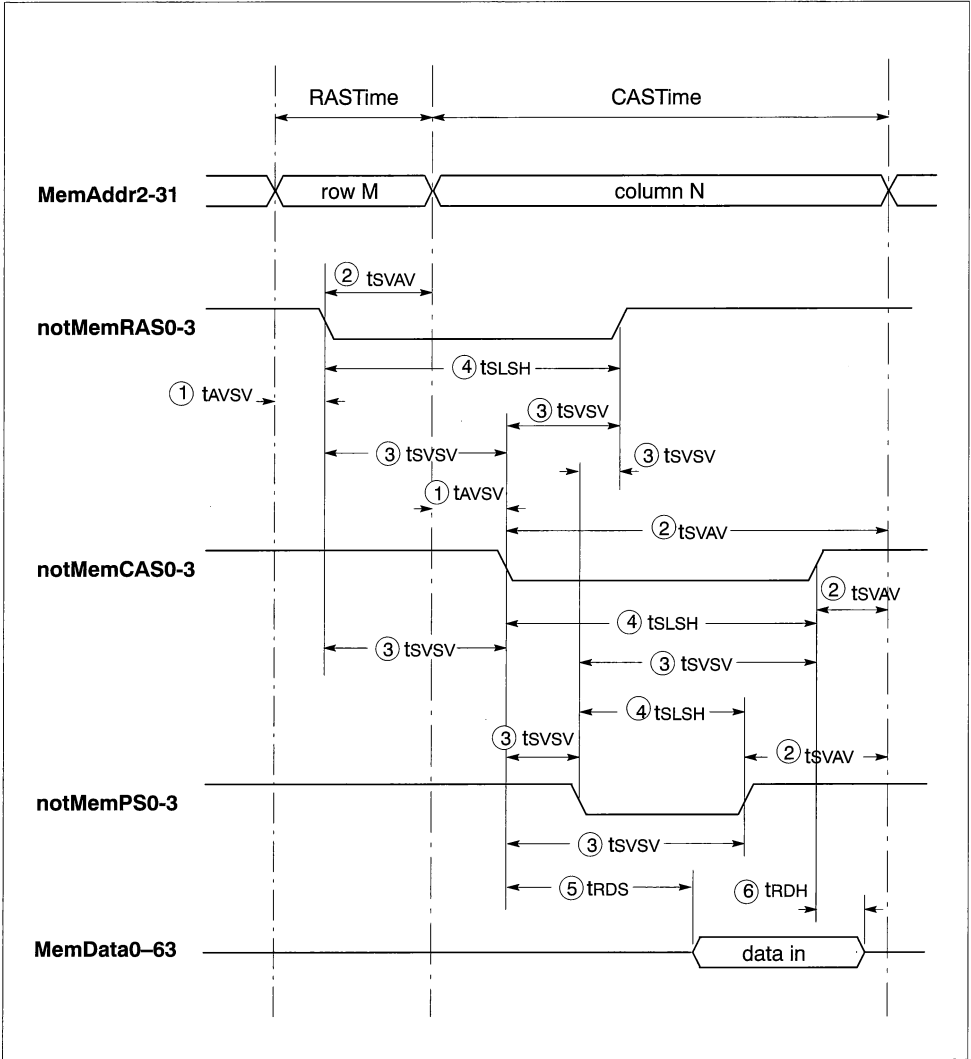


Figure B1.5 Read cycle timing

Write cycle

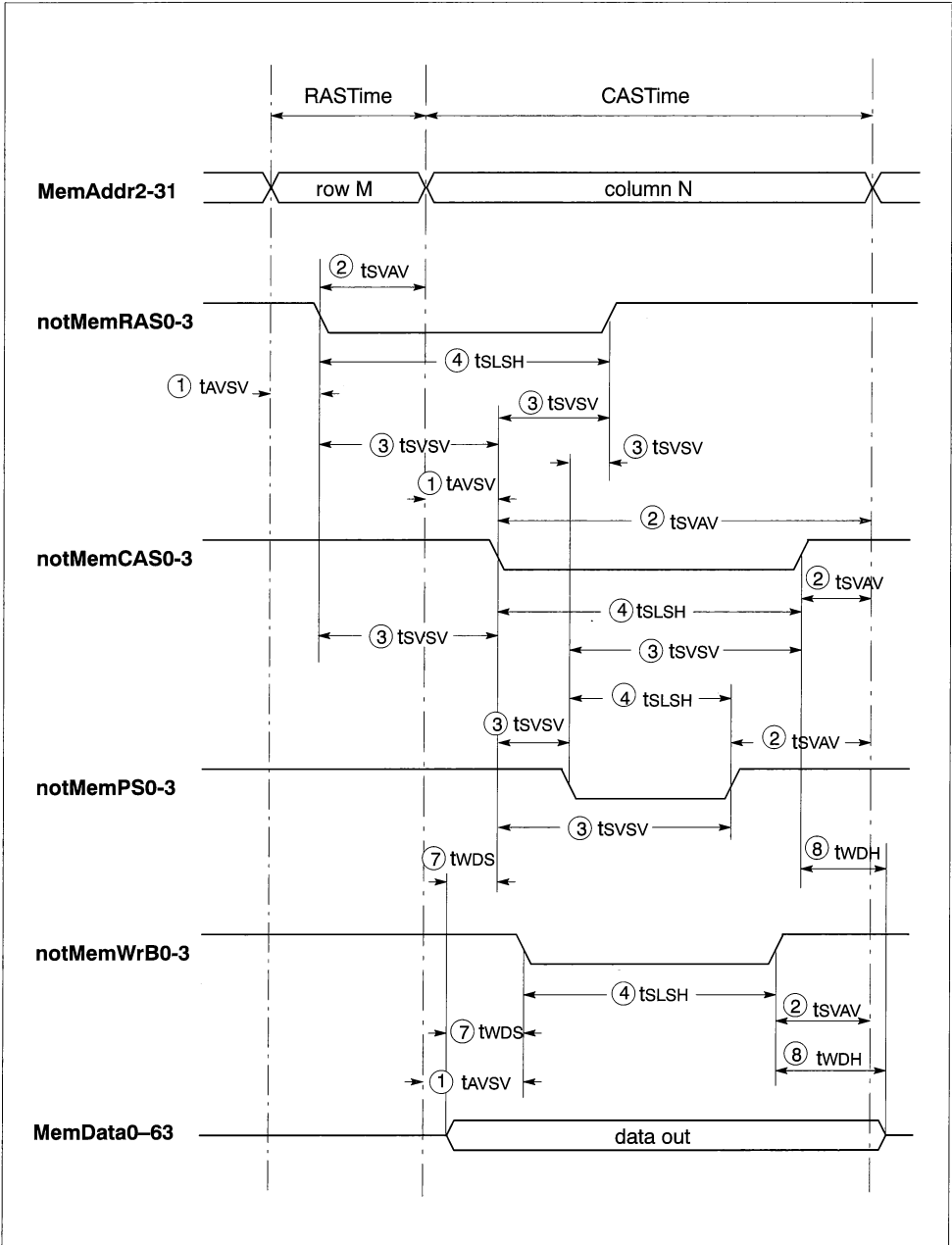


Figure B1.6 Write cycle timing

Consecutive cycles

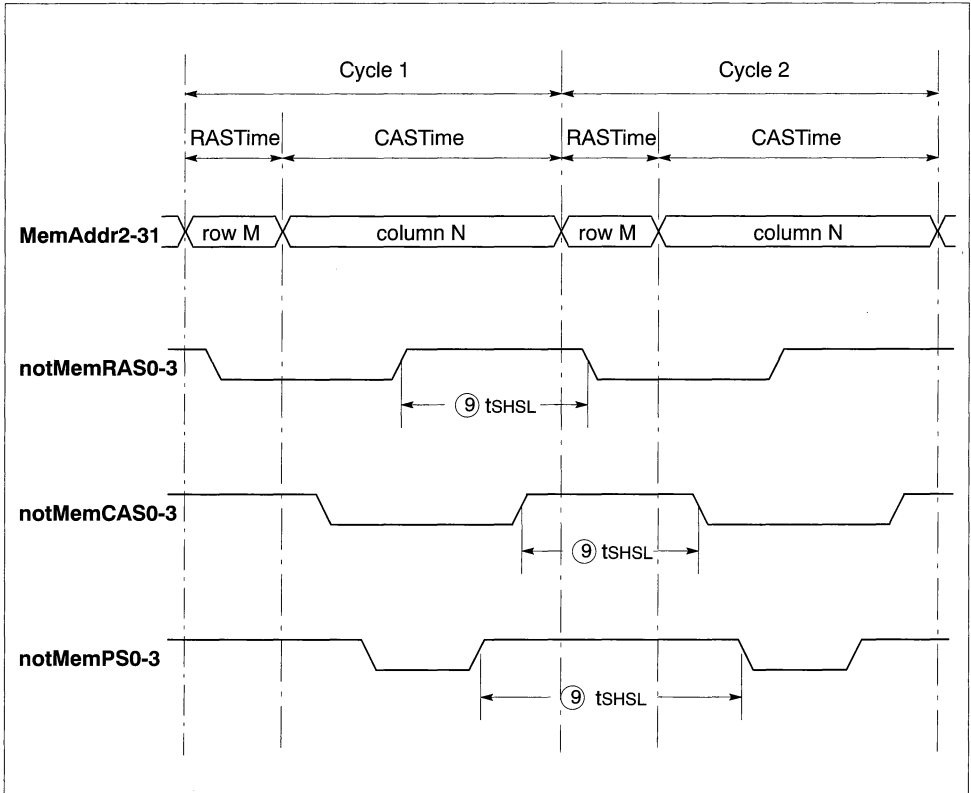


