PM7380

FREEDM™-32P672

FRAME ENGINE AND DATALINK MANAGER 32P672

PROGRAMMER'S GUIDE

PROPRIETARY AND CONFIDENTIAL

ADVANCE

ISSUE 2: JUNE 1999

CONTENTS

LIST OF FIGURES

1 INTRODUCTION

1.1 Scope

The FREEDM-32P672 Programmer's Guide is intended to describe the configurable features and operation of a FREEDM-32P672 from a programmer's perspective. This document may not cover all applications of the FREEDM-32P672. Please contact a PMC-Sierra Applications Engineer for specific uses not covered in this document.

This document is a supplement to the FREEDM-32P672 Longform Datasheet[1]. Both documents should be studied together to interface the FREEDM-32P672 to an embedded processor. In case of a discrepancy between the Programmer's Guide and the Longform Datasheet, the Longform Datasheet shall always be considered correct.

1.2 Target Audience

This document has been prepared for readers with some prior knowledge of the HDLC protocol.

Although the examples provided in this document are described in C language syntax, they are not meant as compile-ready code segments.

1.3 Numbering Conventions

The following numbering conventions are used throughout this document:

binary	011 1010B, 011
decimal	129, 6, 12
hexadecimal	0x1FE2, 09FH

1.4 Register Description

Unless specified otherwise, FREEDM-32P672 registers are described using the convention **REGISTER_NAME**(byte offset from base address). There are two register spaces that can be addressed on a FREEDM-32P672 – they are the normal mode registers and the PCI configuration registers.

1.4.1 Normal Mode Registers

Normal mode registers are used to configure, monitor and control the operation of the FREEDM-32P672. Registers must be accessed as 32-bit values with a dword aligned address. A register value is accessed through the PCI Host interface during a PCI bus read, or write transaction, and has the following characteristics:

- Writing values into unused register bits has no effect. However, to ensure software compatibility with future, feature-enhanced versions of the product, unused register bits should be written with logic zero. Reading back unused bits can produce either a logic one or a logic zero; hence, unused register bits should be masked off by software during a register read access.

- Except where noted, all configuration bits that can be written into can also be read back. This allows the processor controlling the FREEDM-32P672 to determine the programming state of the block.

- Writable normal mode registers are cleared to logic zero upon reset unless otherwise noted.

- Writing into read-only normal mode register bit locations does not affect FREEDM-32P672 operation unless otherwise noted.

- Certain register bits are reserved. These bits are associated with megacell functions that are unused in this application. To ensure that the FREEDM-32P672 operates as intended, reserved register bits must only be written with their default values. Similarly, writing to reserved registers should be avoided.

1.4.2 PCI Configuration Registers

PCI configuration registers are defined by the PCI SIG[2] and are used to install and configure devices on the PCI bus. Registers must be accessed as 32-bit values with a dword aligned address. A register value is only accessed through the PCI Host interface during a PCI configuration cycle, and has the following characteristics:

- Writing values into unused register bits has no effect. However, to ensure software compatibility with future, feature-enhanced versions of the product, unused register bits must be written with logic zero. Reading back unused bits can produce either a logic one or a logic zero; hence unused register bits should be masked off by software when read.

- Except where noted, all configuration bits that can be written into can also be read back. This allows the processor controlling the FREEDM-32P672 to determine the programming state of the block.

- Writable PCI configuration register bits are cleared to logic zero upon reset unless otherwise noted.

- Writing into read-only PCI configuration register bit locations does not affect FREEDM-32P672 operation unless otherwise noted.

- Certain register bits are reserved. These bits are associated with megacell functions that are unused in this application. To ensure that the FREEDM-32P672 operates as intended, reserved register bits must only be written with their default values. Similarly, writing to reserved registers should be avoided.

2 REFERENCES

1. PMC-990262, PMC-Sierra, Inc., "Frame Engine and Data Link Manager 32P672" Longform Datasheet, May 1999, Issue 2.

2. PCI Special Interest Group, PCI Local Bus Specification, June 1, 1995, Version 2.1.

3. PMC-960758, PMC-Sierra, Inc., "Frame Engine and Data Link Manager" Longform Datasheet, May 1998, Issue 5.

3 FREEDM-32P672 OVERVIEW

3.1 FREEDM-32P672 Summary

The PM7380 FREEDM-32P672 Frame Engine and Datalink Manager is an advanced data link layer processor that is ideal for applications such as IETF PPP interfaces for routers, TDM switches, Frame Relay switches and multiplexors, ATM switches and multiplexors, Internet/Intranet access equipment, and packet-based DSLAM equipment. The FREEDM-32P672 implements HDLC processing and PCI Bus memory management functions for a maximum of 672 bi-directional channels. The functional blocks of the FREEDM-32P672 are illustrated in Figure 1.

As many as 32 bi-directional serial links can be connected to the FREEDM-32P672. These are processed by the Receive Channel Assigner (RCAS672) and Transmit Channel Assigner (TCAS672) blocks. The RCAS672 and TCAS672 can be interfaced to H-MVIP, channelised T1/J1/E1, or unchannelised links.

The data stream at each serial port can be assigned to one or more of the FREEDM-32P672 channels. There are as many as 672 receive channels and 672 transmit channels available for assignment to unchannelised links, or to time-slots within a channelised T1/J1/E1 or H-MVIP link.

When configured for 2.048 Mbps H-MVIP operation, the FREEDM-32P672 partitions the 32 physical links into 4 logical groups of 8 links. Links in each logical group share a common clock and a common type 0 frame pulse in each direction.

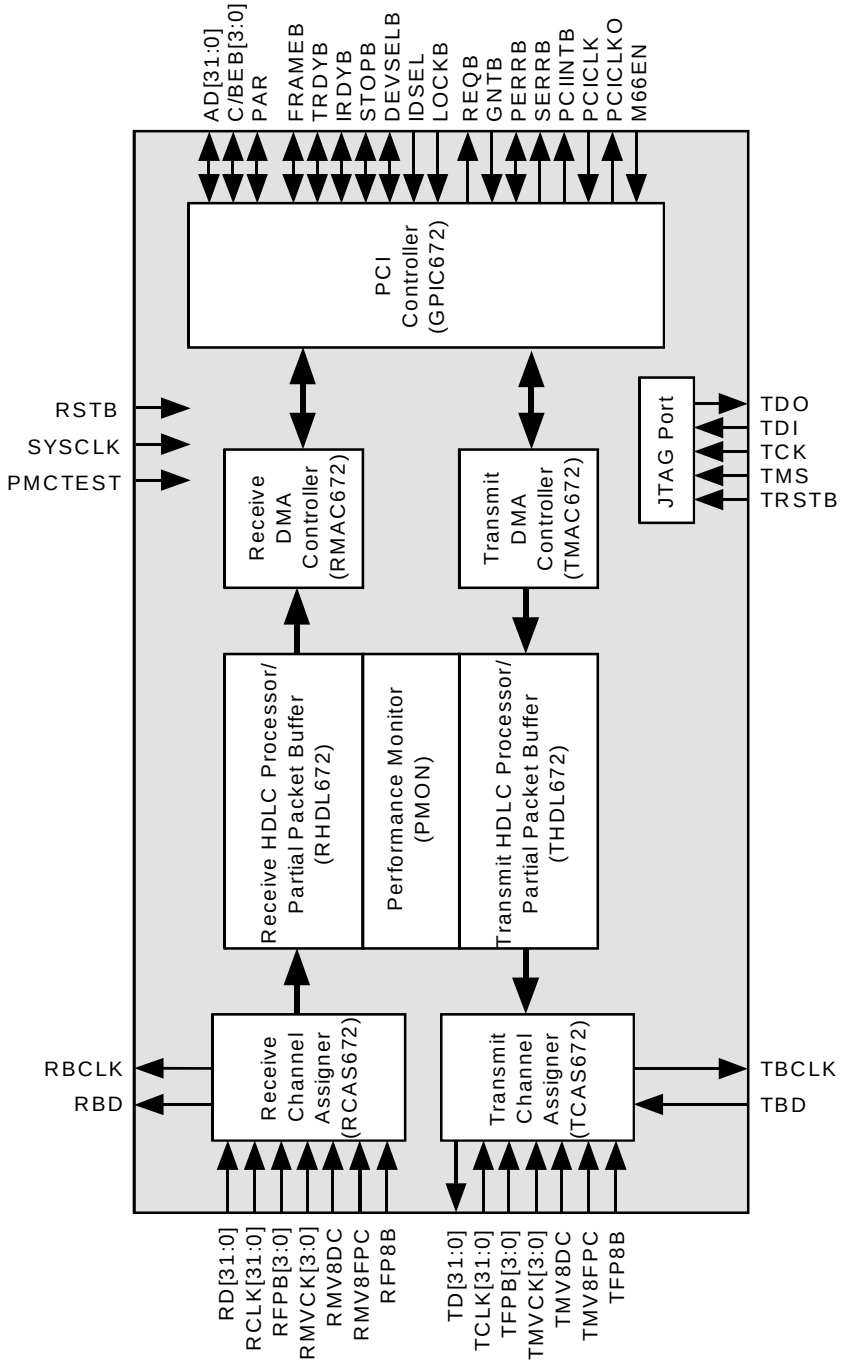
When configured for 8.192 Mbps H-MVIP operation, the FREEDM-32P672 partitions the 32 physical links into 8 logical groups of 4 links. All links configured for 8.192 Mbps H-MVIP operation will share a common type 0 frame pulse, a common frame pulse clock and a common data clock.

For channelised T1/J1/E1 links, the FREEDM-32P672 allows up to 672 bi-directional HDLC channels to be assigned to individual time-slots within a maximum of 32 independently timed T1/J1/E1 links.

For unchannelised links, the FREEDM-32P672 processes up to 32 bi-directional HDLC channels within 32 independently timed links.

Each data stream can be HDLC processed on a channelised basis within the Receive HDLC Processor / Partial Packet Buffer (RHDL672) and Transmit HDLC

Figure 1 – FREEDM-32P672 Block Diagram



Processor / Partial Packet Buffer (THDL672). There is a 32 Kbyte buffer in the RHDL672 and another 32 Kbyte buffer in the THDL672 that must be assigned to FREEDM-32P672 channels to serve as channel FIFO's. Each buffer is a group of 2048 blocks with 16 bytes per block, and a minimum of 3 blocks must be assigned to a channel during provisioning. This allows for flexible assignment of a channel FIFO based on the expected data rate for the channel.

Alternatively, the RHDL672 and THDL672 can provision a channel as transparent, in which case, the raw data stream is transferred without HDLC processing.

The FREEDM-32P672 interfaces to an embedded processor and packet memory through the PCI local bus[2]. The packet memory provides buffer locations where the receive data is written to, and where the transmit data is read from. Data is organized into packets on a per channel basis within the packet memory. The Receive DMA Controller (RMAC672), the Transmit DMA Controller (TMAC672) and the General-Purpose PCI Controller (GPIC672) blocks perform the DMA of buffer data across the PCI local bus.

Each channel provisioned within the FREEDM-32P672 contends for access to the PCI bus based on its configuration within the RMAC672 and TMAC672 blocks. This provides the designer with the flexibility to individually configure each channel to avoid receive overrun or transmit underrun, based on the channel data rate.

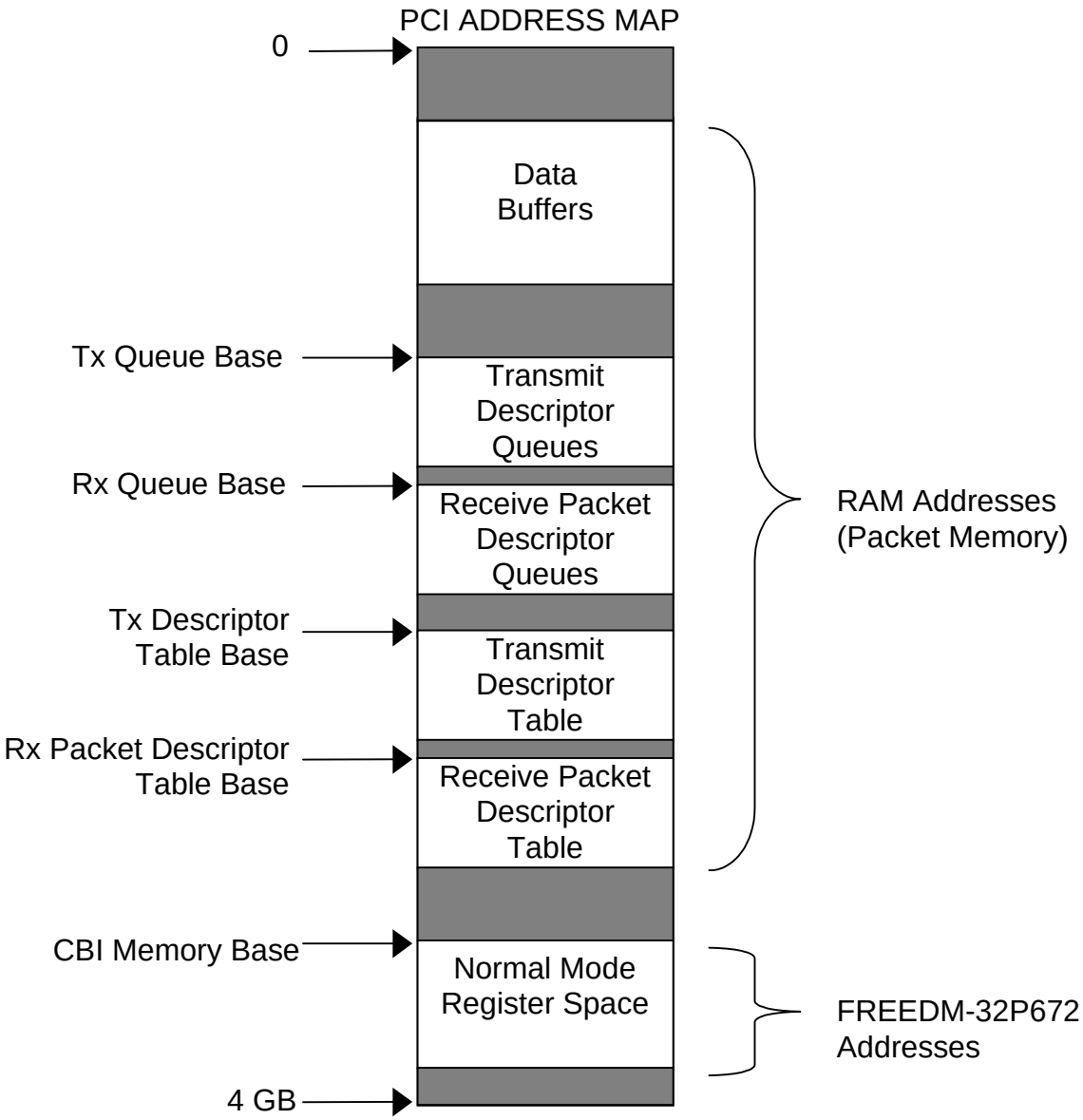
The PMON block provides performance monitor counts for a number of events. These counters can be read via the PCI interface and provides a means for the host software to accumulate performance statistics.

Channelised T1/J1/E1 and unchannelised links can be individually placed in line loopback. In addition to the line loopback capability, the FREEDM-32P672 provides a BERT port for attachment to BERT hardware. Under software control this port can be connected to any one of the 32 bi-directional links for additional diagnostic testing. The line loopback and BERT features are not supported for H-MVIP links. Finally, there is an internal diagnostic loopback configuration for each channel which can be used to diagnose FREEDM-32P672 operation on a per channel basis.

3.2 PCI Interface

Figure 2 shows an address map for a PCI bus which contains one FREEDM-32P672 device. These data structures are required to interface a FREEDM-32P672 to the PCI bus. In this figure, PCI addresses are 32-bit physical addresses which can be observed at the address pins of the bus.

Figure 2 – PCI Address Map



When multiple FREEDM-32P672's are attached to the bus, each FREEDM-32P672 must have a unique set of the following data structures:

- Transmit Descriptor Table
- Receive Packet Descriptor Table

- Transmit Queue Space
- Receive Queue Space
- Normal Mode Register Space

The data structures within RAM are accessed by software running on the embedded processor, or by the FREEDM-32P672. The software must specify the location of these data structures by writing base addresses into the appropriate FREEDM-32P672 registers before activating the FREEDM-32P672.

The FREEDM-32P672 accesses RAM directly using physical addressing whereas the software may use virtual addressing. In systems which use virtual memory management, the software must translate between virtual addresses (i.e. - pointers) and physical addresses. The software must ensure that the values written to FREEDM-32P672 registers are physical addresses rather than virtual addresses. In systems that do not use virtual addressing, or in systems where virtual addresses are identical to physical addresses, no address translation is required.

The Data Buffers are written with receive data by the FREEDM-32P672, or contain transmit data which is read by the FREEDM-32P672. The descriptor tables and the queues are required to manage these buffers.

The Normal Mode Register space is accessed by the software running on the embedded processor to manage and control operation of a FREEDM-32P672 device. This register space is located in the FREEDM-32P672 and is mapped into the PCI address space by the software.

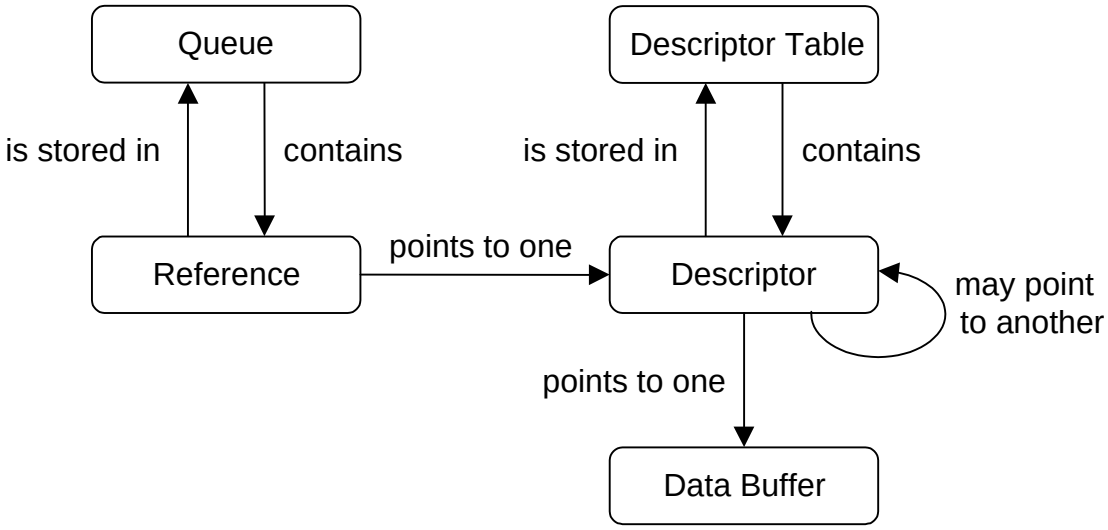
The PCI Configuration Space does not reside in the PCI address map, but it is a requirement for all PCI devices. The Configuration Space is a block of 256 contiguous bytes that reside in the PCI device, and is accessed by the embedded processor in a PCI bus Configuration Read (or Write) transaction, rather than a Memory Read (or Write) transaction. Access to this configuration space is system specific and a thorough discussion of it can be found in the PCI specification[2]. The PCI Configuration Space is discussed in section 6 of this document.

4 DATA STRUCTURES

The RAM data structures accessed by the FREEDM-32P672 are descriptors, descriptor tables, references and queues. The general relationship among them is shown in Figure 3.

In this figure, the direction of the arrows refers to the direction of the relationship. For example, each reference can point to one descriptor. Also, one descriptor may point to another descriptor, thereby specifying a linked-list of descriptors.

Figure 3 – Data Structure Relationships



These data structures are also accessed by software. The queues specify the data which may be accessed by the FREEDM-32P672 or the software, but not both simultaneously.

A Receive Packet consists of a reference pointing to one receive packet descriptor (RPD), or a linked-list of RPDs. A Transmit Packet consists of a reference pointing to one transmit descriptor (TD), or a linked-list of TDs. Transmit Packets may be linked by software, or by the FREEDM-32P672, via separate fields within each descriptor.

4.1 Descriptor Table

The descriptor table is essentially an array, whereby each element of the array is a descriptor and an index to the array is a reference.

A descriptor table holds descriptors of the same kind. The two descriptor tables are the Receive Packet Descriptor Table for receive packets, and the Transmit Descriptor Table for transmit packets.

Allocating a Descriptor Table

A descriptor table can be located anywhere within a 32-bit address space and must be aligned on a 16 byte boundary. The memory allocation must specify a fixed memory address space that cannot be swapped or moved by the operating system.

The size of a descriptor table is specified by the software during initialization. The number of references associated with a FREEDM-32P672 determines the size of the descriptor table. The relationship is: Size (in bytes) = 16* Number of References.

The table index (reference) is a 15-bit value which limits the size to 32,768 descriptors, or 524,288 bytes. The minimum size of the descriptor table depends on the number of channels provisioned. For a descriptor table where each packet is represented by one descriptor, the number of references must be at least 3 times the number of channels provisioned. If the number of descriptors used to represent a packet is greater than one, then the number of references must increase in proportion.

The following FREEDM-32P672 registers must be written with the physical address of the Receive Descriptor Table and the Transmit Descriptor Table, respectively:

Bit	Register
RPDTB[15:0]	**RMAC Packet Descriptor Table Base LSW** (0x288)
RPDTB[31:16]	**RMAC Packet Descriptor Table Base MSW** (0x28C)
TDTB[15:0]	**TMAC Transmit Descriptor Table Base LSW** (0x308)
TDTB[31:16]	**TMAC Transmit Descriptor Table Base MSW** (0x30C)

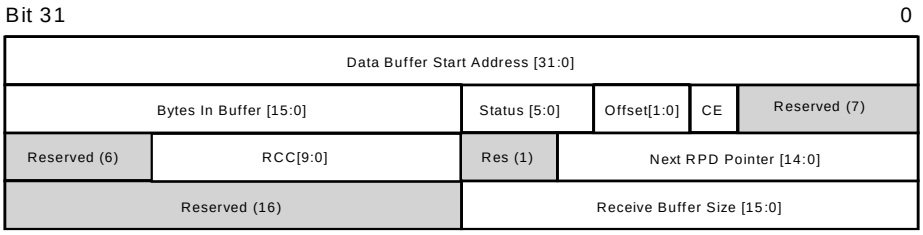
Note: RPDTB[3:0] and TDTB[3:0] must be zero, in order to align the descriptor tables on 16 byte boundaries.

4.2 Receive Packet Descriptor

A Receive Packet Descriptor (RPD) is a 16 byte data structure that contains a number of fields as shown in Figure 4. RPDs are used in the receive direction to describe packets that are received and written to packet memory. Each RPD is

located in the Receive Packet Descriptor Table and is indexed from the base address using a RPD Reference (RPDR).

Figure 4 – Receive Packet Descriptor



The following table describes the individual fields within each RPD:

Field	Description
Data Buffer Start Address[31:0]	The Data Buffer Start Address[31:0] bits point to the data buffer in host memory. This field is expected to be configured by the Host during initialisation. The Data Buffer Start Address field is valid in all RPDs.
CE	The Chain End (CE) bit indicates the end of a linked list of RPDs. When CE is set to logic one, the current RPD is the last RPD of a linked list of RPDs. When CE is set to logic zero, the current RPD is not the last RPD of a linked list. The CE bit is valid for all RPDs written by the RMAC672 to the Receive Ready Queue. When a packet requires only one RPD, the CE bit is set to logic one. The CE bit is ignored for all RPDs read by the RMAC672 from the Receive Free Queues, each of which is assumed to point to only one buffer, i.e. not a chain.
Offset[1:0]	The Offset[1:0] bits indicate the byte offset of the data packet from the start of the buffer. If this value is non-zero, there will be 'dummy' (i.e. undefined) bytes at the start of the data buffer prior to the packet data proper. For a linked list of RPDs, only the first RPD's Offset field is valid. All other RPD Offset fields of the linked list are set to 0.

Field	Description
Status [5:0]	The Status[5:0] bits indicate the status of the received packet. Status[0] Rx buffer overrun Status[1] Packet exceeds max. allowed size Status[2] CRC error Status[3] Packet Length not an exact no. of bytes Status[4] HDLC abort detected Status[5] Unused (set to 0) For a linked list of RPDs, only the last RPD's Status field is valid. All other RPD Status fields of the linked list are invalid and should be ignored. When a packet requires only one RPD, the Status field is valid.
Bytes in Buffer [15:0]	The Bytes in Buffer[15:0] bits indicate the number of bytes actually used in the current RPD's data buffer to store packet data. The count excludes the 'dummy' bytes inserted as a result of a non-zero Offset field. A count greater than 32767 bytes indicates a packet that is shorter than the expected length of the FCS field. The Bytes in Buffer field is invalid when Status[0] or Status[4] is asserted .
Next RPD Pointer [14:0]	The Next RPD Pointer[14:0] bits store a RPDR which enables the RMAC672 to support linked lists of RPDs. This field, which is only valid when CE is equal to logic zero, contains the RPDR to the next RPD in a linked list. The RMAC672 links RPDs when more than one buffer is needed to store a packet. The Next RPD Pointer is not valid for the last RPD in a linked list (when CE=1). When a packet requires only one RPD, the Next RPD Pointer field is not valid.
RCC[9:0]	The Receive Channel Code (RCC[9:0]) bits are used by the RMAC672 to associate a RPD with a channel. For a linked list of RPDs, all the RPDs' RCC[9:0] fields are valid. i.e. all contain the same channel value.

Field	Description
Receive Buffer Size [15:0]	The Receive Buffer Size[15:0] bits indicate the size in bytes of the current RPD's data buffer. This field is expected to be configured by the Host during initialisation. The Receive Buffer Size must be a non-zero integer multiple of sixteen and less than or equal to 32752. The Receive Buffer Size field is valid in all RPDs.

Notes:

- For error checking purposes it is recommended to examine the Bytes in Buffer[15:0] field to ensure that it does not exceed the Receive Buffer Size[15:0].
- Please see Appendix A for the differences in the RPD between the FREEDM-32P672 and the FREEDM-32.

Receive Packet Descriptor Fields Initialized By Software

The following fields of each RPD must be assigned before writing its reference to the RPDRF Large queue, or to the RPDRF Small queue:

Field	Value
Data Buffer Start Address	value is determined during run time or preconfigured
Receive Buffer Size	value is determined during run time or preconfigured

Receive Packet Descriptor Fields Modified By FREEDM-32P672

The following fields are modified by the FREEDM-32P672 after it reads the reference from the RPDRF Large queue, or from the RPDRF Small queue, but before the same reference is written to the RPDR Ready queue:

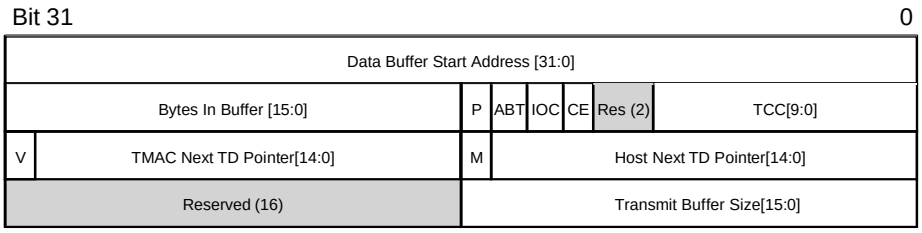
Field	Value
CE	value is determined during run time
Offset	value is determined during run time
Status	value is determined during run time
Bytes in Buffer	value is determined during run time

Field	Value
Next RPD Pointer	value is determined during run time
RCC	value is determined during run time

4.3 Transmit Descriptor

A Transmit Descriptor (TD) is a 16 byte data structure that contains a number of fields as shown in Figure 5. TDs are used in the transmit direction to describe packets that are read from packet memory and transmitted by the FREEDM-32P672. Each TD is located in the Transmit Descriptor Table and is indexed from the base address using a TD Reference (TDR).

Figure 5 – Transmit Descriptor



The following table describes the individual fields within each TD:

Field	Description
Data Buffer Start Address [31:0]	The Data Buffer Start Address[31:0] bits point to the data buffer in host memory. The Data Buffer Start Address field is valid in all TD's.
Bytes In Buffer [15:0]	The Bytes In Buffer[15:0] field is used by the host to indicate the total number of bytes to be transmitted in the current TD. Zero length buffers are illegal.
P	The Priority bit is set by the host to indicate the priority of the associated packet in a two level quality of service scheme. Packets with its P bit set high are queued in the high priority queue in the TMAC672. Packets with the P bit set low are queued in the low priority queue. Packets in the low priority queue will not begin transmission until the high priority queue is empty.

Field	Description
ABT	The Abort (ABT) bit is used by the host to abort the transmission of a packet. When ABT is set to logic 1, the packet will be aborted after all the data in the buffer has been transmitted. If ABT is set to logic 1 in the current TD, the M bit must be set low and the CE bit must be set to high.
IOC	The Interrupt On Complete (IOC) bit is used by the host to instruct the TMAC672 to interrupt the host when the current TD's data buffer has been read. When IOC is logic 1, the TMAC672 asserts the IOCI interrupt when the data buffer has been read. Additionally, the Free Queue FIFO will be flushed. If IOC is logic zero, the TMAC672 will not generate an interrupt and the Free Queue FIFO will operate normally.
CE	The Chain End (CE) bit is used by the host to indicate the end of a linked list of TDs presented to the TMAC672. The linked list can contain one or more packets as delineated by the M bit (see below). When CE is set to logic 1, the current TD is the last TD of a linked list of TDs. When CE is set to logic 0, the current TD is not the last TD of a linked list. When the current TD is not the last of the linked list, the Host Next TD Pointer[14:0] field is valid, otherwise the field is not valid. Note: When CE is set to logic 1, the only valid value for M is logic 0. Note: When presenting raw (i.e. unpacketised) data for transmission, the host should code the M and CE bits as for a single packet chain, i.e. M=1, CE=0 for all TDs except the last in the chain and M=0, CE=1 for the last TD in the chain.
TCC[9:0]	The Transmit Channel Code (TCC[9:0]) bits are used by the host to associate a channel with a TD pointed to by a TDR. All TCC[9:0] fields in a linked list of TDs must be set to the same value.

Field	Description
V	The V bit is used to indicate that the TMAC Next TD Pointer field is valid. When set to logic 1, the TMAC Next TD Pointer[14:0] field is valid. When V is set to logic 0, the TMAC Next TD Pointer[14:0] field is invalid. The V bit is used by the host to reclaim data buffers in the event that data presented to the TMAC672 is returned to the host due to a channel becoming unprovisioned. The V bit is expected to be initialised to logic 0 by the host.
TMAC Next TD Pointer [14:0]	The TMAC Next TD Pointer[14:0] bits are used to store TDRs which permits the TMAC672 to create linked lists of TDs passed to it via the TDRR queue. The TDs are linked with other TDs belonging to the same channel and same priority level. In the case that data presented to the TMAC672 is returned to the host due to a channel becoming unprovisioned, a TDR pointing to the start of the per-channel linked list of TDs is placed on the TDRF queue. It is the responsibility of the host to follow the TMAC672 and host links in order to recover all the buffers.
M	The More (M) bit is used by the host to support packets that require multiple TDs. If M is set to logic 1, the current TD is just one of several TDs for the current packet. If M is set to logic 0, this TD either describes the entire packet (in the single TD packet case) or describes the end of a packet (in the multiple TD packet case). Note: When M is set to logic 1, the only valid value for CE is logic 0.
Host Next TD Pointer [14:0]	The Host Next TD Pointer[14:0] bits are used to store TDRs which permits the host to support linked lists of TDs. As described above, linked lists of TDs are terminated by setting the CE bit to logic 1. Linked lists of TDs are used by the host to pass multiple TD packets or multiple packets associated with the same channel and priority level to the TMAC672.
Transmit Buffer Size [15:0]	The Transmit Buffer Size[15:0] field is used to indicate the size in bytes of the current TD's data buffer. (N.B. The TMAC672 does not make use of this field.)

Note: Please see Appendix B for the differences in the TD between the FREEDM-32P672 and the FREEDM-32.

Transmit Descriptor Fields Initialized By Software

The following fields of a TD (or a linked-list of TDs) must be assigned before writing its reference to the TDR Ready queue:

Field	Value
Data Buffer Start Address	value is determined during run time or preconfigured
Bytes In Buffer	value is determined during run time or preconfigured
P	value is determined during run time or preconfigured
ABT	value is determined during run time or preconfigured
IOC	value is determined during run time or preconfigured
CE	value is determined during run time or preconfigured
TCC	value is determined during run time or preconfigured
V	0
M	value is determined during run time or preconfigured
Host Next TD Pointer	value is determined during run time or preconfigured

Transmit Descriptor Fields Modified By FREEDM-32P672

The following fields may be modified by the FREEDM-32P672 after it reads the reference from the TDR Ready queue, but before the same reference is written to the TDR Free queue:

Field	Value
V	value is determined during run time
TMAC Next TD Pointer	value is determined during run time

4.4 Data Buffers

In the receive path, the FREEDM-32P672 writes receive packet data into data buffers. In the transmit path, the FREEDM-32P672 reads transmit packet data from data buffers. A buffer must be allocated and assigned to each descriptor by the software.

Allocation of Data Buffers

Buffers must be allocated in fixed memory. The receive data buffer size must be a non-zero integer multiple of 16 bytes, with a maximum size of 32,752 bytes and a minimum size of 16 bytes. There is no restriction for the address alignment of the buffers.

For a receive buffer, the following fields of a RPD must be assigned:

Field	Value
Data Buffer Start Address	value is determined during run time or preconfigured
Receive Buffer Size	value is determined during run time or preconfigured

For a transmit buffer, the following fields of a TD must be assigned:

Field	Value
Data Buffer Start Address	value is determined during run time or preconfigured
Bytes In Buffer	value is determined during run time or preconfigured

The FREEDM-32P672 automatically links RPDs when the receive packet length exceeds the buffer size.

The software must link TDs when the packet data is "scattered" among a number of buffers.

4.5 References

References are dword structures used to access descriptors within a descriptor table. They also have status bits which are written by the FREEDM-32P672 after it has processed the packet. The reference, including status bits, is written into a queue by the FREEDM-32P672 during a queue write operation. The status bits indicate the success of receive or transmit processing and should be checked by software when the reference is read from the queue.

4.5.1 Receive Packet Descriptor Reference

Each Receive Packet Descriptor Reference (RPDR) Ready Queue element is 32 bits in size, but only the least significant 17 bits are valid. The 17 least significant bits consist of a 15-bit RPDR and 2 status bits for the RPD pointed at by this RPDR. A RPDR has the following fields:

Bit 31 17	16 15	14 0
UNUSED	STATUS[1:0]	RPDR[14:0]

Field	Description
STATUS[1:0]	The encoding for the status field is as follows: 00 – Successful reception of packet. 01 – Unsuccessful reception of packet. 10 – Unprovisioned partial packet. 11 – Partial packet returned due to RAWMAX limit being reached.
RPDR[14:0]	The RPDR[14:0] field defines the offset of the first RPD in a linked chain of RPDs, each pointing to a buffer containing the received data.

When the RMAC672 writes a STATUS+RPDR to the RPDR Ready queue, it sets bits [23:17] of the queue element to all 0's and leaves bits [31:24] unmodified as follows:

Bit 31 24	23 17	16 15	14 0
UNMODIFIED	000 0000B	STATUS[1:0]	RPDR[14:0]

This may be useful to software which polls host memory to determine when a reference has been written into a queue, instead of responding to an interrupt and reading a FREEDM-32P672 register. The software should write a non-zero value to bits [23:17] after reading the reference, and at a later time it can check whether the non-zero value was overwritten by the FREEDM-32P672, indicating that the FREEDM-32P672 has written another reference into this queue location.

Note: Only one RPDR is written into the RPDR Ready queue per receive packet, and this RPDR represents the linked list of RPDs which identify the receive packet.

4.5.2 Transmit Descriptor Reference

Each Transmit Descriptor Reference (TDR) Free Queue element is 32 bits in size, but only the least significant 18 bits are valid. The 18 least significant bits consist of a 15-bit TDR and 3 status bits for the TD pointed at by this TDR. A TDR has the following fields:

Bit 31 18	17 15	14 0
UNUSED	STATUS[2:0]	TDR[14:0]

Field	Description
Status[2:0]	The TMAC672 fills in the Status field to indicate to the host the results of processing the TD. The encoding is: Status[1:0] Description 00 Last or only buffer of packet, buffer read. 01 Buffer of partial packet, buffer read. 10 Unprovisioned channel, buffer not read. 11 Malformed packet (e.g. Bytes In Buffer field set to 0), buffer not read. Status[2] Description 0 No underflow detected. 1 Underflow detected.
TDR[14:0]	The TDR[14:0] field contains the offset of the TD returned.

When the TMAC672 writes a STATUS+TDR into the TDR Free queue, it sets bits [23:18] of the queue element to all 0's and leaves bits [31:24] unmodified as follows:

Bit 31 24	23 18	17 15	14 0
UNMODIFIED	00 0000B	STATUS[2:0]	TDR[14:0]

This may be useful to software which polls host memory to determine when a reference has been written into a queue, instead of responding to an interrupt and reading a FREEDM-32P672 register. The software should write a non-zero value to bits [23:18] after reading the reference, and at a later time it can check whether the non-zero value was overwritten by the FREEDM-32P672, indicating that the FREEDM-32P672 has written another reference into this queue location.

Notes:

- The TDR associated with each TD of a transmit packet is written to the TDR Free queue. In the case of a packet with multiple TDs there will be multiple TDRs written to the TDR Free queue.
- The Status[2] field of a TDR can be used to identify the occurrence of an underflow condition on the channel associated with the TDR. The underflow may or may not have occurred on the buffer associated with the TDR read from the TDR Free queue.

4.5.3 Access to Descriptors

TDs or RPDs can be accessed using the index field of the reference and the base address of the descriptor table as illustrated by the pseudo code below:

```
/* Need to mask out the upper 17 bits of the descriptor reference to
 * extract the index field. */
#define      RPD_INDEX_MASK    0x00007FFF
#define      TD_INDEX_MASK     RPD_INDEX_MASK
#define      MUL_16_BYTES      4

index = RxReference & RPD_INDEX_MASK;

/* The address of the descriptor in the descriptor table
 * can be determined as shown below */
desc_addr =  desc_table_base_addr + (index << MUL_16_BYTES);
```

4.6 Queues

A queue is a FIFO buffer located in fixed memory that holds a number of references. The FREEDM-32P672 has 5 queues which must be allocated. There are 2 queues for TDRs and 3 queues for RPDRs. The software must allocate memory for each of these queues.

In the receive direction, there is the Receive Packet Descriptor Reference Free Small queue (RPDR Free Small queue), the Receive Packet Descriptor Reference Free Large queue (RPDR Free Large queue), and the Receive Packet

Descriptor Reference Ready queue (RPDR Ready queue). The FREEDM-32P672 reads from the RPDR Free Small queue and the RPDR Free Large queue to get free buffers into which the receive data is written. When the receive operation is complete, the FREEDM-32P672 writes a RPDR to the RPDR Ready queue. The software reads from the RPDR Ready queue to process a receive packet, and it writes to the RPDR Free Small (or Large) queue to reuse the RPD for another packet.

The FREEDM-32P672 obtains free buffers from the RPDR Free Small (or Large) queue based on the following 2-step algorithm:

1. The first buffer into which the receive packet is written is obtained from the RPDR Free Small queue, and if this queue is empty it is obtained from the RPDR Free Large queue.

2. If the receive packet length exceeds the small buffer size then the additional receive data is written into buffers obtained from the RPDR Free Large queue. If the RPDR Free Large queue is empty then the additional buffers are obtained from the RPDR Free Small queue.

In the transmit direction, there is the Transmit Descriptor Reference Ready queue (TDR Ready queue) and the Transmit Descriptor Reference Free queue (TDR Free queue). The software writes a TDR to the TDR Ready queue when it wants the FREEDM-32P672 to transmit a packet. The FREEDM-32P672 reads from the TDR ready queue and starts to transmit the packet, and when it has completed the transmit operation, it writes the TDR to the TDR Free queue. The software reads from the TDR Free queue to confirm that the packet has been transmitted, and to reuse the TD for another packet.