Figure B1.7 Cycle to cycle timing

Memory wait

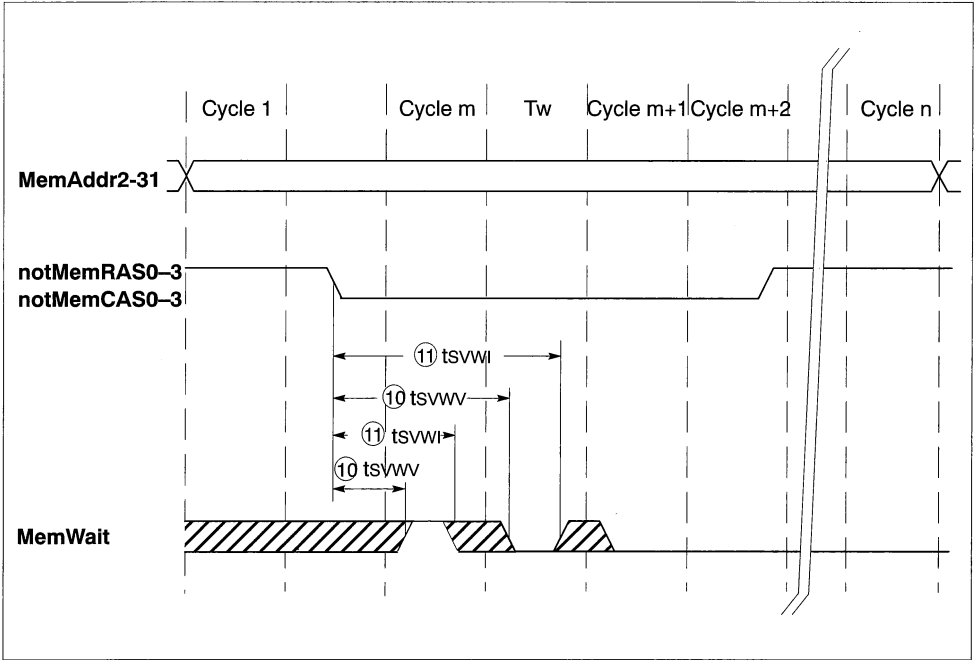


Figure B1.8 Memory wait timing

B1.2.4 Link timings

Symbol	Parameter	Min	Nom	Max	Units
tLODSr	LinkOut rise time		4		
tLODSf	LinkOut fall time		4		
tLIDSr	LinkIn rise time		4		
tLIDSf	LinkIn fall time		4		
tLIHL	Input edge resolution	2			ns
tSDS	Bit period	10		100	ns
Δt_{DSO}	Data / strobe output skew			1	ns
CLIZ	LinkIn capacitance		7		pF