The entity (either the software or the FREEDM-32P672) which reads from a queue and the entity which writes to a queue is as follows:

Queue	**Read By**	**Written By**
RPDR Free Large	FREEDM-32P672	Software
RPDR Free Small	FREEDM-32P672	Software
RPDR Ready	Software	FREEDM-32P672
TDR Free	Software	FREEDM-32P672
TDR Ready	FREEDM-32P672	Software

There are four indexes for each queue that are used to manage its state. These indexes are located in the FREEDM-32P672 Normal Mode Register space. The values are described as follows:

Index	Description
start	The start index marks the lowest address of the queue. This is the first location in the queue. This value should not be modified after initialization.
write	The write index is modified by the entity which writes to the queue. The write index marks the address where a reference can be written. After the reference is written this value is incremented.
read	The read index is modified by the entity which reads from the queue. The read index marks the last location accessed by the reading entity. After the reference is read this value is incremented.
end	The end index marks the address which follows the last location (the highest addressable location) in the queue. This value should not be modified after initialization.

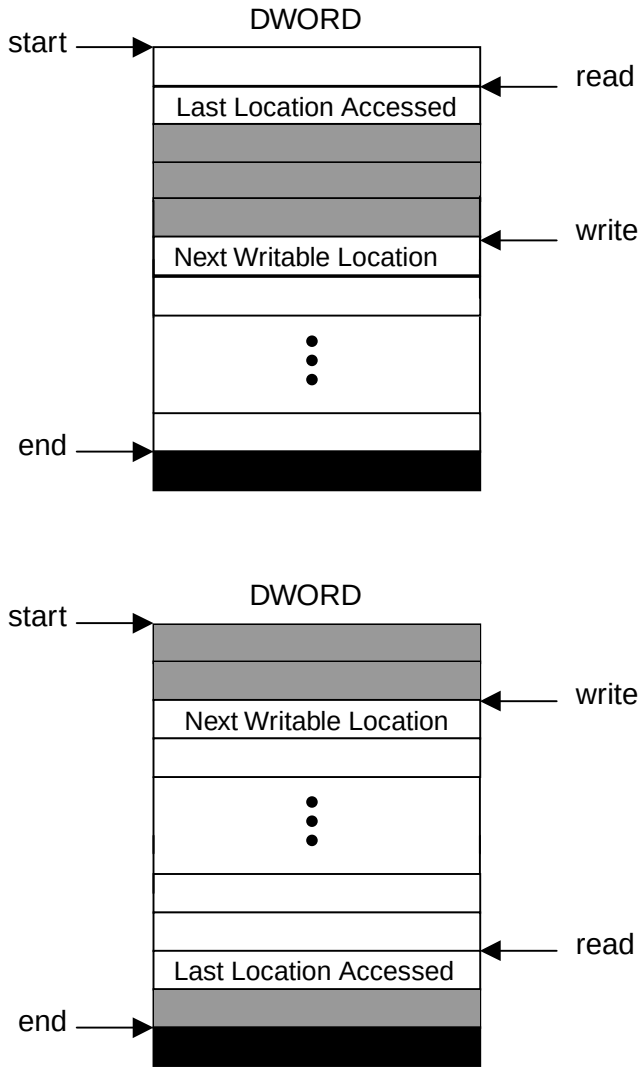
Note: The end index points to a location that is beyond the queue; a reference can not be read from or written to this location. However, the start index of one queue can be set to the end index of another queue.

The various queue entities (references) in Figures 6 to 8 are illustrated using the following legend:

	Empty Reference Location
	Valid Reference Location
	Invalid Reference Location for this Queue

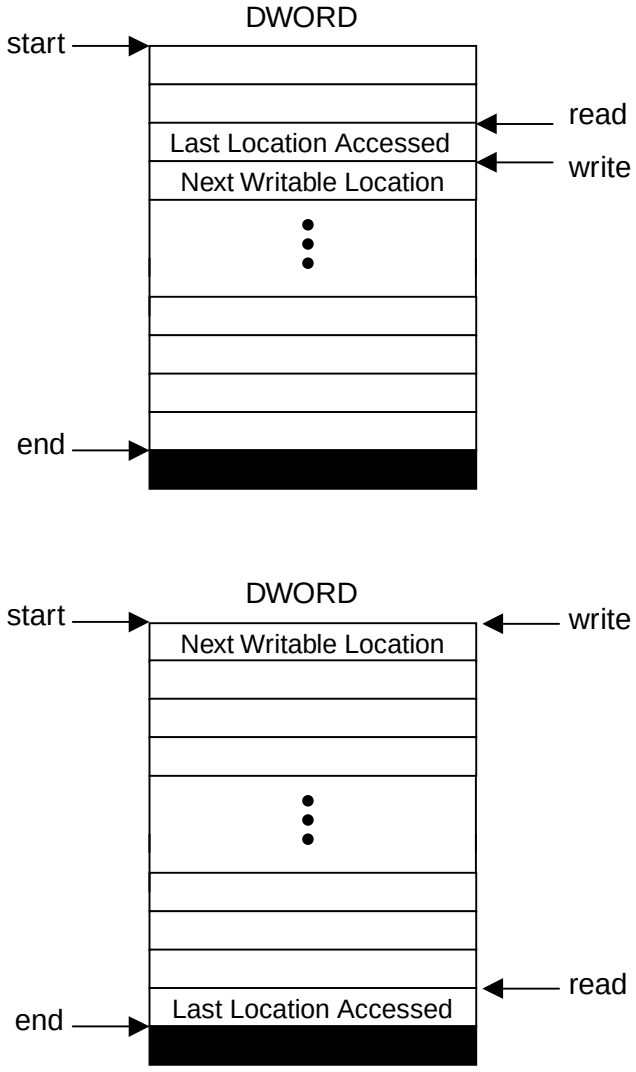
Some normal queue states are illustrated in Figure 6. Note the circular nature of the queues.

Figure 6 – Normal Queue States



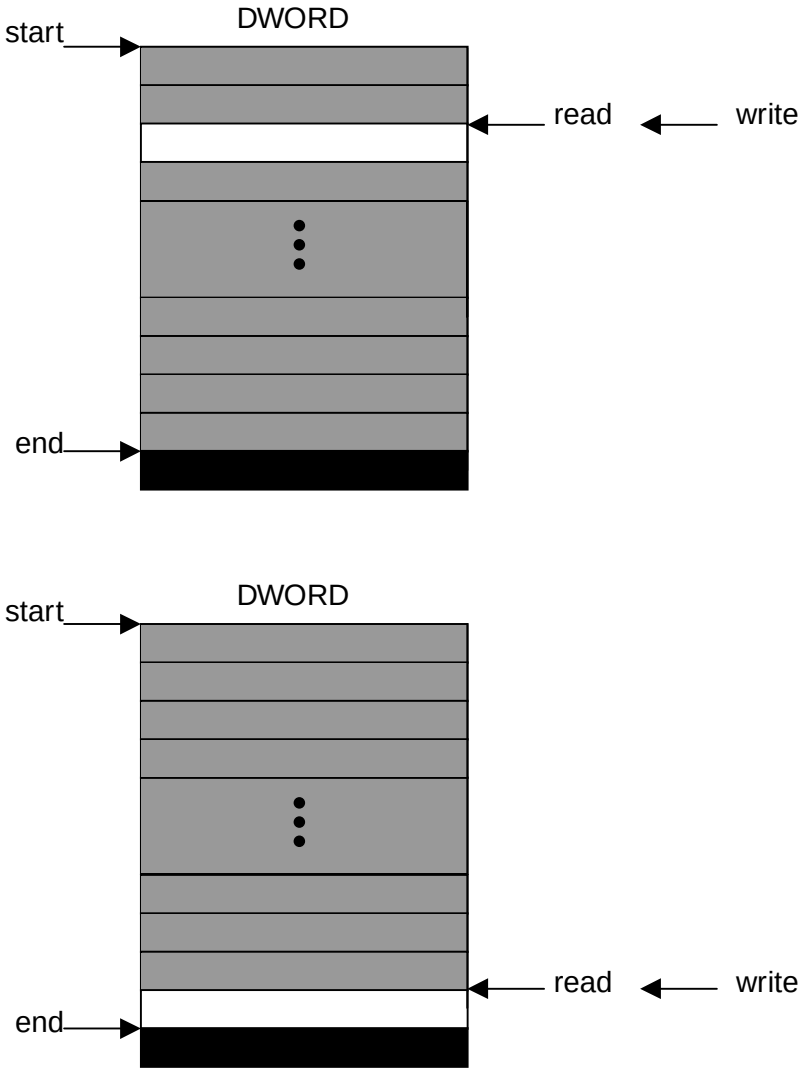
The empty queue states are illustrated in Figure 7. The queue is empty when the read index is one location before the write index, or when the read index is one location before the end index and the write index equals the start index.

Figure 7 – Empty Queue States



The full queue states are illustrated in Figure 8. The queue is full when the read index is equal to the write index.

Figure 8 – Full Queue States



Allocation of Queues

From Figure 8, it can be seen that the physical size of a queue is one dword larger than the number of references in the queue when it is full. Therefore in order to create a queue that holds 128 references, the software must allocate contiguous memory of 129 dwords.

To obtain the best possible bus utilization, the size of a queue should not be too small, as this would lead to more frequent accesses to the read and/or write index registers of the FREEDM-32P672. The minimum recommended queue size is approximately 32 references. In general, the queue should be large enough to hold one reference per provisioned channel.

The queues used for receive packets are located in fixed memory as offsets from a base address. The queues used for transmit packets are located in fixed memory as offsets from another base address. Base addresses must be dword aligned and are programmed as follows for the receive direction and transmit direction, respectively:

Bits	Register
RQB[15:0]	**RMAC Receive Queue Base LSW** (0x290)
RQB[31:16]	**RMAC Receive Queue Base MSW** (0x294)
TQB[15:0]	**TMAC Transmit Queue Base LSW** (0x310)
TQB[31:16]	**TMAC Transmit Queue Base MSW** (0x314)

The RPDR Free Large queue, the RPDR Free Small queue and the RPDR Ready queue must reside within 256Kbytes of the RMAC672 Receive Queue Base address. The size of each queue is specified by assignment of the start, write, read and end indexes.

The TDR Ready queue and the TDR Free queue must reside within 256Kbytes of the TMAC672 Transmit Queue Base address. The size of each queue is specified by assignment of the start, write, read and end indexes.

Initialization of Queues

The software must initialize each queue after the allocation procedure. Normally, a queue is initialized with the state shown below:

Queue	Initial State
RPDR Free Large	Full
RPDR Free Small	Full
RPDR Ready	Empty
TDR Free	Empty
TDR Ready	Empty

The software must write valid RPD References into the RPDRF Small (and Large) queues. The software may force the RMAC672 to store received data in buffers of only one size by setting one of the receive free queues to zero length (i.e. – by setting the start and end index registers of one of the queues to equal values).

The software must also write the following FREEDM-32P672 registers with valid indexes:

Bits	Register
RPDRLFQS[15:0]	**RMAC Packet Descriptor Reference Large Buffer Free Queue Start** (0x298)
RPDRLFQW[15:0]	**RMAC Packet Descriptor Reference Large Buffer Free Queue Write** (0x29C)
RPDRLFQR[15:0]	**RMAC Packet Descriptor Reference Large Buffer Free Queue Read** (0x2A0)
RPDRLFQE[15:0]	**RMAC Packet Descriptor Reference Large Buffer Free Queue End** (0x2A4)
RPDRSFQS[15:0]	**RMAC Packet Descriptor Reference Small Buffer Free Queue Start** (0x2A8)
RPDRSFQW[15:0]	**RMAC Packet Descriptor Reference Small Buffer Free Queue Write** (0x2AC)
RPDRSFQR[15:0]	**RMAC Packet Descriptor Reference Small Buffer Free Queue Read** (0x2B0)
RPDRSFQE[15:0]	**RMAC Packet Descriptor Reference Small Buffer Free Queue End** (0x2B4)
RPDRRQS[15:0]	**RMAC Packet Descriptor Reference Ready Queue Start** (0x2B8)
RPDRRQW[15:0]	**RMAC Packet Descriptor Reference Ready Queue Write** (0x2BC)
RPDRRQR15:0]	**RMAC Packet Descriptor Reference Ready Queue Read** (0x2C0)
RPDRRQE[15:0]	**RMAC Packet Descriptor Reference Ready Queue End** (0x2C4)
TDRFQS[15:0]	**TMAC Transmit Descriptor Reference Free Queue Start** (0x318)

Bits	Register
TDRFQW[15:0]	**TMAC Transmit Descriptor Reference Free Queue Write** (0x31C)
TDRFQR[15:0]	**TMAC Transmit Descriptor Reference Free Queue Read** (0x320)
TDRFQE[15:0]	**TMAC Transmit Descriptor Reference Free Queue End** (0x324)
TDRRQS[15:0]	**TMAC Transmit Descriptor Reference Ready Queue Start** (0x328)
TDRRQW[15:0]	**TMAC Transmit Descriptor Reference Ready Queue Write** (0x32C)
TDRRQR[15:0]	**TMAC Transmit Descriptor Reference Ready Queue Read** (0x330)
TDRRQE[15:0]	**TMAC Transmit Descriptor Reference Ready Queue End** (0x334)

Queue Operation

The following code illustrates how the software can access a queue. It should be noted for a specific queue that the software will only read from it or write to it, but not both read and write to it.

```
#define QUEUE_BATCH_SIZE                          6

#define READ_INDEX_REGISTER(address)              ((*address)&0xFFFF)
#define WRITE_INDEX_REGISTER(address,value)       *address = (dword) value

BOOL ReadQueue(dword* pReference)
{
    dword* pQueueElement;

    /* The following code segment ensures the write index register
     * is not read too frequently. Thereby, minimizing
     * utilization of the PCI bus. */

    if (Headroom == 0) {
       /* Headroom was initialized to zero, and must be reinitialized
        * to a non-zero value in the following code segment before
        * the software is able to read a reference from the queue.
        * The Headroom is the number of references in the queue when the
        * write index was last read by software, minus the number of
```

```
            * these references that have been read. */
           Write = READ_INDEX_REGISTER(pWriteRegister);
           if (Write <= Read)
                 Headroom = Write - Start + End - Read - 1;
           else
                 Headroom = Write - Read - 1;
           /* Exit if the queue is empty */
           if (Headroom == 0) return FALSE;
      }
      Headroom--;

      /* Determine the read index of the reference in the queue.
       * Reading is a pre-increment operation. */
      Read++;
      if (Read == End)
         Read = Start;

      /* Read the reference from a RAM location */
      pQueueElement = pQueueBaseAddress + Read;
      *pReference = *pQueueElement;

      /* The following code segment ensures the read index register
       * is not written too frequently. Thereby, minimizing
       * utilization of the PCI bus. */

      if (CacheSize-- == 0) {
         WRITE_INDEX_REGISTER(pReadRegister, Read);
         CacheSize = QUEUE_BATCH_SIZE;
      }

      return TRUE;
}

BOOL WriteQueue(dword Reference)
{
      dword* pQueueElement;

      /* The following code segment ensures the read index register
       * is not read too frequently. Thereby, minimizing
       * utilization of the PCI bus. */

      if (Headroom == 0) {
         /* Headroom was initialized to zero, and must be reinitialized
          * to a non-zero value in the following code segment before
          * the software is able to write a reference from the queue.
          * The Headroom is the free space in the queue when the
          * read index was last read by software, minus the number of
          * these locations that have been written. */
```

```
            Read = READ_INDEX_REGISTER(pReadRegister);
            if (Read < Write)
                  Headroom = Read - Start + End - Write;
            else
                  Headroom = Read - Write;
            /* Exit if the queue is full */
            if (Headroom == 0) return FALSE;
         }
         Headroom--;

         /* Write the reference to a RAM location */
         pQueueElement = pQueueBaseAddress + Write;
         *pQueueElement = Reference;

         /* Update the write index for next time.
          * Write is a post-increment operation */
         Write++;
         if (Write == End)
            Write = Start;

         /* The following code segment ensures the write index register
          * is not written too frequently. Thereby, minimizing
          * utilization of the PCI bus. */

         if (CacheSize-- == 0) {
            WRITE_INDEX_REGISTER(pWriteRegister, Write);
            CacheSize = QUEUE_BATCH_SIZE;
         }

         return TRUE;
      }
```

An alternative method of reading from a queue is to poll a queue location in RAM, waiting for the FREEDM-32P672 to write a reference to the queue. This method is recommended when interrupts RPQRDYI and TDQFI are disabled, and processing of the RPDR Ready queue and the TDR Free queue must take place by polling. The following code illustrates this method.

```
#define INVALID_REFERENCE      0xFFFFFFFF

/* this routine assumes all empty queue locations were initialized
 * with the value 0xFFFFFFFF */

BOOL PollQueue(dword* pReference)
{
    dword* pQueueElement;

    /* Read the reference from a RAM location */
```

```
        pQueueElement = pQueueBaseAddress + NextReadLocation;
        *pReference = *pQueueElement;

        if (*pReference == INVALID_REFERENCE) {
           /* the queue location was not overwritten by the FREEDM-32P672, so
            * the reference is invalid, and PollQueue() does not return
            * a valid reference. */
           return FALSE;
        }
        else {
           /* the queue location was overwritten by the FREEDM-32P672, so
            * the reference is valid, proceed by overwriting the queue
            * location with an invalid reference. */
           *pQueueElement = INVALID_REFERENCE;

           /* write the FREEDM-32P672 register every n'th packet */
           if (CacheSize++ == QUEUE_BATCH_SIZE) {
                 WRITE_INDEX_REGISTER(pReadRegister, NextReadLocation);
                 CacheSize = 0;
           }

           /* calculate next read index since
            * read is a pre-increment operation */
           NextReadLocation++;
           if (NextReadLocation == End) {
                 NextReadLocation = Start;
           }
           return TRUE;
        }
     }
```

5 INTERRUPT ARCHITECTURE

This section provides an overview of the FREEDM-32P672 interrupt architecture. Detailed information on the individual interrupts is available in the Longform Datasheet[1].

The FREEDM-32P672 provides a number of individual interrupts which are identified as 'I' bits within the **FREEDM-32P672 Master Interrupt Status** (0x008) register. When an interrupt source becomes active, the 'I' bit is set and remains set until the **FREEDM-32P672 Master Interrupt Status** (0x008) register is read.

The FREEDM-32P672 provides interrupts to the PCI bus via the PCIINTB pin of the FREEDM-32P672. This signal is typically routed to an embedded processor via the INTA#, INTB#, INTC# or INTD# pin on the PCI bus. The PCIINTB pin is gated by the **FREEDM-32P672 Master Interrupt Enable** (0x004) register. This register contains 'E' bits which can mask the 'I' bit from causing an interrupt on the PCIINTB pin of the FREEDM-32P672. When the 'E' and 'I' bits of an interrupt source are both high, then the PCIINTB pin is active. When the 'E' bit is low, the interrupt source will not activate the PCIINTB pin, regardless of the 'I' bit status. However, the 'I' bit remains valid when interrupts are disabled and may be polled to detect the various events.

The complete list of 'I' bits and 'E' bits is shown below:

'E' Bit	'I' Bit	Description
SERRE	SERRI	System Error
PERRE	PERRI	Parity Error
RFCSEE	RFCSEI	Receive FCS Error
RABRTE	RABRTI	Receive Abort
RPFEE	RPFEI	Receive Packet Format Error
RFOVRE	RFOVRI	Receive FIFO Overrun Error
RPQSFE	RPQSFI	Small Buffer Cache Read
RPQLFE	RPQLFI	Large Buffer Cache Read
RPQRDYE	RPQRDYI	RPQR Ready Queue Write
RPDFQEE	RPDFQEI	RPDR Free Queue Error
RPDRQEE	RPDRQEI	RPDR Ready Queue Error
TDQFE	TDQFI	TDR Free Queue Write
TDQRDYE	TDQRDYI	TDR Ready Queue Read
TDFQEE	TDFQEI	TDR Free Queue Error
IOCE	IOCI	Interrupt On Complete
TFUDRE	TFUDRI	Transmit FIFO Underflow Error

Interrupt Service Routine

The following code segment illustrates how interrupts for transmit and receive packets can be processed:

```
#define      RPQSFI                      0x0040
#define      RPQLFI                      0x0080
#define      RPQRDYI                     0x0100
#define      TDQFI                       0x0800
#define      IOCI                        0x4000
#define      RX_FREE_INTERRUPT           RPQLFI & RPQSFI
#define      TX_RX_INTERRUPT             TDQFI & IOCI & RPQRDYI
#define      READ_REGISTER(address)      ((*address)&0xFFFF)

/* read and clear the interrupt status */
Status == READ_REGISTER(pFreedmMasterInterruptStatusRegister);

if (Status&(TX_RX_INTERRUPT|RX_FREE_INTERRUPT)) {

    /* disable interrupts scheduled for deferred processing */
    Enable = READ_REGISTER(pFreedmMasterInterruptEnableRegister);

    /* disable active TX_RX_INTERRUPT bits */
    Enable &= ~TX_RX_INTERRUPT;
    WRITE_REGISTER(pFreedmMasterInterruptEnableRegister, Enable);

    /* Schedule processing of these interrupts within a
     * deferred processing routine. The deferred processing routine
     * should run after interrupt service routine, and with a lower
     * priority than the interrupt service routine. The deferred
     * processing routine must enable the relevant 'E' bits when it
     * is done with processing of Status values. */
    ScheduleDPR(Status);
}
```

Notes:

- The pseudo code shows how interrupt status bits are processed to pass control over to routines that do transmit and receive interrupt processing. The actual processing of receive packets and transmit packets must be interleaved to ensure that the host software does not continuously service transmit packets while there are receive packets waiting to be serviced. This could lead to a receive FIFO overrun or a transmit FIFO underflow. By interleaving processing of the TD Free queue and the RPD Ready queue, the

user can ensure that either queue will never be full, and that queue processing latencies are balanced among the transmit and receive paths.