Table B1.8 DS link timings

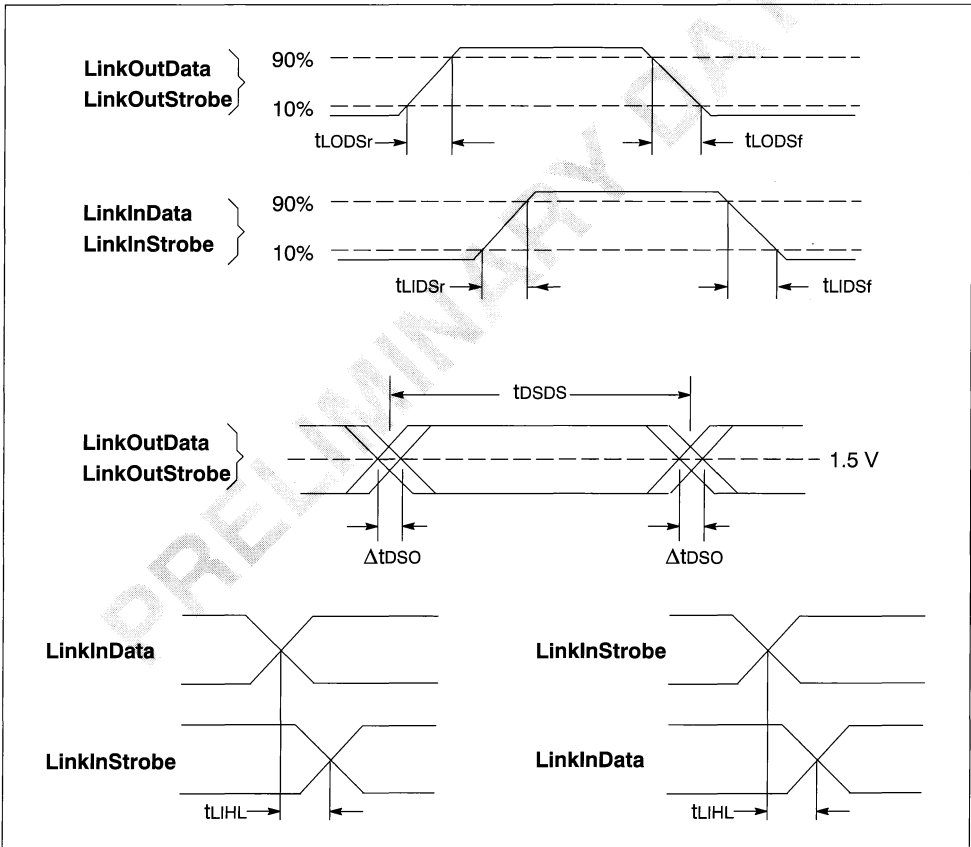


Figure B1.9 DS link timing

B1.3 Processor speed select

ProcSpeed Select2	ProcSpeed Select1	ProcSpeed Select0	Processor clock speed MHz	Processor cycle time ns	Phase lock loop factor (PLLx)
0	0	0	30	33.3	6.0
0	0	1	35	28.6	7.0
0	1	0	40	25.0	8.0
0	1	1	45	22.2	9.0
1	0	0	50	20.0	10.0
1	0	1	INMOS reserved		
1	1	0	INMOS reserved		
1	1	1	INMOS reserved		

Table B1.9 Processor speed selection

B1.4 Link speed select

SpeedMultiply	SpeedDivide1:0				BaseSpeed
	0:0	0:1	1:0	1:1	
	/ 1	/ 2	/ 4	/ 8	
8	80	40	20	10.0	10
10	100	50	25	12.5	10
12	Reserved	60	30	15.0	10
14	Reserved	70	35	17.5	10
16	Reserved	80	40	20.0	10
18	Reserved	90	45	22.5	10
20	Reserved	100	50	25.0	10