- The pseudo code does not show how to process "critical error interrupts"; the list of these is shown in the table above. These interrupts must be processed in a manner analogous to the TX_RX_INTERRUPT bits shown in the pseudo code.

6 PCI CONFIGURATION SPACE

The purpose of the PCI Configuration Space is to provide device specific information in a common template such that software can identify each PCI device in the system, determine the individual functions provided by each device and allocate system resources to each device. The software must also write bits to enable the FREEDM-32P672 to respond as a target to a PCI host master transaction. Please see section 10.1 for specific operational procedures and register values that must be modified.

6.1 Accessing the PCI Configuration Space

The FREEDM-32P672 responds to Type 0 configuration cycles for a single function device, as described in the PCI specification[2]. The FREEDM-32P672 only uses the IDSEL pin and the AD[1:0] = 00B to determine whether to respond to a configuration cycle. During the address phase of the configuration cycle the AD[7:2] pins specify which of the 64 DWORD aligned Configuration Space registers is accessed. During the subsequent data phases, the BE#[3:0] pins specify which byte lanes within the 32-bit data bus are accessed.

The method of generating the configuration cycle is described in the PCI specification for a PC-AT compatible architecture, but for other system architectures, the method of generating configuration accesses is not defined in the PCI specification. The designer of the system must provide a mechanism that allows PCI configuration cycles to be generated by software. The designer must also specify an API to read and/or write registers within the Configuration Space.

6.2 PCI Configuration Registers

Portions of the PCI Configuration Space are mandatory in order for a PCI device to be in full compliance with the PCI specification. This section identifies the registers which are implemented in the FREEDM-32P672. The reader is referred to the PCI specification[2] and the Longform Datasheet[1] for an in-depth description of these registers.

The mandatory fields are listed below and shown in bold text in Figure 9.

- Vendor ID
- Device ID
- Command

- PCI Status
- Revision ID
- Class Code
- Header Type

Implementation of the other registers in a Type 0 Configuration Space is optional. Fields marked with asterisks (*) are not implemented in the FREEDM-32P672 Configuration Space. These fields will return 0 when read.

Figure 9 – FREEDM-32P672 Type 0 Configuration Space Header

DWORD Register	Address	Byte 3	Byte 2	Byte 1	Byte 0
1	0x00	**Device ID**		**Vendor ID**	
2	0x04	**Status**		**Command**	
3	0x08	**Class Code**			**Revision ID**
4	0x0C	BIST*	**Header Type**	Latency Timer	Cache Line Size
5	0x10	Base Address 0 (CBI Memory Address)			
6	0x14	Base Address 1*			
7	0x18	Base Address 2*			
8	0x1C	Base Address 3*			
9	0x20	Base Address 4*			
10	0x24	Base Address 5*			
11	0x28	Cardbus CIS Pointer*			
12	0x2C	Subsystem ID*		Subsystem Vendor ID*	
13	0x30	Expansion ROM Base Address*			
14	0x34	Reserved*			
15	0x38	Reserved*			
16	0x3C	Max_Lat	Min_Gnt	Interrupt Pin	Interrupt Line

7 CONFIGURING THE SERIAL LINKS

Each of the 32-bidirectional links is controlled via the RCAS672 and the TCAS672 blocks of the FREEDM-32P672. The RCAS672 controls the receive data stream while the TCAS672 controls the transmit data stream.

The RCAS672 extracts data bits from each of 32 receive links, based on the link configuration specified in the Normal Mode Register Space. The RCAS672 can align data to a link's gapped clock in channelised mode, extract unaligned data in unchannelised mode, or extract byte synchronized data in unchannelised mode.

The TCAS672 sources data bits onto each of 32 transmit links, based on the link configuration specified in the Normal Mode Register Space. The TCAS672 can provide data aligned to a link's gapped clock in channelised mode, provide unaligned data in unchannelised mode, or provide byte synchronized data in unchannelised mode.

Each of the serial link configurations is discussed separately in the following sections.

7.1 Channelised T1/J1 Links

The receive bit stream is input on RD[n], and the RCLK[n] input is a 1.544 MHz clock that provides bit timing. The clock is gapped to align time-slot 1 of the channelised T1/J1 receive link (see Figure 10).

The transmit bit stream is output on TD[n], and the TCLK[n] input is a 1.544 MHz clock that provides bit timing. The clock is gapped to align time-slot 1 of the channelised T1/J1 transmit link (see Figure 11).

In each direction, transmit or receive, a T1/J1 frame consists of 24 time-slots or bytes which are mapped to one or more channels of the FREEDM-32P672. The data stream of each direction is processed and clocked independently.

Figure 10 – Channelised T1/J1 Receive Link Timing

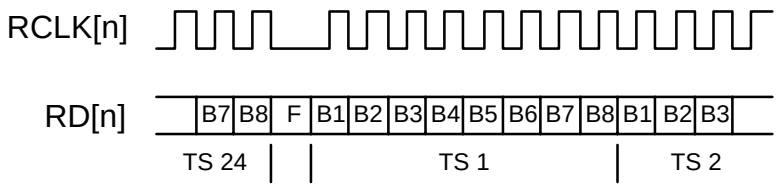
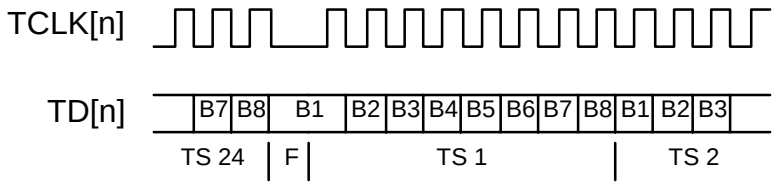


Figure 11 – Channelised T1/J1 Transmit Link Timing



To configure the RCAS672 and TCAS672 to interface to channelised T1/J1 links, the following bits are written:

Bit	Register	Value
MODE[2:0]	**RCAS Link #n Configuration** (0x180 – 0x1FC)	001
BSYNC	**RCAS Link #n Configuration** (0x180 – 0x188)	X
FTHRES[6:0]	**RCAS Framing Bit Threshold** (0x108)	0x25
MODE[2:0]	**TCAS Link #n Configuration** (0x480 – 0x4FC)	001
BSYNC	**TCAS Link #n Configuration** (0x480 – 0x488)	X
FTHRES[6:0]	**TCAS Framing Bit Threshold** (0x408)	0x25
FDATA[7:0]	**TCAS Idle Time-slot Fill Data** (0x40C)	0xFF

Notes:

- The **RCAS Link #n Configuration** register must be chosen from the range 0x180 through 0x1FC corresponding to the link being configured for channelised T1/J1 operation. The BSYNC bit is present only in registers with $0 \leq n \leq 2$. The BSYNC value is ignored for the channelised mode of operation.

- The **TCAS Link #n Configuration** register must be chosen from the range 0x480 through 0x4FC corresponding to the link being configured for channelised T1/J1 operation. The BSYNC bit is present only in registers with $0 \leq n \leq 2$. The BSYNC value is ignored for the channelised mode of operation.

- The FTHRES[6:0] value assumes a SYSCLK of 40Mhz. For other values of SYSCLK the framing threshold value is $1.5 * f_{SYSCLK} / 1.544\ MHz$. (This value ensures that the threshold is suitable for T1/J1 and E1 links).

- The FDATA[7:0] bits of the **TCAS Idle Time-slot Fill Data** (0x40C) register, and the FTHRES[6:0] bits of the **RCAS Framing Bit Threshold** (0x108) / **TCAS Framing Bit Threshold** (0x408) registers, affect all links of a

FREEDM-32P672. The programmer should ensure that these values are suitable for all links attached to a FREEDM-32P672.

7.2 Channelised E1 Links

The receive bit stream is input on RD[n], and the RCLK[n] input is a 2.048 MHz clock that provides bit timing. The clock is gapped to align time-slot 1 of the channelised E1 receive link (see Figure 12).

The transmit bit stream is output on TD[n], and the TCLK[n] input is a 2.048 MHz clock that provides bit timing. The clock is gapped to align time-slot 1 of the channelised E1 transmit link (see Figure 13).

In each direction, transmit or receive, an E1 frame consists of 31 time-slots or bytes which are mapped to one or more channels of the FREEDM-32P672. The data stream of each direction is processed and clocked independently.

Figure 12 – Channelised E1 Receive Link Timing

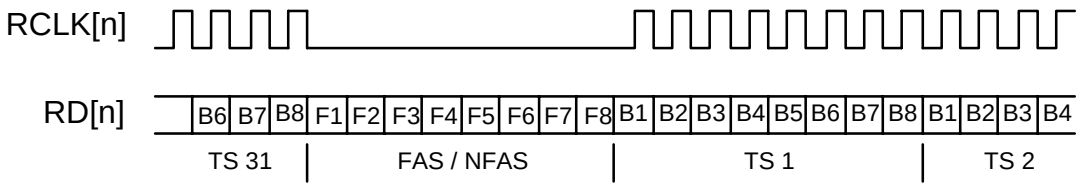
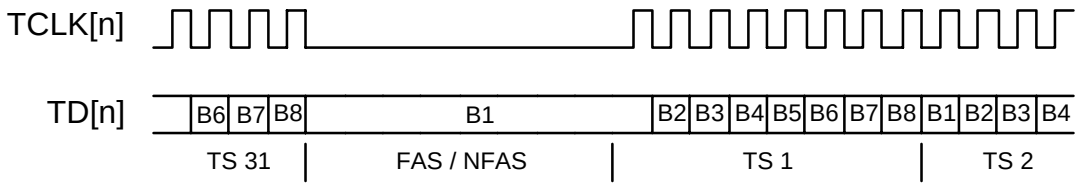


Figure 13 – Channelised E1 Transmit Link Timing



To configure the RCAS672 and TCAS672 to interface to channelised E1 links, the following bits are written:

Bit	Register	Value
MODE[2:0]	**RCAS Link #n Configuration** (0x180 – 0x1FC)	010
BSYNC	**RCAS Link #n Configuration** (0x180 – 0x188)	X
FTHRES[6:0]	**RCAS Framing Bit Threshold** (0x108)	0x25

Bit	Register	Value
MODE[2:0]	**TCAS Link #n Configuration** (0x480 – 0x4FC)	010
BSYNC	**TCAS Link #n Configuration** (0x480 – 0x488)	X
FTHRES[6:0]	**TCAS Framing Bit Threshold** (0x408)	0x25
FDATA[7:0]	**TCAS Idle Time-slot Fill Data** (0x40C)	0xFF

Notes:

- The **RCAS Link #n Configuration** register must be chosen from the range 0x180 through 0x1FC corresponding to the link being configured for channelised E1 operation. The BSYNC bit is present only in registers with $0 \leq n \leq 2$. The BSYNC value is ignored for the channelised mode of operation.

- The **TCAS Link #n Configuration** register must be chosen from the range 0x480 through 0x4FC corresponding to the link being configured for channelised E1 operation. The BSYNC bit is present only in registers with $0 \leq n \leq 2$. The BSYNC value is ignored for the channelised mode of operation.

- Since channelised E1 links have a framing byte, its gap will always be longer than the gap for the T1/J1 framing bit. Therefore, the same FTHRES[6:0] value of 0x25 can be used. The FTHRES[6:0] value assumes a SYSCLK of 40Mhz. For other values of SYSCLK the framing threshold value is $1.5 * f_{SYSCLK} / 1.544\ MHz$. (This value ensures the threshold is suitable for T1/J1 and E1 links).

- The FDATA[7:0] bits of the **TCAS Idle Time-slot Fill Data** (0x40C) register, and the FTHRES[6:0] bits of the **RCAS Framing Bit Threshold** (0x108) / **TCAS Framing Bit Threshold** (0x408) registers, affect all links of a FREEDM-32P672. The programmer should ensure that these values are suitable for all links attached to a FREEDM-32P672.

7.3 Unchannelised Links with Byte Synchronization

The bit stream of the unchannelised receive link that is attached to the RD[n] input is sampled on rising edges of the RCLK[n] input (see Figure 14). The receive data is byte aligned to the gapped RCLK[n] input, with the most significant bit of the byte clocked in following the gap.

The unchannelised transmit link is attached to the TD[n] output and is driven on falling edges of the TCLK[n] input (see Figure 15). The transmit data is byte

aligned to the gapped TCLK[n] input, with the most significant bit of the byte clocked out during the gap.

This mode of operation is only available for links 0 through 2 of a FREEDM-32P672.

In each direction, transmit or receive, the data stream is processed and clocked independently.

Figure 14 – Unchannelised Receive Link Timing with Byte Synchronization

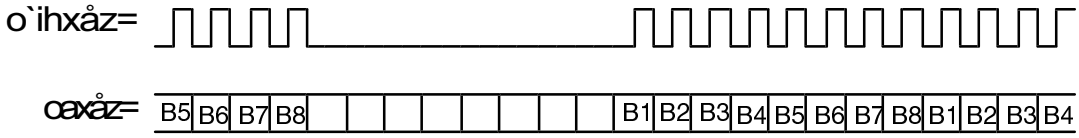
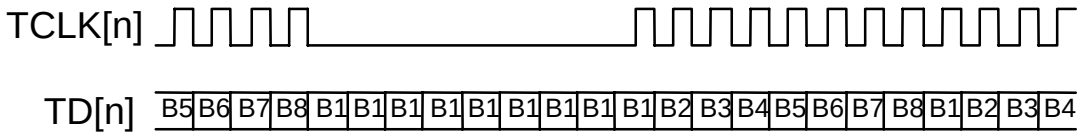


Figure 15 – Unchannelised Transmit Link Timing with Byte Synchronization



To configure the RCAS672 and TCAS672 to interface to byte synchronized unchannelised links, the following bits are written:

Bit	Register	Value
MODE[2:0]	**RCAS Link #n Configuration** (0x180 – 0x1FC)	000
BSYNC	**RCAS Link #n Configuration** (0x180 – 0x188)	1
FTHRES[6:0]	**RCAS Framing Bit Threshold** (0x108)	see note
MODE[2:0]	**TCAS Link #n Configuration** (0x480 – 0x4FC)	000
BSYNC	**TCAS Link #n Configuration** (0x480 – 0x488)	1
FTHRES[6:0]	**TCAS Framing Bit Threshold** (0x408)	see note
FDATA[7:0]	**TCAS Idle Time-slot Fill Data** (0x40C)	0xFF

Notes:

- The **RCAS Link #n Configuration** register must be chosen from the range 0x180 through 0x1FC corresponding to the link being configured. The

unchannelised link with byte synchronization is only supported in registers with $0 \leq n \leq 2$.

- The **TCAS Link #n Configuration** register must be chosen from the range 0x480 through 0x4FC corresponding to the link being configured. The unchannelised link with byte synchronization is only supported in registers with $0 \leq n \leq 2$.

- The FTHRES[6:0] must be set based on the expected gap width of RCLK[n] or TCLK[n]. The reader should refer to the Longform Datasheet[1] on how to set this value.

- The FDATA[7:0] bits of the **TCAS Idle Time-slot Fill Data** (0x40C) register, and the FTHRES[6:0] bits of the **RCAS Framing Bit Threshold** (0x108) / **TCAS Framing Bit Threshold** (0x408) registers, affect all links of a FREEDM-32P672. The programmer should ensure that these values are suitable for all links attached to a FREEDM-32P672.

7.4 Unchannelised Links without Synchronization

The bit stream of the unchannelised receive link that is attached to the RD[n] input is sampled on rising edges of the RCLK[n] input (see Figure 16). The receive data is not aligned to the RCLK[n] input.

The unchannelised transmit link is attached to the TD[n] output and is driven on falling edges of the TCLK[n] input (see Figure 17). The transmit data is not aligned to the TCLK[n] input.

In each direction, transmit or receive, the data stream is processed and clocked independently.

Figure 16 – Unchannelised Receive Link Timing without Synchronization

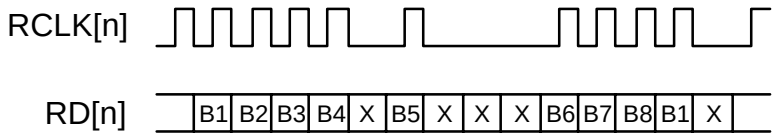
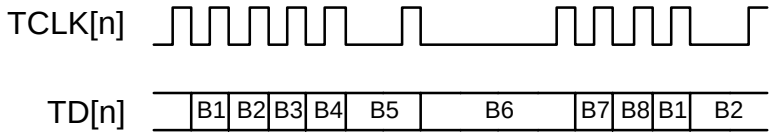


Figure 17 – Unchannelised Transmit Link Timing without Synchronization



To configure the RCAS672 and TCAS672 to interface to unchannelised links without synchronization, the following bits are written:

Bit	Register	Value
MODE[2:0]	**RCAS Link #n Configuration** (0x180 – 0x1FC)	000
BSYNC	**RCAS Link #n Configuration** (0x180 – 0x188)	0
FTHRES[6:0]	**RCAS Framing Bit Threshold** (0x108)	XXH
MODE[2:0]	**TCAS Link #n Configuration** (0x480 – 0x4FC)	000
BSYNC	**TCAS Link #n Configuration** (0x480 – 0x488)	0
FTHRES[6:0]	**TCAS Framing Bit Threshold** (0x408)	XXH
FDATA[7:0]	**TCAS Idle Time-slot Fill Data** (0x40C)	0xFF

Notes:

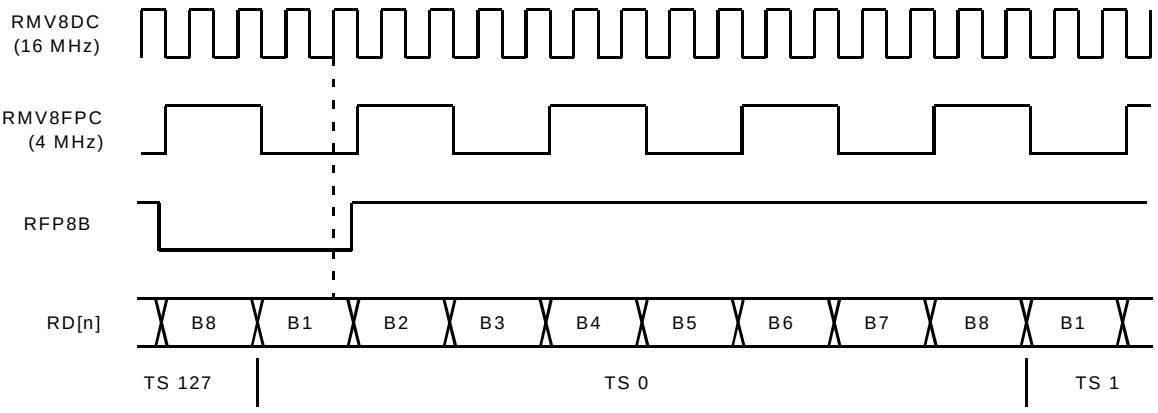
- The **RCAS Link #n Configuration** register must be chosen from the range 0x180 through 0x1FC corresponding to the link being configured for unchannelised operation.
- The **TCAS Link #n Configuration** register must be chosen from the range 0x480 through 0x4FC corresponding to the link being configured for unchannelised operation.
- The BSYNC bit must only be programmed for links where $0 \leq n \leq 2$. For other links, there is no BSYNC bit.
- The FTHRES[6:0] bits of the **RCAS Framing Bit Threshold** (0x108) / **TCAS Framing Bit Threshold** (0x408) registers have no effect on the operation of an unchannelised link with BSYNC low, or on unchannelised links with no BSYNC bit.
- The FDATA[7:0] bits of the **TCAS Idle Time-slot Fill Data** (0x40C) register, and the FTHRES[6:0] bits of the **RCAS Framing Bit Threshold** (0x108) / **TCAS Framing Bit Threshold** (0x408) registers, affect all links of a

FREEDM-32P672. The programmer should ensure that these values are suitable for all links attached to a FREEDM-32P672.