Table B1.10 Link transmission speed in Mbits/sec

B1.5.2 208 pin CLCC package dimensions

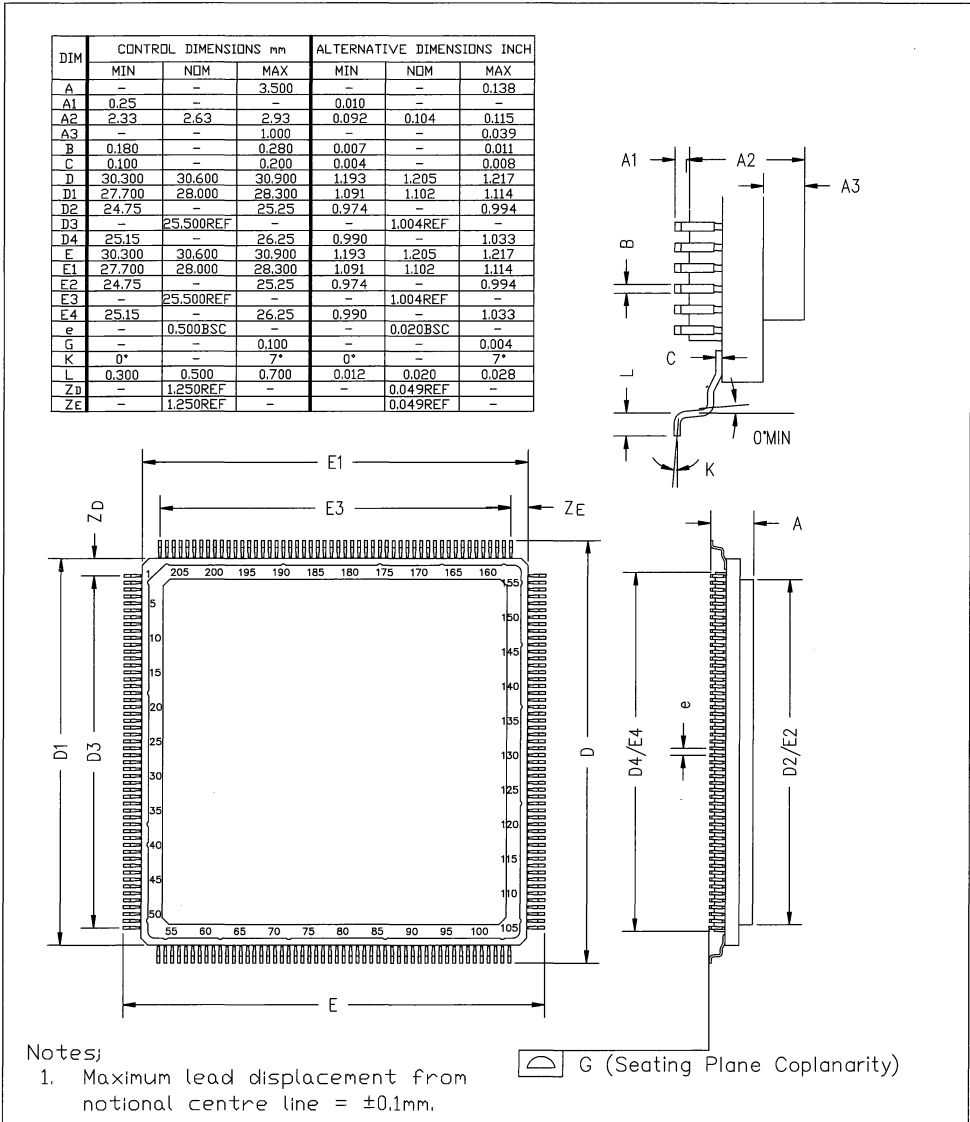


Figure B1.11 208 pin CLCC package dimensions

B1.5.3 208 pin CLCC package thermal characteristics

The junction to case thermal resistance (θ_{JC}) of the package is given below.

Symbol	Parameter	Min	Nom	Max	Units
θ_{JC}	Junction to case thermal resistance			1	°C/W

Table B1.11 Thermal resistance

Index for Parts 1 and 2

A

accesses
 cacheable, 140
 non-cacheable, 140
 page-mode, 155

address bus, 150, 153

Address0-3, 164

ajw, 79

Areg, 65

autonomous operation, 107

B

bank switching, 159

BaseSpeed, 213

block move, 72

Boot, 116

boot sequence, 177

BootData, 116

BootDataHandshake, 116

BootHandshake, 116

booting
 from link, 124
 from ROM, 178
 from ROM then link, 125

bootspace, 177
 timing, 178

Breg, 65

BusReleaseTime, 155

byte-stream mode, 195

C

cache
 instruction and data, 129
 workspace, 66

cache instructions, 144
 performance, 145

CacheMode, 140, 167

call, 79

CapMinus, 221

CapPlus, 221

CASStrobe0-3, 169

CASTime, 155

central processing unit, 65
 configuration registers, 75

channel addresses, 196

channels, 187
 byte-stream, 195
 events, 205
 external, 187
 internal, 187
 resetting, 193
 stopping, 194
 virtual, 189

ChanWriteLock, 202

CLink0-1, 110

clock input, timings, 223

clock stability, 221

ClockIn, 221

clocking system, 221

configuration, 124

configuration bus, 225

configuration bus masters, 225

configuration register addresses
 cache, 230
 central processing unit, 227
 control link, 231
 data/strobe link, 230
 programmable memory interface, 227
 scheduler, 230
 system services, 229
 virtual channel processor, 229

configuration registers
 addresses, 225
 cache, 146
 control link, 110
 CPU, 75
 data/strobe links, 215
 programmable memory interface, 164
 system services, 113
 virtual channel processor, 199

configuration space, 225

constants. *See* special values

control commands, 115

control link, errors, 122

control links, 110

control system, 105
 error codes, 121
 errors, 121
 tiers of handshaking, 106

CPeek, 116

CPeekHandshake, 116

CPoke, 116

CPokeHandshake, 116

CPU. *See* central processing unit

CPU write locking, 226

Creg, 65

crossbar switch, 134

cycle, 149

D

data bus, 150, 153

data/strobe links, 209
 configuration registers, 215
 errors, 213

de-activated channel, 193

debugging
 P-processes, 83
 systems, 123

DeviceID, 113

DeviceRevision, 113

direct memory access, 151, 163

disg, 194

DMA. *See* direct memory access

DoAllocate, 147

DoPMIConfigured, 168, 178

DoRamSize, 146

DRAM cycles, 155

DSLlinkPLL, 114, 215

E

electrical specifications, 239
 absolute maximum ratings, 239
 operating conditions, 239