7.5 8.192 Mbps H-MVIP Links

The timing relationship of the receive data clock (RMV8DC), frame pulse clock (RMV8FPC), data (RD[n]) and frame pulse (RFP8B[n]) signals of a link configured for 8.192 Mbps H-MVIP operation with a type 0 frame pulse is shown in Figure 18. The falling edges of each RMV8FPC are aligned to a falling edge of the corresponding RMV8DC for 8.192 Mbps H-MVIP operation. The FREEDM-32P672 samples RFP8B low on the falling edge of RMV8FPC and references this point as the start of the next frame. The FREEDM-32P672 samples the data provided on RD[n] at the ¾ point of the data bit using the rising edge of RMV8DC as indicated for bit 1 (B1) of time-slot 0 (TS 0) in Figure 18. B1 is the most significant bit and B8 is the least significant bit of each octet. Time-slots can be ignored by setting the PROV bit in the corresponding word of the receive channel provision RAM in the RCAS672 block to low.

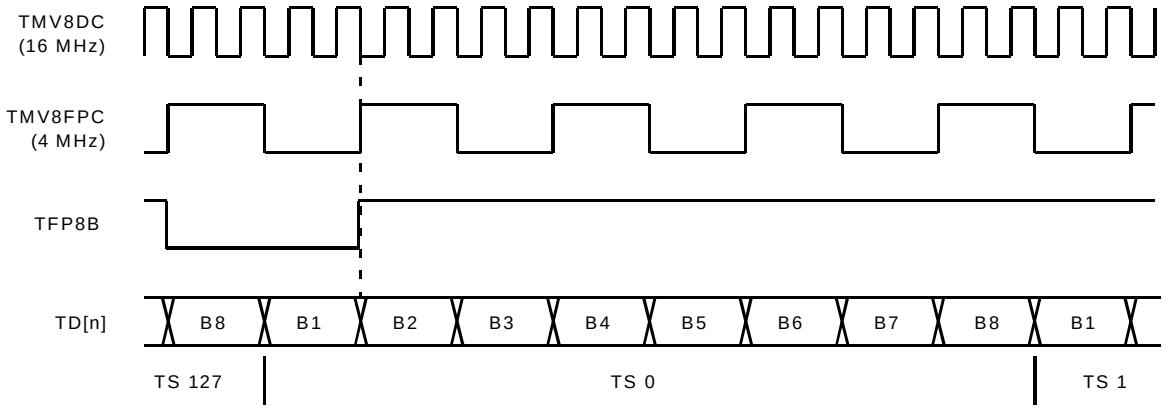
Figure 18 – Receive 8.192 Mbps H-MVIP Link Timing



The timing relationship of the transmit data clock (TMV8DC), frame pulse clock (TMV8FPC), data (TD[n]) and frame pulse (TFP8B) signals of a link configured for 8.192 Mbps H-MVIP operation with a type 0 frame pulse is shown in Figure 19. The falling edges of each TMV8FPC are aligned to a falling edge of the corresponding TMV8DC for 8.192 Mbps H-MVIP operation. The FREEDM-32P672 samples TFP8B low on the falling edge of TMV8FPC and references this point as the start of the next frame. The FREEDM-32P672 updates the data provided on TD[n] on every second falling edge of TMV8DC as indicated for bit 2 (B2) of time-slot 0 (TS 0) in Figure 19. The first bit of the next frame is updated on TD[n] on the falling TMV8DC clock edge for which TFP8B is also sampled low. B1 is the most significant bit and B8 is the least significant bit of each octet.

Time-slots that are not provisioned to belong to any channel (PROV bit in the corresponding word of the transmit channel provision RAM in the TCAS672 block set low) transmits the contents of the **TCAS Idle Time-slot Fill Data** (0x40C) register.

Figure 19 – Transmit 8.192 Mbps H-MVIP Link Timing



To configure the RCAS672 and TCAS672 to interface to 8.192 Mbps H-MVIP links, the following bits are written:

Bit	Register	Value
MODE[2:0]	**RCAS Link #n Configuration** (0x180 – 0x1FC)	111
BSYNC	**RCAS Link #n Configuration** (0x180 – 0x188)	X
FTHRES[6:0]	**RCAS Framing Bit Threshold** (0x108)	XXH
MODE[2:0]	**TCAS Link #n Configuration** (0x480 – 0x4FC)	111
BSYNC	**TCAS Link #n Configuration** (0x480 – 0x488)	X
FTHRES[6:0]	**TCAS Framing Bit Threshold** (0x408)	XXH
FDATA[7:0]	**TCAS Idle Time-slot Fill Data** (0x40C)	0xFF

Notes:

- When link 4m ($0 \le m \le 7$) is configured for operation in 8.192 Mbps H-MVIP mode in the receive direction (MODE[2:0] = ”111”), data cannot be received on links 4m+1, 4m+2 and 4m+3. However, links 4m+1, 4m+2 and 4m+3 must be configured for 8.192 Mbps H-MVIP mode for correct operation of the RCAS672. From a channel assignment point of view in the RCAS672 (Registers 0x100, 0x104), time-slots 0 through 31 of the H-MVIP link are

treated as time-slots 0 through 31 of link 4m, time-slots 32 through 63 of the H-MVIP link are treated as time-slots 0 through 31 of link 4m+1, time-slots 64 through 95 of the H-MVIP link are treated as time-slots 0 through 31 of link 4m+2 and time-slots 96 through 127 of the H-MVIP link are treated as time-slots 0 through 31 of link 4m+3.

- When link 4m (0≤m≤7) is configured for operation in 8.192 Mbps H-MVIP mode in the transmit direction (MODE[2:0] = "111"), links 4m+1, 4m+2 and 4m+3 are driven with constant ones. However, links 4m+1, 4m+2 and 4m+3 must be configured for 8.192 Mbps H-MVIP mode for correct operation of the TCAS672. From a channel assignment point of view in the TCAS672 (Registers 0x40, 0x404), time-slots 0 through 31 of link 4m are mapped to time-slots 0 through 31 of the H-MVIP link, time-slots 0 through 31 of link 4m+1 are mapped to time-slots 32 through 63 of the H-MVIP link, time-slots 0 through 31 of link 4m+2 are mapped to time-slots 64 through 95 of the H-MVIP link and time-slots 0 through 31 of link 4m+3 are mapped to time-slots 96 through 127 of the H-MVIP link.

- The BSYNC value is ignored for links configured for H-MVIP.

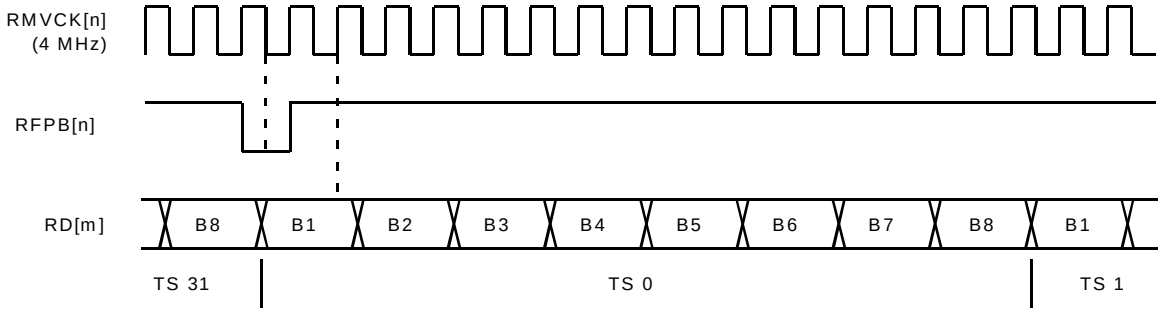
- The FTHRES[6:0] bits of the **RCAS Framing Bit Threshold** (0x108) / **TCAS Framing Bit Threshold** (0x408) registers have no effect on the operation of H-MVIP links.

- The FDATA[7:0] bits of the **TCAS Idle Time-slot Fill Data** (0x40C) register, and the FTHRES[6:0] bits of the **RCAS Framing Bit Threshold** (0x108) / **TCAS Framing Bit Threshold** (0x408) registers, affect all links of a FREEDM-32P672. The programmer should ensure that these values are suitable for all links attached to a FREEDM-32P672.

7.6 2.048 Mbps H-MVIP Links

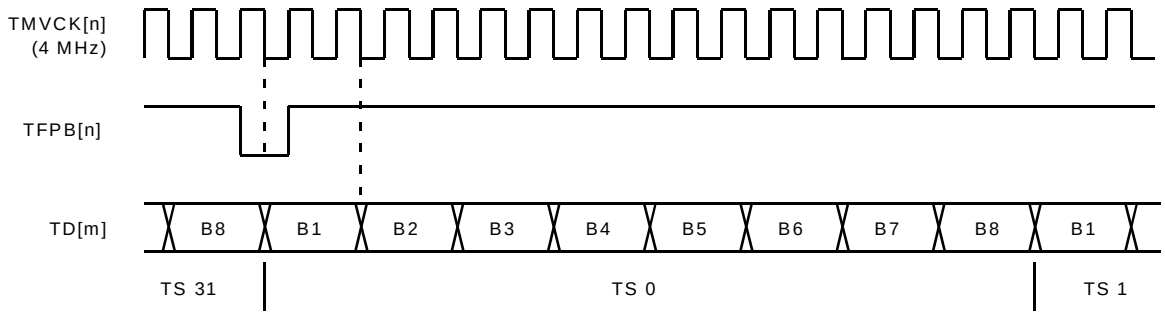
The timing relationship of the receive data clock (RMVCK[n]), data (RD[m], where 8n≤m≤8n+7) and frame pulse (RFPB[n]) signals of a link configured for 2.048 Mbps H-MVIP operation with a type 0 frame pulse is shown in Figure 20. The FREEDM-32P672 samples RFPB[n] low on the falling edge of the corresponding RMVCK[n] and references this point as the start of the next frame. The FREEDM-32P672 samples the data provided on RD[m] at the ¾ point of the data bit using the rising edge of the corresponding RMVCK[n] as indicated for bit 1 (B1) of time-slot 0 (TS 0) in Figure 20. B1 is the most significant bit and B8 is the least significant bit of each octet. Time-slots can be ignored by setting the PROV bit in the corresponding word of the receive channel provision RAM in the RCAS672 block to low.

Figure 20 – Receive 2.048 Mbps H-MVIP Link Timing



The timing relationship of the transmit data clock (TMVCK[n]), data (TD[m], where 8n≤m≤8n+7) and frame pulse (TFPB[n]) signals of a link configured for 2.048 Mbps H-MVIP operation with a type 0 frame pulse is shown in Figure 21. The FREEDM-32P672 samples TFPB[n] low on the falling edge of the corresponding TMVCK[n] and references this point as the start of the next frame. The FREEDM-32P672 updates the data provided on TD[m] on every second falling edge of the corresponding TMVCK[n] as indicated for bit 2 (B2) of time-slot 0 (TS 0) in Figure 21. The first bit of the next frame is updated on TD[n] on the falling TMVCK[n] clock edge for which TFPB[n] is also sampled low. B1 is the most significant bit and B8 is the least significant bit of each octet. Time-slots that are not provisioned to belong to any channel (PROV bit in the corresponding word of the transmit channel provision RAM in the TCAS672 block set low) transmits the contents of the **TCAS Idle Time-slot Fill Data** (0x40C) register.

Figure 21 – Transmit 2.048 Mbps H-MVIP Link Timing



To configure the RCAS672 and TCAS672 to interface to 2.048 Mbps H-MVIP links, the following bits are written:

Bit	Register	Value
MODE[2:0]	**RCAS Link #n Configuration** (0x180 – 0x1FC)	011

Bit	Register	Value
BSYNC	**RCAS Link #n Configuration** (0x180 – 0x188)	X
FTHRES[6:0]	**RCAS Framing Bit Threshold** (0x108)	XXH
MODE[2:0]	**TCAS Link #n Configuration** (0x480 – 0x4FC)	011
BSYNC	**TCAS Link #n Configuration** (0x480 – 0x488)	X
FTHRES[6:0]	**TCAS Framing Bit Threshold** (0x408)	XXH
FDATA[7:0]	**TCAS Idle Time-slot Fill Data** (0x40C)	0xFF

Notes:

- The **RCAS Link #n Configuration** register must be chosen from the range 0x180 through 0x1FC corresponding to the link being configured for 2.048 Mbps H-MVIP operation.
- The **TCAS Link #n Configuration** register must be chosen from the range 0x480 through 0x4FC corresponding to the link being configured for 2.048 Mbps H-MVIP operation.
- The BSYNC value is ignored for links configured for H-MVIP.
- The FTHRES[6:0] bits of the **RCAS Framing Bit Threshold** (0x108) / **TCAS Framing Bit Threshold** (0x408) registers have no effect on the operation of H-MVIP links.
- The FDATA[7:0] bits of the **TCAS Idle Time-slot Fill Data** (0x40C) register, and the FTHRES[6:0] bits of the **RCAS Framing Bit Threshold** (0x108) / **TCAS Framing Bit Threshold** (0x408) registers, affect all links of a FREEDM-32P672. The programmer should ensure that these values are suitable for all links attached to a FREEDM-32P672.

8 CONFIGURING THE PCI INTERFACE

Configuration of the PCI interface involves initialization of data structures, which is covered in section 4, and mapping of the Normal Mode Register Space, which is covered in section 10.1. This section covers configuration and control of the DMA activities. These are accessible within the RMAC672, TMAC672 and GPIC672 block registers.

8.1 Configuring the Receive DMA Controller (RMAC672)

The RMAC672 is the DMA controller which writes receive data into packet memory. It sources data from the RHDL672 and requests the GPIC672 to write the data across the PCI bus and into packet memory.

The RMAC672 is configured by programming bits within the **RMAC Control** (0x280) register. The values programmed affect all receive channels. The default configuration is as follows:

Bit	Register	Value
ENABLE	**RMAC Control** (0x280)	0
LCACHE	**RMAC Control** (0x280)	1
SCACHE	**RMAC Control** (0x280)	1
RAWMAX[1:0]	**RMAC Control** (0x280)	11
RPQ_RDYN[2:0]	**RMAC Control** (0x280)	000
RPQ_LFN[1:0]	**RMAC Control** (0x280)	00
RP1_SFN[1:0]	**RMAC Control** (0x280)	00
Reserved	**RMAC Control** (0x280)	0

The default indicates that the RMAC672 is disabled from DMA'ing receive data into packet memory.

Activation of the RMAC672

By default, the RMAC672 is disabled from DMA'ing receive data into packet memory. The ENABLE bit must be set to allow DMA of receive data into packet memory. The encoding of this bit is:

ENABLE	Function
0	The RMAC672 does not accept data from the RHDL672 and does not write data to host memory
1	The RMAC672 accepts data from the RHDL672 and writes it to host memory.

Free Buffer Cache Enable

The FREEDM-32P672 reads from packet memory to obtain unused receive buffers, and stores them in a cache if caching is enabled. The access can read just one RPDR, or six RPDRs if the cache is enabled. There is a separate cache for the small buffers and the large buffers; they can be individually enabled.

LCACHE (or SCACHE)	Function
0	The RMAC672 reads just one RPDR at a time.
1	The RMAC672 reads up to six RPDRs and stores them in a cache.

Raw Data Notification

The RAWMAX[1:0] field determines notification of receive occurrences. This field only applies to channels that are provisioned with the DELIN bit set low within the **RHDL Indirect Channel Data Register #1** (0x204) register. When the unprocessed data fills RAWMAX[1:0] + 1 buffers, the resulting buffer chain is placed in the RPDR Ready queue.

RPQRDYI, RPQLFI and RPQSFI Interrupt Frequency

The RPQ_RDYN[2:0] field indicates the number of RPDRs written to the RPDR Ready queue by the FREEDM-32P672 before an RPQRDYI interrupt is asserted. It essentially controls the frequency of RPQRDYI interrupts. When this interrupt occurs the software must process the linked list of buffers for each RPDR (packet) that is read from the RPDR Ready queue. Valid values are:

RPQ_RDYN[2:0]	No of RPDRs
000	1
001	4
010	6
011	8
100	16
101	32
110	Reserved
111	Reserved

The RPQ_LFN[1:0] field sets the number of times that a block of RPDRs are read from the Large Buffer Free Queue to the RMAC672's internal cache before the RPDR Large Buffer Free Queue interrupt (RPQLFI) is asserted. It essentially controls the frequency of RPQLFI interrupts. When this interrupt occurs the software must replenish the RPDRF Large queue with large buffers. Valid values are:

RPQ_LFN[1:0]	No of Reads
00	1
01	4
10	8
11	Reserved

The RPQ_SFN[1:0] field sets the number of times that a block of RPDRs are read from the Small Buffer Free Queue to the RMAC672's internal cache before the RPDR Small Buffer Free Queue interrupt (RPQSFI) is asserted. It essentially controls the frequency of RPQSFI interrupts. When this interrupt occurs the software must replenish the RPDRF Small queue with small buffers. Valid values are:

RPQ_SFN[1:0]	No of Reads
00	1
01	4

RPQ_SFN[1:0]	No of Reads
10	8
11	Reserved

8.2 Configuring the Transmit DMA Controller (TMAC672)

The TMAC672 is the DMA controller which reads transmit data from packet memory. It reads TDRs from the TDR Ready queue to determine the transmit buffer data which must be DMA'd across the PCI bus and passed onto the THDL672 block.

The TMAC672 is configured by programming bits within the **TMAC Control** (0x300) register. The values programmed affect all transmit channels. The default configuration is as follows:

Bit	Register	Value
ENABLE	**TMAC Control** (0x300)	0
CACHE	**TMAC Control** (0x300)	1
TDQ_RDYN[2:0]	**TMAC Control** (0x300)	000
TDQ_FRN[1:0]	**TMAC Control** (0x300)	00
FQFLUSH	**TMAC Control** (0x300)	0

The default indicates that the TMAC672 is disabled from DMA'ing transmit data from packet memory.

Activation of the TMAC672

By default, the TMAC672 is disabled from DMA'ing data from packet memory. The ENABLE bit must be set to allow DMA of transmit data. The encoding of this bit is:

ENABLE	Function
0	The TMAC672 does not read the TDR Ready queue in packet memory to transmit new packets. Once all linked lists of TDs built up by the TMAC672 have been exhausted, no more data will be transmitted on the TD[31:0] links.
1	The TMAC672 can read the TDR Ready queue in packet memory to transmit new packets.

Free Buffer Cache Enable

The CACHE enable bit allows the TMAC672 to cache up to six TDRs before writing them to the TDR Free queue.

CACHE	Function
0	The TMAC672 writes one TDR at a time to the TDR Free queue.
1	The TMAC672 caches up to six TDRs and writes them to the TDR Free queue at one time.

TDQRDYI and TDQFI Interrupt Frequency

The TDQ_RDYN[2:0] field indicates the number of TDRs read from the TDR Ready queue by the FREEDM-32P672 before an TDQRDYI interrupt is asserted. It essentially controls the frequency of TDQRDYI interrupts. Valid values are:

TDQ_RDYN[2:0]	No of TDRs
000	1
001	4
010	6
011	8
100	16
101	32
110	Reserved
111	Reserved

The TDQ_FRN[1:0] field sets the number of times that a block of TDRs are written to the TDR Free Queue to the TMAC672's internal cache before the TDR Free Queue interrupt (TDQFI) is asserted. It essentially controls the frequency of TDQFI interrupts. When this interrupt occurs the software must collect each TDR that is read from the TDR Free queue in order to confirm that a transmit packet was transmitted, so that the buffers can be reused. Valid values are:

TDQ_FRN[1:0]	No of Reads
00	1
01	4
10	8
11	Reserved

Free Queue Flush

The Free Queue Flush bit (FQFLUSH) may be used to initiate a dump of the free queue cache retained locally within the TMAC672 to the free queue located in PCI host memory. The FQFLUSH bit is self-clearing and will reset to zero when the flush is complete.