EmiBadAddress, 75

enbg, 194

endpw, 68

error code, virtual channel processor, 200

Error message, 119
 codes, 121

ErrorAddress, 168, 178

ErrorCode, 114, 121

ErrorHandshake, 119

errors
 control link, 122
 control system, 121
 data/strobe links, 213
 in stand alone mode, 122
 programmable memory interface, 176
 virtual channel processor, 204

event channel addresses, 206

EventIn0-3, 205

EventOut0-3, 205

events, 205

external bus cycles, 152

external cycles
 DRAM, 155
 non-DRAM, 158

ExternalIRCBASE, 197, 198

F

fdca, 145

fdcl, 145

floating point
 formats, 77
 instructions, 78
 registers, 77
 rounding modes, 78

floating point unit, 77

floating-point status register, 77

forced air flow cooling, 236

FormatControl0-3, 165

FPAreg, 65, 77

FPBreg, 65, 77

FPCreg, 65, 77

FPstatusReg. *See* floating point status register

FPU. *See* floating point unit

G

gajw, 79

goprot, 79

grant, 194

H

HdrAreaBase, 197, 200

heat sinks, 236

I

ica, 145

icl, 145

Identify, 115

IdentifyHandshake, 115

in, 187

InitialPtr, 75

initialization

instruction and data cache, 147

links, 217

programmable memory interface, 177

virtual channel processor, 203

InitialWptr, 75

instruction

characteristics, 88

format, 85

instruction and data cache, 129

address bits, 131

arbitration, 138

cache memory, 132

configuration registers, 146

initialization, 147

line structure, 132

operation, 136

structure, 130

instruction pointer, 65

IptrReg. *See* instruction pointer

J

j0, 71

L

L-processes, 69

workspace, 69

ldcnt, 188

ldconf, 225

lddevvid, 88

ldtimer, 71

link

connections, 218

errors

reliable links, 214

unreliable links, 214

format, 209

initialization, 217

protocol, 209

reset, 217

speeds, 212

timings, 219

link disconnection, 213

link queues, 190

Link0-3Command, 215

Link0-3Mode, 215

links, 209

control, 110

data/strobe, 209

TTL compatibility, 218

LocalizeError

links, 213, 215

virtual channel processor, 202, 204

M

main cache. *See* instruction and data cache

Mask0-3, 164, 165

MemAddr2-31, 150

MemData0-63, 150

MemGranted, 151

memory management, 79

registers, 83

MemReqIn, 151

MemReqOut, 151

MemStart, 198

MemWait, 151, 171

message, 189

MinInvalidChannel, 198

mkrc, 194

ModeStatus, 113

move, 72

N

ntfix, 88
 non-DRAM cycles, 158
notMemBootCE, 152, 178, 180
notMemCAS0-3, 151
notMemPS0-3, 151
notMemRAS0-3, 150
notMemRf, 152
notMemStrobe, 152
notMemWrB0-3, 150

O

out, 187

P

P-processes, 79
 debugging, 83
 region addresses, 80
 region descriptor, 82
 package dimensions, 208 pin CLCC, 234, 328
 package pinout, 233, 327
 package specifications, 233
 package thermal characteristics, 234, 328
 packets, 189
 structure, 189
 page-mode accesses, 155
 parity errors, 213
Peek, 116
PeekHandshake, 116
 performance, 101
 cache instructions, 145
 floating point operations, 78, 103
 integer operations, 101
 predefines, 104
prefix, 88
 phase, 149
 phase lock loop factor, 222
 phase locked loops, 221
 decoupling, 221
 pin descriptions, 63

pipeline, 72
 groupings, 73
 operation, 73
 stages, 72, 85
 PMI. *See* programmable memory interface
PMIWriteLock, 175
Poke, 116
PokeHandshake, 116
PortSize, 167
 power rating, 240
PrechargeTime, 155
 primary instructions, 90
ProcClockOut, 222
 processes, 67
 active, 67
 inactive, 67
 L-processes, 69
 P-processes, 79
 priority, 68
 processor clock output, timings, 224
 processor speed, 222
ProcSpeedSelect0-2, 222
 programmable memory interface, 149
 configuration registers, 164
 errors, 176
 initialization, 177
ProgStrobe0-3, 169
 protection, 79
PstateReg, 83

R

RamAddress, 147
RamLineNumber, 147
RamSize, 146
RASBits0-3, 165
RASStrobe0-3, 169
RASTime, 155
Reason, 75
Reboot, 116
RebootHandshake, 116
RecoverError, 116, 122
RecoverErrorHandshake, 116
 refill cycles, 141, 162
 refill engine, 135

refresh, 152, 174
RefreshControl, 174
 region descriptor, 82
 registers, 83
RegionReg0-3, 83
 regions, 80
RemapBootBank, 175
 replace pointer, 135
Reset, 126
Reset, 115
 reset, 126
 level 0, 126
 level 1, 115, 126
 level 2, 115, 127
 level 3, 115, 127
 links, 217
 virtual channel processor, 203
resetch, 193, 217
ResetHandshake, 115
 resetting, channels, 193
 resource data structure, 194
 resources, 194
Run, 116
RunHandshake, 116