FQ_FLUSH	Function
0	No effect.
1	The TMAC672 dumps the contents of the free queue cache to the free queue in PCI host memory.

8.3 Configuring the General-Purpose PCI Controller (GPIC672)

The GPIC672 provides the interface to a 32-bit PCI bus operating at up to 66 MHz and bridges between the timing domain of the DMA controllers (specified by SYSCLK pin) and the timing domain of the PCI bus (specified by PCICLK pin). All transactions on the PCI bus that are initiated by the RMAC672 or TMAC672 are translated into PCI bus activity by the GPIC672. Except for the PCI Configuration Space registers and parity checking, the GPIC672 does not perform operations on the PCI bus data.

The GPIC672 is configured by programming bits within the **GPIC Control** (0x080) register. The default configuration is as follows:

Bit	Register	Value
Reserved	**GPIC Control** (0x080)	0
LENDIAN	**GPIC Control** (0x080)	1
SOE_E	**GPIC Control** (0x080)	0
PONS_E	**GPIC Control** (0x080)	0
RPWTH[5:0]	**GPIC Control** (0x080)	00 0000B

Little Endian Mode Bit

The LENDIAN bit controls the format of buffer data read from or written to packet memory. By default, the LENDIAN mode bit is set indicating Little Endian format. The Little Endian and Big Endian formats are described in Figures 22 and 23. The encoding of this bit is:

LENDIAN	Function
0	Buffer data is in Big Endian format.
1	Buffer data is in Little Endian format.

Figure 22 – Little Endian Format

		Bit 31 24	23 16	15 8	7 Bit 0
DWORD	00	BYTE 3	BYTE 2	BYTE 1	BYTE 0
Address	04	BYTE 7	BYTE 6	BYTE 5	BYTE 4
		•	•	•	•
		•	•	•	•
		•	•	•	•
	n-4	BYTE n-1	BYTE n-2	BYTE n-3	BYTE n-4

Figure 23 – Big Endian Format

		Bit 31 24	23 16	15 8	7 Bit 0
DWORD	00	BYTE 0	BYTE 1	BYTE 2	BYTE 3
Address	04	BYTE 4	BYTE 5	BYTE 6	BYTE 7
		•	•	•	•
		•	•	•	•
		•	•	•	•
	n-4	BYTE n-4	BYTE n-3	BYTE n-2	BYTE n-1

Notes:

- Since the LENDIAN bit only controls the format of the buffer data, all of the control data structures such as queue elements, descriptors and descriptor references must be in Little Endian format.

- For Big Endian addressing memory, byte swapping is usually done by the host CPU, the PCI bridge or the software.

The SOE_E and PONS_E Bits

The stop on error enable (SOE_E) and the report PERR on SERR enable (PONS_E) are described in the Longform Datasheet[1]. These correspond to faults detected at the hardware level by the PCI bus interface.

Threshold for Early Bus Arbitration

The Receive Packet Write Threshold bits (RPWTH[5:0]) control early arbitration for the PCI bus. For non-zero values of RPWTH[5:0], the GPIC672 will begin requesting access to the PCI bus when the number of dwords of packet data loaded by the RMAC672 reaches the threshold specified by RPWTH[5:0]. When the Receive Packet Write Threshold is set to zero, the GPIC672 will begin requesting access to the PCI bus shortly after data starts to be loaded by the RMAC672. Non-zero values are usually used when SYSCLK runs slower than PCICLK.

9 HDLC AND CHANNEL FIFO CONFIGURATION

The FREEDM-32P672 processes the data stream in the receive direction via the RHDL672 block and it processes the data stream in the transmit direction via the THDL672 block. Each of these blocks must be configured via the Normal Mode Register Space.

9.1 Configuring the RHDL672

The RHDL672 is configured by programming bits within the **RHDL Configuration** (0x220) register and the **RHDL Maximum Packet Length** (0x224) register. The values programmed affect all receive channels. The default configuration is as follows:

Bit	Register	Value
LENCHK	**RHDL Configuration** (0x220)	0
TSTD	**RHDL Configuration** (0x220)	0
MAX[15:0]	**RHDL Maximum Packet Length** (0x224)	0xFFFF

The default indicates no maximum packet length checking and datacom bit ordering.

Maximum Packet Length

The RHDL672 may be configured to abort packets which exceed the maximum length of *n* where $0 \leq n \leq 0xFFFF$. The following bits are written to enable or disable this feature:

LENCHK	MAX[15:0]	Function
0	0xFFFF	Receive packets are not checked for maximum size and MAX[15:0] must be set to 0xFFFF.
1	n	Receive packets with total length, including address, control, information and FCS fields, greater than MAX[15:0] bytes are aborted and the remainder of the frame discarded.

Datacom/Telecom Bit Order

The RHDL672 may be configured to reverse the order of bits within a data byte of a write access on the PCI bus. The following bit is written to specify the order of bits:

TSTD	Function
0	Datacom standard: least significant bit of each byte on the PCI bus (AD[0], AD[8], AD[16], AD[24]) is the first HDLC bit received. Normally, when HDLC processing is enabled, the TSTD bit must be set to zero.
1	Telecom standard: most significant bit of each byte on the PCI bus (AD[7], AD[15], AD[23], AD[31]) is the first HDLC bit received.

9.2 Configuring the THDL672

The THDL672 is configured by programming bits within the **THDL Configuration** (0x3B0) register. The values programmed affect all transmit channels. The default configuration is as follows:

Bit	Register	Value
BURST[3:0]	**THDL Configuration** (0x3B0)	0000B
BURSTEN	**THDL Configuration** (0x3B0)	0
TSTD	**THDL Configuration** (0x3B0)	0
BIT8	**THDL Configuration** (0x3B0)	0

The default indicates PCI DMA transfer size is controlled by XFER[3:0] and data is formatted in datacom bit ordering.

Enabling Burst DMA Transfer

The burst length enable bit (BURSTEN) controls the use of BURST[3:0] in determining the amount of data requested in a single DMA transaction for channels whose channel transfer size is set to one block (XFER[3:0] = 0000B). BURSTEN has no effect on channels configured with other transfer sizes. The following bits are written to enable or disable this feature:

BURSTEN	BURST[3:0]	Function
0	X	The amount of data in a DMA transfer is limited to one block.
1	0 through 15 are valid	The THDL672 may combine several channel transfer size amounts into a single transaction. BURST[3:0] defines the maximum number of 16 byte blocks, less one, that is transferred in each DMA transaction. Thus, the minimum number of blocks is one (16 bytes) and the maximum is sixteen (256 bytes).

Datacom/Telecom Bit Order

The THDL672 may be configured to reverse the order of bits within a data byte of a read access on the PCI bus. The following bit is written to specify the order of bits:

TSTD	Function
0	Datacom standard: least significant bit of each byte on the PCI bus (AD[0], AD[8], AD[16], AD[24]) is the first HDLC bit transmitted. Normally, when HDLC processing is enabled, the TSTD bit must be set to zero.
1	Telecom standard: most significant bit of each byte on the PCI bus (AD[7], AD[15], AD[23], AD[31]) is the first HDLC bit transmitted.

BIT8

The BIT8 field affects channels of the THDL672 that are configured with 7BIT set. The BIT8 value specifies the data bit transmitted on the least significant bit of each octet.

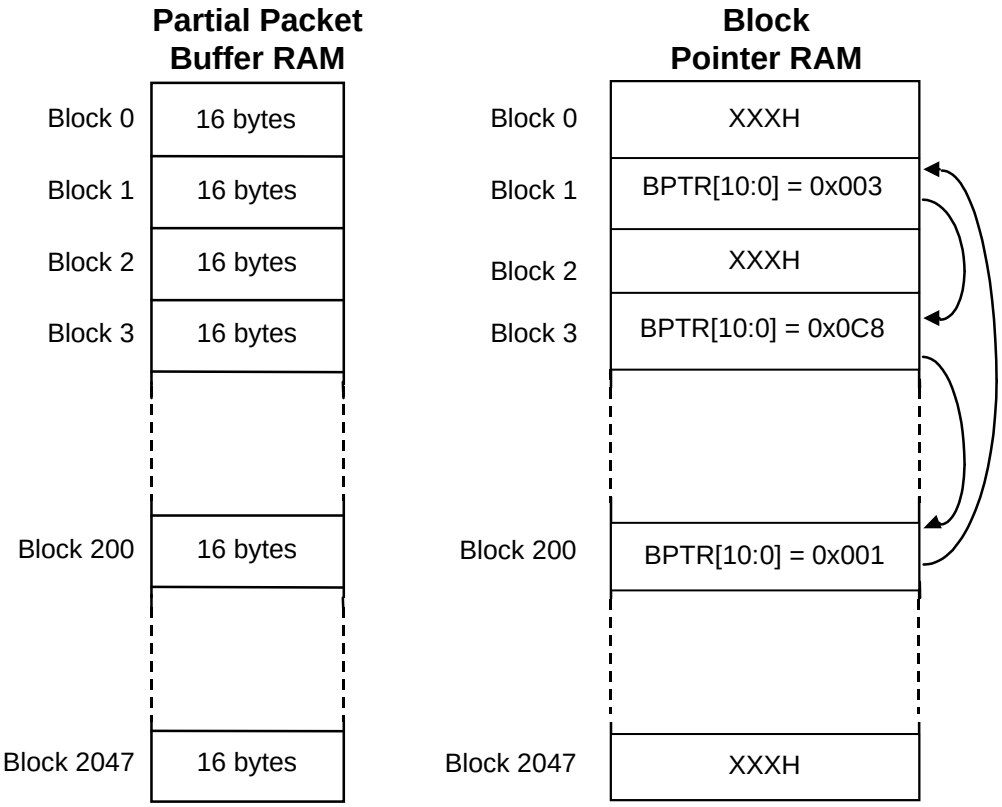
BIT8	Function
0	Channels configured for 7BIT will transmit a zero on the least significant bit of each octet.
1	Channels configured for 7BIT will transmit a one on the least significant bit of each octet.

9.3 Programming a Channel FIFO

A Channel FIFO is created from 3 or more blocks of internal RAM, and each block holds 16 bytes of packet data. There is a total of 2048 blocks (32 Kbytes) available to assign among the receive channels, and another 2048 blocks (32 Kbytes) available to assign among the transmit channels.

A FIFO is created by assigning a circular linked list of blocks as shown in Figure 24. This shows a channel FIFO consisting of 3 blocks. The quantity of buffers and the arrangement of links is chosen by the programmer, and the selection of blocks can be arbitrary. The programmer must ensure that a block is not assigned to more than one circularly linked list.

Figure 24 – Specifying a Channel FIFO



9.3.1 Receive Channel FIFO

A receive channel FIFO is programmed by repeating the following procedure for each block within the circularly linked list:

1. Poll the BUSY bit of the **RHDL Indirect Block Select** (0x210) register until it is zero. This ensures that a previous indirect RAM access has completed and that a new indirect RAM access can be started.

2. Write the following register with the next block in the circular linked list, or exit if all links have been programmed:

Bit	Register	Value
BPTR[10:0]	**RHDL Indirect Block Data** (0x214)	0 through 0x7FF are valid
Reserved	**RHDL Indirect Block Data** (0x214)	0

3. Specify the block and update the internal block pointer RAM by writing the following register. Proceed to step 1.

Bit	Register	Value
BLOCK[10:0]	**RHDL Indirect Block Select** (0x210)	0 through 0x7FF are valid
Reserved	**RHDL Indirect Block Select** (0x210)	0
BRWB	**RHDL Indirect Block Select** (0x210)	0
BUSY	**RHDL Indirect Block Select** (0x210)	X

9.3.2 Transmit Channel FIFO

A transmit channel FIFO is programmed by repeating the following procedure for each block within the circularly linked list:

1. Poll the BUSY bit of the **THDL Indirect Block Select** (0x3A0) register until it is zero. This ensures that a previous indirect RAM access has completed and that a new indirect RAM access can be started.

2. Write the following register with the next block in the circular linked list, or exit if all links have been programmed:

Bit	Register	Value
BPTR[10:0]	**THDL Indirect Block Data** (0x3A4)	0 through 0x7FF are valid

Bit	Register	Value
Reserved[0]	**THDL Indirect Block Data** (0x3A4)	0
Reserved[1]	**THDL Indirect Block Data** (0x3A4)	0

3. Specify the block and update the internal block pointer RAM by writing the following register. Proceed to step 1.

Bit	Register	Value
BLOCK[10:0]	**THDL Indirect Block Select** (0x3A0)	0 through 0x7FF are valid
Reserved	**THDL Indirect Block Select** (0x3A0)	0
BRWB	**THDL Indirect Block Select** (0x3A0)	0
BUSY	**THDL Indirect Block Select** (0x3A0)	X

9.4 RHDL672 Channel Configuration

The RHDL672 provides configurable options for each receive channel as identified in the following register fields:

Bit	Register
DELIN	**RHDL Indirect Channel Data Register #1** (0x204)
STRIP	**RHDL Indirect Channel Data Register #1** (0x204)
XFER[3:0]	**RHDL Indirect Channel Data Register #2** (0x208)
OFFSET[1:0]	**RHDL Indirect Channel Data Register #2** (0x208)
CRC[1:0]	**RHDL Indirect Channel Data Register #2** (0x208)
INVERT	**RHDL Indirect Channel Data Register #2** (0x208)
PRIORITY	**RHDL Indirect Channel Data Register #2** (0x208)
7BIT	**RHDL Indirect Channel Data Register #2** (0x208)

Note: When writing to **RHDL Indirect Channel Data Register #1** (0x204), the reserved bit (bit 11) must be set low for correct operation of the FREEDM-32P672.

Delineation

The data bits from the RCAS672 can be written directly to the Partial Packet Buffer or processed for flag sequence delineation, bit de-stuffing and CRC verification. The following bit enables or disables this feature:

DELIN	Function
0	Data is written to the Partial Packet Buffer without any HDLC processing (no flag sequence delineation, bit de-stuffing nor CRC verification) on the incoming stream.
1	Data is processed for flag sequence delineation, bit de-stuffing and optionally, CRC verification (CRC verification depends on CRC[1:0] value).

Strip FCS Bit

The indirect frame check sequence discard bit (STRIP) enables the RHDL672 to remove the FCS data before writing to the channel FIFO. STRIP is ignored when DELIN is low or when CRC[1:0] = 00B. This feature is configured as follows:

STRIP	Function
0	Includes FCS data with the data stream written to the channel FIFO.
1	Removes the FCS data from the data stream written to the channel FIFO.

DMA Transfer Size

The indirect channel transfer size configures the amount of data transferred in each transaction. When the channel FIFO depth reaches the depth specified by XFER[3:0] or when an end-of-packet exists in the FIFO, a request will be made to the RMAC672 to initiate a PCI write access to transfer the data to the PCI host. During the PCI bus DMA activity, other channels cannot gain access to the bus. Specifying a large transfer size may affect bus access latencies for other channels. The following bits specify the channel transfer size:

XFER[3:0]	Function
0 through 15 are valid	Specifies the data transfer size in blocks: Blocks = XFER[3:0] +1, and there are 16 bytes per block.

Notes:

- XFER[3:0] should be set such that the number of blocks transferred is at least two fewer than the total allocated to the associated channel.

- To ensure optimum PCI bus utilization efficiency, the programmer can choose an XFER size and receive buffer size such that the receive buffer size is an integral multiple of the XFER size. (i.e. - for a buffer size of *n* bytes and an XFER size of *x* bytes, the relationship, $n = i * x$, must be true where $i = 1, 2, 3, ...$). The programmer must choose a value of *n* that is a multiple of 16 for receive buffers, and must convert XFER value from blocks to bytes using the relationship of 16 bytes per block.

Insertion of Offset Bytes

The RHDL672 can be configured to insert offset bytes into the data stream before writing the data stream to the channel FIFO. The offset bytes are placed before each packet and their value is undefined. The following configuration options are available:

OFFSET[1:0]	Function
00	RHDL672 does not insert offset bytes
01	RHDL672 inserts 1 offset byte per packet
10	RHDL672 inserts 2 offset bytes per packet
11	RHDL672 inserts 3 offset bytes per packet

CRC Algorithm

The RHDL672 can perform CRC verification of the incoming data stream. The available options are as follows:

CRC[1:0]	DELIN	Function
X	0	No CRC verification
00	1	No CRC verification
01	1	CRC-CCITT verification
10	1	CRC-32 verification
11	1	Reserved

HDLC Data Inversion

The INVERT bit configures the RHDL672 to logically invert the incoming HDLC stream from the RCAS672 before processing it. The bit is specified as follows:

INVERT	Function
0	HDLC stream is not inverted.
1	HDLC stream is inverted.

Specifying Receive Channel Priority

All receive channels that must transfer data from their channel FIFO to packet memory contend for access to the PCI bus. The PRIORITY bit allows specified channels to have priority access to the PCI bus. The bit encoding is as follows:

PRIORITY	Function
0	This channel is serviced after channels with PRIORITY=1.
1	This channel is serviced before channels with PRIORITY=0.

Handling of Robbed bit Signaling

The 7BIT enable bit configures the RHDL672 to ignore the least significant bit of each octet (last bit of each octet received) in the corresponding link RD[n]. This bit is encoded as follows:

7BIT	Function
0	The entire receive data stream is processed.
1	The least significant bit (last bit of each octet received) is ignored.

9.5 THDL672 Channel Configuration

The THDL672 provides configurable options for each transmit channel as identified in the following register fields:

Bit	Register
DELIN	**THDL Indirect Channel Data Register #1** (0x384)
CRC[1:0]	**THDL Indirect Channel Data Register #1** (0x384)
FLEN[10:0]	**THDL Indirect Channel Data Register #2** (0x388)

Bit	Register
DFCS	**THDL Indirect Channel Data Register #2** (0x388)
INVERT	**THDL Indirect Channel Data Register #2** (0x388)
PRIORITYB	**THDL Indirect Channel Data Register #2** (0x388)
7BIT	**THDL Indirect Channel Data Register #2** (0x388)
XFER[3:0]	**THDL Indirect Channel Data Register #3** (0x38C)
FLAG[2:0]	**THDL Indirect Channel Data Register #3** (0x38C)
LEVEL[3:0]	**THDL Indirect Channel Data Register #3** (0x38C)
IDLE	**THDL Indirect Channel Data Register #3** (0x38C)
TRANS	**THDL Indirect Channel Data Register #3** (0x38C)

Note: When writing to **THDL Indirect Channel Data Register #1** (0x384), the reserved bit (bit 11) must be set low for correct operation of the FREEDM-32P672. When writing to **THDL Indirect Channel Data Register #2** (0x388), the reserved bit (bit 11) must be set low for correct operation of the FREEDM-32P672.

Frame Delineation

The transmit packet data from packet memory can be written directly to the outgoing data stream or processed for flag sequence insertion, bit stuffing and CRC generation. The following bit enables or disables this feature:

DELIN	Function
0	Data is written directly to the outgoing data stream without any HDLC processing (no flag sequence insertion, bit stuffing nor CRC generation).
1	Data is processed for flag sequence insertion, bit stuffing and optionally, CRC generation (CRC generation depends on CRC[1:0] value).

CRC Algorithm

The THDL672 can perform CRC generation on the outgoing data stream. The available options are as follows:

CRC[1:0]	DELIN	Function
X	0	No CRC generation
00	1	No CRC generation
01	1	CRC-CCITT generation
10	1	CRC-32 generation
11	1	Reserved

Channel FIFO Length

The indirect FIFO length (FLEN[10:0]) is the number of blocks, less one, that is provisioned to the circular channel FIFO specified by the FPTR[10:0] block pointer.

FLEN[10:0]	Function
0 through 2047 are valid	Specifies the Channel FIFO size in blocks, where Blocks = FLEN[10:0] + 1, and each block is 16 bytes.

Inverting the FCS

The diagnose frame check sequence bit (DFCS) specifies whether the FCS field inserted into the transmit data stream is inverted. This is provided for diagnostic purposes and is programmed as follows:

DFCS	Function
0	FCS field in the outgoing HDLC stream is not inverted.
1	FCS field in the outgoing HDLC stream is logically inverted.