S

secondary instructions, 91
 semaphores, 72
 data structure, 72
 server process, 194
setchmode, 217
ShiftAmount, 165
signal, 72
 special values, 311
 speed selection
 link, 212
 processor, 222
SpeedDivide, 212
SpeedMultiply, 114, 212
 stability clock, 221
 stack
 floating point, 65, 77
 integer, 65

stack extension, 79
 stand alone mode, 107
 errors, 122
Start, 115
StartFromROM, 124, 178
StartHandshake, 115
startp, 68
 status register, 70
StatusReg, 70
 See also status register
stconf, 225
Stop, 71, 115, 123
stopch, 194
StopHandshake, 115
 stopping, channels, 194
 strobes
 CAS, 151
 programmable, 151
 RAS, 150
 registers, 169
 write, 150
syscall, 79
SysServWriteLock, 114
 system services, 113

T

thermal management, 235
 forced air flow cooling, 236
 heat sinks, 236
ThReg. *See* trap handler register
 timers, 71
 timer registers, 71
TimingControl0-3, 171
 timings
 bootspace, 178
 clock input, 223
 cycle to cycle, 185
 data/strobe link, 219
 memory wait, 186
 processor clock output, 224
 read cycle, 183
 write cycle, 184
tin, 71
 trap handler, 69
 data structure, 70
 trap-handler register, 69

V

variable-length communication, 188
VCP. *See* virtual channel processor
VCPCCommand, 199
VCPLink0-3HdrOffset, 200
VCPLink0-3MaxHeader, 202
VCPLink0-3MinHeader, 202
VCPLink0-3Mode, 202
VCPStatus, 199
vin, 188
virtual channel processor, 189
 error code, 200
 errors, 204
 initialization, 203
 operation, 193
 protocol, 189

 reset, 203

virtual channels, 189

virtual link control block, 191

virtual links, 189

vout, 188

W

wait, 72

WdescStubReg, 83

workspace cache, 66
 operation, 66

workspace pointer, 65

Wptr. *See* workspace pointer

write-back cycles, 143

WriteStrobe0-3, 169

SALES OFFICES

EUROPE

DENMARK

2730 HERLEV

Herlev Torv, 4
Tel. (45-44) 94.85.33
Telex: 35411
Telefax: (45-44) 948694

FINLAND

LOHJA SF-08150

Katakatu, 26
Tel. (358-12) 155.11
Telefax: (358-12) 155.66

FRANCE

94253 GENTILLY Cedex

7 - avenue Gallieni - BP. 93
Tel.: (33-1) 47.40.75.75
Telex: 632570 STMHQ
Telefax: (33-1) 47.40.79.10

67000 STRASBOURG

20, Place des Halles
Tel. (33-88) 75.50.66
Telefax: (33-88) 22.29.32

GERMANY

8011 GRASBRUNN

Bretonischer Ring 4
Postfach 1122
Tel.: (49-89) 460060
Telefax: (49-89) 4605454
Teletex: 897107=STDISTR

1000 BERLIN 37

Clay Allee 323
Tel.: (49-30) 8017087-89
Telefax: (49-30) 8015552

6000 FRANKFURT

Gutleutstrasse 322
Tel. (49-69) 237492-3
Telefax: (49-69) 231957
Teletex: 6997689=STVBF

3000 HANNOVER 51

Rotenburger Strasse 28A
Tel. (49-511) 615960-3
Teletex: 5118418 CSFBEH
Telefax: (49-511) 6151243

8500 NÜRNBERG 20

Erlenstegenstrasse, 72
Tel.: (49-911) 59893-0
Telefax: (49-911) 5980701

7000 STUTTGART 31

Mittlerer Pfad 2-4
Tel. (49-711) 13968-0
Telefax: (49-711) 8661427

ITALY

20090 ASSAGO (MI)

V.le Milanofiori - Strada 4 - Palazzo A/4/A
Tel. (39-2) 89213.1 (10 linee)
Telex: 330131 - 330141 SGSAGR
Telefax: (39-2) 8250449

40033 CASALECCHIO DI RENO (BO)

Via R. Fucini, 12
Tel. (39-51) 593029
Telex: 512442
Telefax: (39-51) 591305

00161 ROMA

Via A. Torlonia, 15
Tel. (39-6) 8443341
Telex: 620653 SGSATE I
Telefax: (39-6) 8444474