Specifying Transmit Channel Priority

All transmit buffer data must be read from packet memory, and across the PCI bus. Each transmit channel contends for access to the PCI bus and the PRIORITYB bit allows channels, where the Channel FIFO free space is greater than the starving trigger level and there are no complete packets within the FIFO, to have higher priority access to the PCI bus. The bit encoding is as follows:

PRIORITYB	Function
0	The channel has higher priority access when the Channel FIFO free space is greater than the starving trigger level, and the last byte of the packet has not been placed into the Channel FIFO.
1	The channel is inhibited from making expedited requests for data to the TMAC672. It has lower priority than channels with PRIORITYB=0 when the channel with PRIORITYB=0 has a partial packet in its Channel FIFO and the Channel FIFO is making an expedited data request.

Robbed Bit Signaling

The least significant stuff enable bit (7BIT) configures the THDL672 to stuff the least significant bit of each octet assigned to the transmit channel in the corresponding transmit link (TD[n]).

7BIT	Function
0	The entire octet contains valid data and BIT8 is ignored.
1	The least significant bit (last bit of each octet transmitted) does not contain channel data and is forced to the value configured by the BIT8 register bit.

DMA Transfer Size

The indirect channel transfer size specifies the amount of data that the partial packet processor requests from the TMAC672 block. When the channel FIFO free space reaches or exceeds the limit specified by XFER[3:0], the partial packet processor will make a request for data to the TMAC672 to retrieve the XFER[3:0] + 1 blocks of data. During the PCI bus DMA activity, other channels cannot gain access to the bus. Specifying a large transfer size may affect bus access latencies for other channels. The following bits specify the channel transfer size:

XFER[3:0]	Function
0 through 15 are valid	Specifies the data transfer size in blocks, where Blocks = XFER[3:0] + 1, and each block is 16 bytes.

Notes:

- To prevent lockup, the channel transfer size (XFER[3:0]) can be configured to be less than or equal to the start transmission level set by LEVEL[3:0] and TRANS. Alternatively, the channel transfer size can be set such that the total number of blocks in the logical channel FIFO minus the start transmission level is an integer multiple of the channel transfer size.

- To ensure optimum PCI bus utilization efficiency, the programmer can choose an XFER size and transmit buffer size such that the transmit buffer size is an integral multiple of the XFER size. (i.e. - for a buffer size of *n* bytes and an XFER size of *x* bytes, the relationship, $n = i * x$, must be true where $i = 1, 2, 3, ...$). The programmer must convert XFER value from blocks to bytes using the relationship of 16 bytes per block.

Specifying The Number of Flag or Idle Bytes Inserted Between Frames

The THDL672 can be configured to insert either flag or idle bytes (8 bits of one's) into the data stream between HDLC packets. The number of these is programmed as follows:

FLAG[2:0]	Minimum Number of Flag/Idle Bytes
000	1 flag / 0 Idle byte
001	2 flags / 0 idle byte
010	4 flags / 2 idle bytes
011	8 flags / 6 idle bytes
100	16 flags / 14 idle bytes
101	32 flags / 30 idle bytes
110	64 flags / 62 idle bytes
111	128 flags / 126 idle bytes

Interframe Time Fill

The IDLE bit specifies the byte pattern inserted in the data stream between HDLC packets.

IDLE	Function
0	Flag bytes are inserted between HDLC packets.

IDLE	Function
1	HDLC idle (all one's bit with no bit-stuffing) is inserted between HDLC packets.

Specifying the Channel FIFO's Starving Level and Start Transmit Level

The HDLC processor starts transmitting a packet when the channel FIFO free space is less than or equal to the level specified in the appropriate Start Transmission Level column of the following table or when an end of a packet is stored in the channel FIFO.

When the channel FIFO free space is less than or equal to than the level specified in the Starving Trigger Level column of the following table and the HDLC processor is transmitting a packet and an end of a packet is not stored in the channel FIFO, the partial packet buffer makes expedite requests to the TMAC672 to retrieve XFER[3:0] + 1 blocks of data.

The starving trigger level and start transmission level are programmed via the LEVEL[3:0] and the TRANS field as follows:

LEVEL[3:0]	Starving Trigger Level	Start Transmission Level (TRANS=0)	Start Transmission Level (TRANS=1)
0000	2 Blocks (32 bytes free)	1 Block (16 bytes free)	1 Block (16 bytes free)
0001	3 Blocks (48 bytes free)	2 Blocks (32 bytes free)	1 Block (16 bytes free)
0010	4 Blocks (64 bytes free)	3 Blocks (48 bytes free)	2 Blocks (32 bytes free)
0011	6 Blocks (96 bytes free)	4 Blocks (64 bytes free)	3 Blocks (48 bytes free)
0100	8 Blocks (128 bytes free)	6 Blocks (96 bytes free)	4 Blocks (64 bytes free)
0101	12 Blocks (192 bytes free)	8 Blocks (128 bytes free)	6 Blocks (96 bytes free)
0110	16 Blocks (256 bytes free)	12 Blocks (192 bytes free)	8 Blocks (128 bytes free)
0111	24 Blocks (384 bytes free)	16 Blocks (256 bytes free)	12 Blocks (192 bytes free)

LEVEL[3:0]	Starving Trigger Level	Start Transmission Level (TRANS=0)	Start Transmission Level (TRANS=1)
1000	32 Blocks (512 bytes free)	24 Blocks (384 bytes free)	16 Blocks (256 bytes free)
1001	48 Blocks (768 bytes free)	32 Blocks (512 bytes free)	24 Blocks (384 bytes free)
1010	64 Blocks (1 Kbytes free)	48 Blocks (768 bytes free)	32 Blocks (512 bytes free)
1011	96 Blocks (1.5 Kbytes free)	64 Blocks (1 Kbytes free)	48 Blocks (768 bytes free)
1100	192 Blocks (3 Kbytes free)	128 Blocks (2 Kbytes free)	96 Blocks (1.5 Kbytes free)
1101	384 Blocks (6 Kbytes free)	256 Blocks (4 Kbytes free)	192 Blocks (2 Kbytes free)
1110	768 Blocks (12 Kbytes free)	512 Blocks (8 Kbytes free)	384 Blocks (4 Kbytes free)
1111	1536 Blocks (24 Kbytes free)	1024 Blocks (16 Kbytes free)	768 Blocks (8 Kbytes free)

10 FREEDM-32P672 OPERATIONAL PROCEDURES

10.1 Device Identification, Location and System Resource Assignment

This section describes the software interaction required to identify a FREEDM-32P672 device on the PCI bus, map the Normal Mode Registers in packet memory, and initialize the PCI configuration registers.

Identifying and Locating a FREEDM-32P672

The software can identify each device attached to a PCI bus segment by reading the Device ID and the Vendor ID within the Configuration Space Header. As described in section 6.1 the software must activate the IDSEL pin of a PCI device in order to access the Configuration Space. The IDSEL pin of each PCI device is activated in turn and the first DWORD register is read to identify whether it has the FREEDM-32P672 Device ID (0x7380) and Vendor ID (0x11F8). If a value other than 0xFFFF is read in these fields then a PCI device is present at the IDSEL pin.

In addition to these fields the FREEDM-32P672 specifies a revision identifier within the REVID[7:0] field which may be useful to distinguish between future revisions of the FREEDM-32P672.

Memory Mapping the Register Space

During power-up the packet software needs to build a consistent address map and assign memory resources based on the requirements of each PCI device. The memory requirements are identified via the 6 base address registers in the PCI Configuration Space of each device.

The software writes a base address register with a value of all 1's and reads back the register. The software scans the returned value from the least significant bit upwards to determine the size of memory to assign to the base address register. The binary weighted value of the first one bit found (after the four least significant bits which are used for configuration) indicates the required amount of space. For example, a device that wants a 1M address space would build the top 12 bits and hardwire the others to zero.

For a FREEDM-32P672 device, only the first base address register – the **CBI Memory Base Address Register** (0x10) – is implemented, all other base address registers will be read as a value of all 1's. The first base address

register will return the memory space requirement for the Normal Mode Registers – a memory size of 4Kbytes.

The packet software must assign a base address for this 4Kbyte memory space by writing the **CBI Memory Base Address Register** (0x10) within the PCI Configuration Space with the base address. The value can be read from this register at a later time to determine the mapping of the Normal Mode Register Space.

Enabling the FREEDM-32P672 onto the PCI Bus

Following assignment of the memory base address the software must enable the FREEDM-32P672 to respond to PCI memory accesses and to participate on the PCI bus as a bus master. Additionally, the FREEDM-32P672 can be enabled to report system and parity errors. The **Command** (0x04) register of the PCI Configuration Space is written as follows:

Bit	Word Sized Configuration Register	Value
MCNTRL	**Command** (0x04)	1
MSTREN	**Command** (0x04)	1
PERREN	**Command** (0x04)	1
SERREN	**Command** (0x04)	1

Initializing the PCI Configuration Space Registers

In addition to enabling the FREEDM-32P672 onto the PCI bus, the following register bits must also be initialized:

Bit	Byte Sized Configuration Register	Value
CLSIZE[7:0]	**Cache Line Size** (0x0C)	see note
LT[7:0]	**Latency Timer** (0x0D)	see note
INTLNE[7:0]	**Interrupt Line** (0x3C)	see note

Notes:

- CLSIZE[7:0] should be set equal to the cache line size of the embedded processor. The FREEDM-32P672 uses the memory read multiple command if the data transfer size is greater than the cache line size. It will use the memory read cache line if the data transfer is the same size, or less than the

cache line, but greater than a dword. It uses a memory read if the data transfer size is a single dword.

- LT[7:0] should be set based on the expected initial latency of data transfers on the PCI bus, and on the expected maximum data transaction size specified via the XFER field of the RMAC672 and TMAC672 blocks. The value is specified as a multiple of the PCI bus clock frequency. For a transfer size of 8 blocks and no initial latency, the value should be larger than *(8*4 + 3) = 35.*

- INTLNE[7:0] should be assigned for use by software after power-on. The value is determined based on which input of the system interrupt controller the FREEDM-32P672 interrupt pin is connected to.

10.2 Reset

This section describes the procedure to reset the FREEDM-32P672 via software. The FREEDM-32P672 is powered on in an inactive state and should be reset via software following a hardware reset, or as required by the embedded processor. The reset procedure is normally followed by the initialization procedure.

The steps to reset a FREEDM-32P672 are:

1. If the FREEDM-32P672 was active before the reset procedure then the deactivation procedure must be done (see section 10.5).

2. The RESET bit in the **FREEDM-32P672 Master Reset** (0x000) register must be written high and then written low.

This reset procedure has the following effects:

- The RESET bit allows the FREEDM-32P672 to be reset under software control. If the RESET bit is a logic one, the entire FREEDM-32P672 except the PCI Interface is held in reset. This bit is not self-clearing. Therefore, a logic zero must be written to bring the FREEDM-32P672 out of reset. Holding the FREEDM-32P672 in a reset state places it into a low power, stand-by mode. A hardware reset clears the RESET bit, thus negating the software reset.

- The GPIC672 PCI Configuration register values are preserved under software reset. All Normal Mode registers are set to their default values.

- None of the channel provisioning, or the Channel FIFO configuration is preserved under software reset.

10.3 Initialization

This section describes the procedure to initialize the FREEDM-32P672. This procedure assumes the software has already allocated the data structures in packet memory. A detailed discussion of allocation of data structures can be found in section 4.

The initialization procedure normally follows the software reset procedure and is followed by the activation procedure.

The steps to initialize a FREEDM-32P672 are:

1. Assign base addresses for the Transmit Descriptor Table, the Receive Packet Descriptor Table, the Transmit Queue Base, and the Receive Queue Base. The register accesses are described in sections 4.1 and 4.6.

2. Assign start, read, write, and end indexes for all queues. The register accesses are described in section 4.6.

3. Configure the serial links. The register accesses are described in section 7.

4. Configure the GPIC672 interface. The register accesses are described in section 8.3.

5. Configure HDLC processing of the RHDL672 and the THDL672 blocks. The register accesses are described in sections 9.1 and 9.2.

10.4 Activation Procedure

The activation procedure is required to place the FREEDM-32P672 in a state after which the software may service FREEDM-32P672 interrupts, provision/unprovision channels, make transmit requests and monitor the status of the FREEDM-32P672.

The activation procedure normally follows the initialization procedure.

The steps to activate a FREEDM-32P672 are:

1. Enable interrupt 'E' bits as described in section 5.

2. Enable the FREEDM-32P672 DMA activity by setting the ENABLE bits of the RMAC672 and TMAC672 as described in sections 8.1 and 8.2.

3. The SYSCLKA, TDBA, RFP8A, TFP8A, RFPA[3:0], and TFPA[3:0] bits in the **FREEDM-32P672 Master Clock/Frame Pulse/BERT Activity Monitor and**

Accumulation Trigger (0x00C) register should be read periodically to detect for stuck at conditions. The SYSCLKA bit must be read high for proper operation of the FREEDM-32P672. A low value indicates a failure in clocking that is provided at the SYSCLK input pin of the FREEDM-32P672. Similarly, a low value in the other register bits indicates a failure in clocking that is provided by the corresponding input pin.

4. The TLGA[7:0] and the RLGA[7:0] bits in the **FREEDM-32P672 Master Link Activity Monitor** (0x010) register should be read periodically to detect for stuck at conditions. The bits which correspond to links attached to the FREEDM-32P672 should be read high. A low value indicates a failure in clocking that is provided by the corresponding RCLK[n:m], RMVCK[n], RMV8DC, TCLK[n:m], TMVCK[n], or TMV8DC input pins.[1]

10.5 Deactivation Procedure

The deactivation procedure is required to place the FREEDM-32P672 in a state in which it will not interrupt the embedded processor, or make accesses to the packet memory. This procedure should occur after the FREEDM-32P672 actively transfers packets, or to gracefully shut down the FREEDM-32P672.

The steps to deactivate a FREEDM-32P672 are:

1. Disable interrupt 'E' bits as described in section 5.

2. Disable the FREEDM-32P672 DMA activity by programming the ENABLE bits to zero in the RMAC672 and the TMAC672 as described in sections 9.1 and 9.2.

3. Continue by performing the software reset procedure.

10.6 Provisioning a Channel

The provisioning procedure normally follows the activation procedure and enables the FREEDM-32P672 to receive and/or transmit packets.

10.6.1 Receive Channel Provisioning

The steps to provision a receive channel *RCC*, where $0 \leq RCC \leq 671$ are:

[1]Each RLGA[7:0] or TLGA[7:0] bit corresponds to a group of 4 non H-MVIP links (in addition to the H-MVIP links). If less than four links are configured, then the unused RCLK or TCLK inputs should be tied to the same clock as one of the configured links in this group.

1. Disable FREEDM-32P672 processing of the channel's data stream to allow for graceful provisioning. Write the following bits:

Bit	Register	Value
DCHAN[9:0]	**RCAS Channel Disable** (0x10C)	*RCC*
CHDIS	**RCAS Channel Disable** (0x10C)	1

2. Program the Channel FIFO as described in section 9.3.1.

3. Poll the BUSY bit of the **RHDL Indirect Channel Select** (0x200) register until it is zero. This ensures that a previous indirect RAM access has completed and that a new indirect RAM access can be started.

4. Specify the HDLC configuration for this channel by writing appropriate bits in the **RHDL Indirect Channel Data Register #1** (0x204) and the **RHDL Indirect Channel Data Register #2** (0x208) as described in section 9.4. When writing the **RHDL Indirect Channel Data Register #1** (0x204), ensure that the PROV bit is set, and ensure that the FPTR[10:0] bits identify a block within the circular linked list of buffers of step 2.

5. Specify the RHDL672 channel to provision by writing the following register. Then poll the BUSY bit to ensure that it is low before proceeding to step 6.

Bit	Register	Value
CHAN[9:0]	**RHDL Indirect Channel Select** (0x200)	*RCC*
CRWB	**RHDL Indirect Channel Select** (0x200)	0
BUSY	**RHDL Indirect Channel Select** (0x200)	X

6. Poll the BUSY bit of the **RCAS Indirect Link and Time-slot Select** (0x100) register until it is zero. This ensures that a previous indirect RAM access has completed and that a new indirect RAM access can be started.

7. Specify the RCAS672 channel that is provisioned. Write the following register:

Bit	Register	Value
CHAN[9:0]	**RCAS Indirect Channel Data** (0x104)	*RCC*
PROV	**RCAS Indirect Channel Data** (0x104)	1
CDLBEN	**RCAS Indirect Channel Data** (0x104)	0

8. For a **channelised** link, specify the time-slots which are assigned for processing on this channel by writing the following register once for each time-slot that is assigned to the channel. Valid values for TSLOT[4:0] are 1 through 24 for a T1/J1 link, 1 through 31 for an E1 link, and 0 through 31 for an H-MVIP link. For an **unchannelised** link, TSLOT[4:0] must only have the value 0, and this register is written just once. Each write must be followed by a read to determine whether the BUSY bit (bit15) is low, and to ensure that the indirect RAM has been updated.

Bit	Register	Value
TSLOT[4:0]	**RCAS Indirect Link and Time-slot Select** (0x100)	see above
LINK[4:0]	**RCAS Indirect Link and Time-slot Select** (0x100)	0 through 31 are valid
Reserved[1:0]	**RCAS Indirect Link and Time-slot Select** (0x100)	00
RWB	**RCAS Indirect Link and Time-slot Select** (0x100)	0
BUSY	**RCAS Indirect Link and Time-slot Select** (0x100)	X

9. Enable FREEDM-32P672 processing of the channel data stream to allow for graceful provisioning. Write the following bits:

Bit	Register	Value
DCHAN[9:0]	**RCAS Channel Disable** (0x10C)	*RCC*
CHDIS	**RCAS Channel Disable** (0x10C)	0

Warning:

- The RCAS Channel Disable bit (CHDIS) is only applicable to one channel at a time. In other words, the receive channel provisioning procedure needs to be run once for each channel.

- The programmer must ensure that the channel has not been provisioned, or has been unprovisioned before doing the provisioning procedure. The reset procedure has the effect of unprovisioning all channels of the FREEDM-32P672.

- Continuous polling of a register in a tight loop involves multiple PCI memory read transactions and may have an adverse effect on the PCI bus bandwidth

available for other activities. The recommended method of polling the BUSY bit is to read the register on expiration of a system timer, or after a number of CPU clock ticks. Recommended time intervals are in the range 1 msec through 100 msec.

- A Channel is not provisioned until the BUSY bit toggles low.

10.6.2 Transmit Channel Provisioning

The steps to provision a transmit channel *TCC*, where $0 \leq TCC \leq 671$ are:

1. Disable FREEDM-32P672 processing of the channel's data stream to allow for graceful provisioning. Write the following bits:

Bit	Register	Value
DCHAN[9:0]	**TCAS Channel Disable** (0x410)	*TCC*
CHDIS	**TCAS Channel Disable** (0x410)	1

2. Program the Channel FIFO as described in section 9.3.2 for a transmit channel.

3. Poll the BUSY bit of the **THDL Indirect Channel Select** (0x380) register until it is zero. This ensures that a previous indirect RAM access has completed and that a new indirect RAM access can be started.

4. Specify the HDLC configuration for this transmit channel by writing the **THDL Indirect Channel Data Register #1** (0x384), **THDL Indirect Channel Data Register #2** (0x388) and the **THDL Indirect Channel Data Register #3** (0x38C) as described in section 9.5. In writing the **THDL Indirect Channel Data Register #1** (0x384), ensure the PROV bit is set, and ensure the FPTR[10:0] bits identify a block within the circular linked list of buffers of step 2.

5. Specify the THDL672 channel that is provisioned by writing the following register. Then poll the BUSY bit to ensure it is low before proceeding with step 6.

Bit	Register	Value
CHAN[9:0]	**THDL Indirect Channel Select** (0x380)	*TCC*
CRWB	**THDL Indirect Channel Select** (0x380)	0
BUSY	**THDL Indirect Channel Select** (0x380)	X

6. Poll the BUSY bit of the **TCAS Indirect Link and Time-slot Select** (0x400) register until it is zero. This ensures that a previous indirect RAM access has completed and that a new indirect RAM access can be started.