NETHERLANDS

5652 AR EINDHOVEN

Meerenakkerweg 1
Tel.: (31-40) 550015
Telex: 51186
Telefax: (31-40) 528835

SPAIN

08021 BARCELONA

Calle Platon, 6 4th Floor, 5th Door
Tel. (34-3) 4143300-4143361
Telefax: (34-3) 2021461

28027 MADRID

Calle Albacete, 5
Tel. (34-1) 4051615
Telex: 46033 TCCEE
Telefax: (34-1) 4031134

SWEDEN

S-16421 KISTA

Borgarfjordsgatan, 13 - Box 1094
Tel.: (46-8) 7939220
Telex: 12078 THSW5
Telefax: (46-8) 7504950

SWITZERLAND

1218 GRAND-SACONNEX (GENEVA)

Chemin Francois-Lehmann, 18/A
Tel. (41-22) 7986462
Telex: 415493 STM CH
Telefax: (41-22) 7984869

UNITED KINGDOM and EIRE

MARLOW, BUCKS

Planar House, Parkway
Globe Park
Tel.: (44-628) 890800
Telex: 847458
Telefax: (44-628) 890391

AMERICAS**BRAZIL**

05413 SÃO PAULO
R. Henrique Schaumann 286-CJ33
Tel. (55-11) 883-5455
Telex: (391)11-37988 "UMBR BR"
Telefax: (55-11) 282-2367

CANADA**NEPEAN ONTARIO**

301 Moodie Drive
Suite 307
Tel. 613/829-9944

U.S.A.

NORTH & SOUTH AMERICAN
MARKETING HEADQUARTERS
1000 East Bell Road
Phoenix, AZ 85022
(1-602) 867-6100

SALES COVERAGE BY STATE**ALABAMA**

Huntsville - (205) 533-5995

ARIZONA

Phoenix - (602) 867-6217

CALIFORNIA

Santa Ana - (714) 957-6018
San Jose - (408) 452-8585

COLORADO

Boulder (303) 449-9000

ILLINOIS

Schaumburg - (708) 517-1890

INDIANA

Kokomo - (317) 455-3500

MASSACHUSETTS

Lincoln - (617) 259-0300

MICHIGAN

Livonia - (313) 953-1700

NEW JERSEY

Voorhees - (609) 772-6222

NEW YORK

Poughkeepsie - (914) 454-8813

NORTH CAROLINA

Raleigh - (919) 787-6555

TEXAS

Carrollton - (214) 466-8844

FOR RF AND MICROWAVE
POWER TRANSISTORS CON-
TACT

THE FOLLOWING REGIONAL
OFFICE IN THE U.S.A.

PENNSYLVANIA

Montgomeryville - (215) 361-6400

ASIA / PACIFIC**AUSTRALIA**

NSW 2220 HURTSVILLE
Suite 3, Level 7, Otis House
43 Bridge Street
Tel. (61-2) 5803811
Telefax: (61-2) 5806440

HONG KONG**WANCHAI**

22nd Floor - Hopewell centre
183 Queen's Road East
Tel. (852) 8615788
Telex: 60955 ESGIES HX
Telefax: (852) 8656589

INDIA**NEW DELHI 110001**

LiasionOffice
62, Upper Ground Floor
World Trade Centre
Barakhamba Lane
Tel. (91-11) 3715191
Telex: 031-66816 STMI IN
Telefax: (91-11) 3715192

MALAYSIA**PETALING JAYA, 47400**

11C, Jalan SS21/60
Damansara Utama
Tel.: (03) 717-3976
Telefax: (03) 719-9512

PULAU PINANG 10400

4th Floor - Suite 4-03
Bangunan FOP-123D Jalan Anson
Tel. (04) 379735
Telefax (04) 379816

KOREA**SEOUL 121**

8th floor Shinwon Building
823-14, Yuksam-Dong
Kang-Nam-Gu
Tel. (82-2) 553-0399
Telex: SGGKOR K29998
Telefax: (82-2) 552-1051

SINGAPORE**SINGAPORE 2056**

28 Ang Mo Kio - Industrial Park 2
Tel. (65) 4821411
Telex: RS 55201 ESGIES
Telefax: (65) 4820240

TAIWAN**TAIPEI**

12th Floor
325, Section 1 Tun Hua South Road
Tel. (886-2) 755-4111
Telex: 10310 ESGIE TW
Telefax: (886-2) 755-4008

JAPAN**TOKYO 108**

Nisseki - Takanawa Bld. 4F
2-18-10 Takanawa
Minato-Ku
Tel. (81-3) 3280-4121
Telefax: (81-3) 3280-4131