7. Specify the TCAS672 channel that is provisioned. Write the following register:

Bit	Register	Value
CHAN[9:0]	**TCAS Indirect Channel Data** (0x404)	*TCC*
PROV	**TCAS Indirect Channel Data** (0x404)	1

8. For a **channelised** link, specify the time-slots which are assigned for processing on this channel by writing the following register once for each time-slot that is assigned to the channel. Valid values for TSLOT[4:0] are 1 through 24 for a T1/J1 link, 1 through 31 for an E1 link, and 0 through 31 for an H-MVIP link. For an **unchannelised** link, TSLOT[4:0] must only have the value 0, and this register is written just once. Each write must be followed by a read to determine whether the BUSY bit (bit15) is low, and to ensure that the indirect RAM has been updated.

Bit	Register	Value
TSLOT[4:0]	**TCAS Indirect Link and Time-slot Select** (0x400)	see above
LINK[4:0]	**TCAS Indirect Link and Time-slot Select** (0x400)	0 through 31 are valid
Reserved[1:0]	**TCAS Indirect Link and Time-slot Select** (0x400)	00
RWB	**TCAS Indirect Link and Time-slot Select** (0x400)	0
BUSY	**TCAS Indirect Link and Time-slot Select** (0x400)	X

9. Poll the BUSY bit of the **TMAC Indirect Channel Provisioning** (0x304) register until it is zero. This ensures that a previous indirect RAM access has completed and that a new indirect RAM access can be started.

10. Specify the TMAC672 channel to provision. Write the following register fields, then poll the BUSY bit to ensure the provisioning process has completed.

Bit	Register	Value
CHAN[9:0]	**TMAC Indirect Channel Provisioning** (0x304)	*TCC*

Bit	Register	Value
PROV	**TMAC Indirect Channel Provisioning** (0x304)	1
RWB	**TMAC Indirect Channel Provisioning** (0x304)	0
BUSY	**TMAC Indirect Channel Provisioning** (0x304)	X

11. Enable FREEDM-32P672 processing of the channel data stream to allow for graceful provisioning. Write the following bits:

Bit	Register	Value
DCHAN[9:0]	**TCAS Channel Disable** (0x410)	*TCC*
CHDIS	**TCAS Channel Disable** (0x410)	0

Warning:

- The TCAS Channel Disable bit (CHDIS) is only applicable to one channel at a time. In other words, the transmit channel provisioning procedure needs to be run once for each channel.
- The programmer must ensure that the channel has not been provisioned, or has been unprovisioned before doing the provisioning procedure. The reset procedure has the affect of unprovisioning all channels of the FREEDM-32P672.
- Continuous polling of a register in a tight loop involves multiple PCI memory read transactions and may have an adverse effect on the PCI bus bandwidth available for other activities. The recommended method of polling the BUSY bit is to read the register on expiration of a system timer, or after a number of CPU clock ticks. Recommended time intervals are in the range 1 msec through 100 msec.
- A Channel is not provisioned until the BUSY bit toggles low.

10.7 Unprovisioning a Channel

The unprovisioning procedure is normally applied to channels that are provisioned.

10.7.1 Receive Channel Unprovisioning

The steps to unprovision a receive channel *RCC*, where $0 \leq RCC \leq 671$ are:

1. Disable FREEDM-32P672 processing of the channel's data stream to allow for graceful unprovisioning. Write the following bits:

Bit	Register	Value
DCHAN[9:0]	**RCAS Channel Disable** (0x10C)	*RCC*
CHDIS	**RCAS Channel Disable** (0x10C)	1

2. Poll the BUSY bit of the **RCAS Indirect Link and Time-slot Select** (0x100) register until it is zero. This ensures that a previous indirect RAM access has completed and that a new indirect RAM access can be started.

3. Specify the RCAS672 channel to unprovision by writing the following register:

Bit	Register	Value
CHAN[9:0]	**RCAS Indirect Channel Data** (0x104)	*RCC*
PROV	**RCAS Indirect Channel Data** (0x104)	0
CDLBEN	**RCAS Indirect Channel Data** (0x104)	X

4. For a **channelised** link, specify the time-slots which are unassigned on this channel by writing the following register once for each time-slot that is unassigned. Valid values for TSLOT[4:0] are 1 through 24 for a T1/J1 link, 1 through 31 for an E1 link, and 0 through 31 for an H-MVIP link. For an **unchannelised** link, TSLOT[4:0] must only have the value 0, and this register is written just once. Each write must be followed by a read to determine whether the BUSY bit (bit15) is low, and to ensure that the indirect RAM has been updated.

Bit	Register	Value
TSLOT[4:0]	**RCAS Indirect Link and Time-slot Select** (0x100)	see above
LINK[4:0]	**RCAS Indirect Link and Time-slot Select** (0x100)	0 through 31 are valid
Reserved[1:0]	**RCAS Indirect Link and Time-slot Select** (0x100)	00
RWB	**RCAS Indirect Link and Time-slot Select** (0x100)	0
BUSY	**RCAS Indirect Link and Time-slot Select** (0x100)	X

5. Poll the BUSY bit of the **RHDL Indirect Channel Select** (0x200) register until it is zero. This ensures that a previous indirect RAM access has completed and that a new indirect RAM access can be started.

6. Read the RHDL672 channel data by writing the following register. Then poll the BUSY bit to ensure it is low before proceeding with step 7.

Bit	Register	Value
CHAN[9:0]	**RHDL Indirect Channel Select** (0x200)	*RCC*
CRWB	**RHDL Indirect Channel Select** (0x200)	1
BUSY	**RHDL Indirect Channel Select** (0x200)	X

7. Read the RHDL672 indirect channel data and check that the TAVAIL bit of the **RHDL Indirect Channel Data #1** (0x204) register is zero. This ensures that the last DMA transfer request for this channel has completed. If the TAVAIL bit is zero, proceed to step 8, otherwise, return to step 6.

8. Write the **RHDL Indirect Channel Data #1** (0x204) register with PROV modified to zero, while keeping the same FPTR[10:0] bits.

9. Specify the RHDL672 channel to unprovision by writing the following register. Then poll the BUSY bit to ensure that it is low before proceeding with step 10.

Bit	Register	Value
CHAN[9:0]	**RHDL Indirect Channel Select** (0x200)	*RCC*
CRWB	**RHDL Indirect Channel Select** (0x200)	0
BUSY	**RHDL Indirect Channel Select** (0x200)	X

10. Poll the BUSY bit of the **RMAC Indirect Channel Provisioning** (0x284) register until it is zero. This ensures that a previous indirect RAM access has completed and that a new indirect RAM access can be started. Please see the notes below for the case where the BUSY bit is stuck at one due to empty receive free queues.

11. Ensure that partially filled buffer(s) for the channel are returned to the RPDR Ready queue by writing the following register. Then poll the BUSY bit to ensure that it is low before proceeding with step 12.

Bit	Register	Value
CHAN[9:0]	**RMAC Indirect Channel Provisioning** (0x284)	*RCC*

Bit	Register	Value
PROV	**RMAC Indirect Channel Provisioning** (0x284)	0
RWB	**RMAC Indirect Channel Provisioning** (0x284)	0
BUSY	**RMAC Indirect Channel Provisioning** (0x284)	X

12. Enable FREEDM-32P672 processing of the unprovisioned channel. Write the following bits:

Bit	Register	Value
DCHAN[9:0]	**RCAS Channel Disable** (0x10C)	*RCC*
CHDIS	**RCAS Channel Disable** (0x10C)	0

Note: If the out-of-buffer situation occurs (RPDFQEI error occurs and the BUSY bit of the **RMAC Indirect Channel Provisioning** (0x284) register is stuck at one) unexpectedly because an extremely large erroneous packet of a channel consumes all of the free buffers, the following steps can be used to free up the buffers:

1. Disable the RMAC672 by setting the ENABLE bit of the **RMAC Control** (0x280) register to zero.

2. Place a descriptor pointing to the emergency buffer onto either the RPDR Free Small queue or RPDR Free Large queue. Then, wait a short time for the buffer to be consumed and written to, and for the RMAC672 to return to its state machine's main loop.

3. Unprovision channels using the standard procedure in this section, with the following changes:

   - Keep the RMAC672 disabled by setting the ENABLE bit of the **RMAC Control** (0x280) register to zero, while unprovisioning the channels.

   - Do not wait for the TAVAIL bit in the **RHDL Indirect Channel Data #1** (0x204) register to clear.

   - As part of this process, channels will be flushed in the **RMAC Indirect Channel Provisioning** (0x284) register. This will cause a linked list of buffers to be returned on the RPDR Ready queue. Traverse the linked list to reclaim the buffers.

4. When sufficient buffers have been recovered, re-enable the RMAC672 by setting the ENABLE bit of the **RMAC Control** (0x280) register to one. Remember to set one buffer aside for future emergency use.

Warning:

- The RCAS Channel Disable bit (CHDIS) is only applicable to one channel at a time. In other words, the receive channel unprovisioning procedure needs to be run once for each channel.

- Continuous polling of a register in a tight loop involves multiple PCI memory read transactions and may have an adverse effect on the PCI bus bandwidth available for other activities. The recommended method of polling the BUSY bit is to read the register on expiration of a system timer, or after a number of CPU clock ticks. Recommended time intervals are in the range 1 msec through 100 msec.

- A Channel is not unprovisioned until the BUSY bit toggles low.

10.7.2 Transmit Channel Unprovisioning

The steps to unprovision a transmit channel *TCC*, where $0 \leq TCC \leq 671$ are:

1. Unprovision the TMAC672 channel. Poll the BUSY bit of the **TMAC Indirect Channel Provisioning** (0x304) register until it is zero. Then write the following bits:

Bit	Register	Value
CHAN[9:0]	**TMAC Indirect Channel Provisioning** (0x304)	*TCC*
PROV	**TMAC Indirect Channel Provisioning** (0x304)	0
RWB	**TMAC Indirect Channel Provisioning** (0x304)	0
BUSY	**TMAC Indirect Channel Provisioning** (0x304)	X

2. Poll the BUSY bit of the **TMAC Indirect Channel Provisioning** (0x304) register until it is zero. This ensures that a previous indirect RAM access has completed and that a new indirect RAM access can be started.

3. Disable FREEDM-32P672 processing of the channel's data stream to allow for graceful unprovisioning. Write the following bits:

Bit	Register	Value
DCHAN[9:0]	**TCAS Channel Disable** (0x410)	*TCC*
CHDIS	**TCAS Channel Disable** (0x410)	1

4. Poll the BUSY bit of the **TCAS Indirect Link and Time-slot Select** (0x400) register until it is zero. This ensures that a previous indirect RAM access has completed and that a new indirect RAM access can be started.

5. Specify the TCAS672 channel to be unprovisioned. Write the following register:

Bit	Register	Value
CHAN[9:0]	**TCAS Indirect Channel Data** (0x404)	*TCC*
PROV	**TCAS Indirect Channel Data** (0x404)	0

6. For a **channelised** link, specify the time-slots which are unassigned for processing on this channel by writing the following register once for each time-slot that is unassigned. Valid values for TSLOT[4:0] are 1 through 24 for a T1/J1 link, 1 through 31 for an E1 link, and 0 through 31 for an H-MVIP link. For an **unchannelised** link, TSLOT[4:0] must only have the value 0, and this register is written just once. Each write must be followed by a read to determine whether the BUSY bit (bit15) is low, and to ensure that the indirect RAM has been updated.

Bit	Register	Value
TSLOT[4:0]	**TCAS Indirect Link and Time-slot Select** (0x400)	see above
LINK[4:0]	**TCAS Indirect Link and Time-slot Select** (0x400)	0 through 31 are valid
Reserved[1:0]	**TCAS Indirect Link and Time-slot Select** (0x400)	00
RWB	**TCAS Indirect Link and Time-slot Select** (0x400)	0
BUSY	**TCAS Indirect Link and Time-slot Select** (0x400)	X

7. Poll the BUSY bit of the **THDL Indirect Channel Select** (0x380) register until it is zero. This ensures that a previous indirect RAM access has completed and that a new indirect RAM access can be started.

8. Read the THDL672 channel data by writing the following register. Then poll the BUSY bit to ensure that it is low before proceeding with step 9.

Bit	Register	Value
CHAN[9:0]	**THDL Indirect Channel Select** (0x380)	*TCC*
CRWB	**THDL Indirect Channel Select** (0x380)	1
BUSY	**THDL Indirect Channel Select** (0x380)	X

9. Read the **THDL Indirect Channel Data #1** (0x384) register. Then write this register with PROV modified to zero, while keeping the same FPTR[10:0] bits.

10. Specify the THDL672 channel to unprovision by writing the following register. Then poll the BUSY bit to ensure that it is low before proceeding with step 11.

Bit	Register	Value
CHAN[9:0]	**THDL Indirect Channel Select** (0x380)	*TCC*
CRWB	**THDL Indirect Channel Select** (0x380)	0
BUSY	**THDL Indirect Channel Select** (0x380)	X

11. Enable FREEDM-32P672 processing of the channel. Write the following bits:

Bit	Register	Value
DCHAN[9:0]	**TCAS Channel Disable** (0x410)	*TCC*
CHDIS	**TCAS Channel Disable** (0x410)	0

Warning:

- The TCAS Channel Disable bit (CHDIS) is only applicable to one channel at a time. In other words, the transmit channel unprovisioning procedure needs to be run once for each channel.

- Continuous polling of a register in a tight loop involves multiple PCI memory read transactions and may have an adverse effect on the PCI bus bandwidth available for other activities. The recommended method of polling the BUSY bit is to read the register on expiration of a system timer, or after a number of

CPU clock ticks. Recommended time intervals are in the range 1 msec through 100 msec.

- A Channel is not unprovisioned until the BUSY bit toggles low.

10.8 Receive Sequence

The following sequence of activities takes place when a packet is received on a provisioned receive channel:

1. The FREEDM-32P672 reads a RPDR associated with a free buffer as follows:

   - If this is the first buffer of the receive packet it will read a RPDR associated with a small free buffer from the Small Buffer Cache, or if the cache is empty, from the RPDRF Small queue. RPDRs are read from the queue up to six at a time and stored in the cache.

   - If this is not the first buffer of the receive packet it will read a RPDR associated with a large free buffer from the Large Buffer Cache, or if the cache is empty, from the RPDRF Large queue. RPDRs are read from the queue up to six at a time and stored in the cache.

2. The FREEDM-32P672 generates an interrupt if the programmed number of RPDRs have been read from the RPDR Small (or Large) queue. The RPQSFI or the RPQLFI status bits of the **FREEDM-32P672 Master Interrupt Status** (0x008) register indicate which queue must be replenished with free RPDRs by the software. The software can use the WriteQueue() routine of section 4.6 to replenish the queue. The frequency of the interrupt, and the number of RPDRs to replenish per interrupt, can be programmed via the RPQ_LFN[1:0] and the RPQ_SFN[1:0] bits of the **RMAC Control** (0x280) register.

3. If the receive packet requires multiple buffers to be filled, and the RPDR is not the first for this packet, then the following fields of the previous RPD are written by the FREEDM-32P672: RCC[9:0], CE, Status[5:0], Bytes in Buffer[15:0], and Next RPD Pointer[14:0].

4. The FREEDM-32P672 reads the RPD associated with the free RPDR to determine the buffer address, size and offset.

5. The FREEDM-32P672 writes receive data to the buffer. The priority and configuration of the channel determines when the PCI bus access may occur, and the number of blocks transferred in each access.

6. The above sequence is repeated until the end of packet occurs.

7. When the end of packet occurs the following fields of the RPD are written by the FREEDM-32P672: RCC[9:0], CE, Status[5:0], Bytes in Buffer[15:0], and Next RPD Pointer[14:0].

8. The STATUS[1:0] field of the RPDR is assigned and the first RPDR in the linked list of RPDs which describe the receive packet data is written to the RPDR Ready queue by the FREEDM-32P672.

9. The embedded processor is interrupted if the programmed number of RPDRs have been written to the RPDR Ready queue. The RPQRDYI interrupt status bit is set within the **FREEDM-32P672 Master Interrupt Status** (0x008) register. The number of RPDRs required to generate an interrupt is specified by the RPQ_RDYN[2:0] field of the **RMAC Control** (0x280) register.

10. The software responds to the interrupt by reading from the RPDR Ready queue as per the ReadQueue() procedure of section 4.6. Each RPDR represents one packet, whereby the RPDR points to the first RPD in the packet. When the Status[1:0] bits of the RPDR read from the queue has an error status of 01B the software must go to the last RPDR that links the individual RPDs and examine the Status[5:0] field of the last RPD to determine the cause of error.

Note: During the receive channel unprovisioning procedure, all RPDRs read from the RPDRF Large or Small queue that are partially processed for the unprovisioned channel are written to RPDR Ready queue. The software must examine the STATUS[1:0] bits of the RPDR to determine whether it is an unprovisioned partial packet. If the RPDR is an unprovisioned partial packet then the software must unlink the chain based on the CE bit and the RMAC Next RPD Pointer[14:0] of each RPD in the chain. This will ensure that all RPDs and data buffers are returned to the RPDR Ready queue.

10.9 Transmit Sequence

The following sequence of activities takes place when a packet is transmitted on a provisioned transmit channel:

1. The software initializes and links one or more TDs to identify the transmit packet(s). The software must not re-use a TD within the Transmit Descriptor Table until it has been freed onto the TDR Free queue.

2. The software writes one or more TDRs to the TDR Ready queue as per the WriteQueue() routine of section 4.6. Each TDR that is written to the TDR

Ready queue points to the first TD in chain of TDs which describe the packet(s).

3. The software writes the **TMAC Transmit Descriptor Reference Ready Queue Write** (0x32C) register during the WriteQueue() routine. In response to a change in this register the FREEDM-32P672 reads packet memory at the TDR Ready queue locations that were written by the software.

4. The TMAC672 links the first TDR of each packet read from the TDR Ready queue to a linked list maintained by the TMAC672. The TMAC672 linked list is maintained on a channel basis. The TMAC672 writes the TMAC Next TD Pointer[14:0] field and the V bit field within the first TD of the last packet in the TMAC672 linked list. There is no linking if the TMAC672 linked list is empty.

5. The TMAC672 reads and transmits buffer data according to the Host linked list and the TMAC672 linked list. The priority and configuration of each provisioned channel determines when the buffer data is read.

6. After reading the contents of a buffer the TMAC672 assigns the STATUS[2:0] bits of the TDR and then writes the TDR as follows:

   - the TDR is written directly to the TDR Free queue if the CACHE bit is zero within the **TMAC Control** (0x300) register. Or,

   - if the IOC bit is set within the TD then all TDRs within the TDR Cache are flushed to the TDR Free queue the TDR is written to the TDR Free queue. Or,

   - the TDR is written to the TDR cache because the CACHE bit is set within the **TMAC Control** (0x300) register. Freed TDRs are cached and written up to six at a time to the TDR Free queue.

7. The embedded processor is interrupted if either of the following has occurred:

   - one or more TDR are written to the TDR Free queue because the IOC bit was set in the last TD processed by the TMAC672. The IOCI status bit is set within the **FREEDM-32P672 Master Interrupt Status** (0x008) register.

   - one or more TDR are written to the TDR Free queue because the CACHE bit is set, and the TDQ_FRN[1:0] bits of the TMAC Control (0x300) register specify that an interrupt should occur. The TDQFI status bit is set within the **FREEDM-32P672 Master Interrupt Status** (0x008) register.

- one TDR is written to the TDR Free queue because the CACHE bit is zero in the **TMAC Control** (0x300) register. The TDQFI status bit is set within the **FREEDM-32P672 Master Interrupt Status** (0x008) register.

8. The software responds to the interrupt by reading from the TDR Free queue as per the ReadQueue() procedure of section 4.6. The FREEDM-32P672 returns TDRs to the free queue (or the TDR cache) one at a time, as the data buffer is processed. Therefore the software must examine the Status[2:0] field of the TDR to determine whether the TDR is associated with the last data buffer in the linked list - indicating the transmit packet has been completely processed - or to determine whether any errors occurred in processing the individual data buffers of the linked TDs that form the packet.

Note: During the transmit channel unprovisioning procedure, a TDR that is being processed by the FREEDM-32P672 and that belongs to the unprovisioned channel is written to TDR Free queue. The software must examine the STATUS[2:0] bits to determine whether the TDR is associated with an unprovisioned partial packet. If the TDR is an unprovisioned partial packet, then the software must unlink the chain based on the V bit and the TMAC Next TD Pointer[14:0] of each TD in the chain. It must also unlink descriptors in the host chain, based on the CE bit and the Host Next TD Pointer[14:0], of the unprovisioned TDR. This will ensure that all TDs (and data buffers) of the unprovisioned channel are returned to the host.

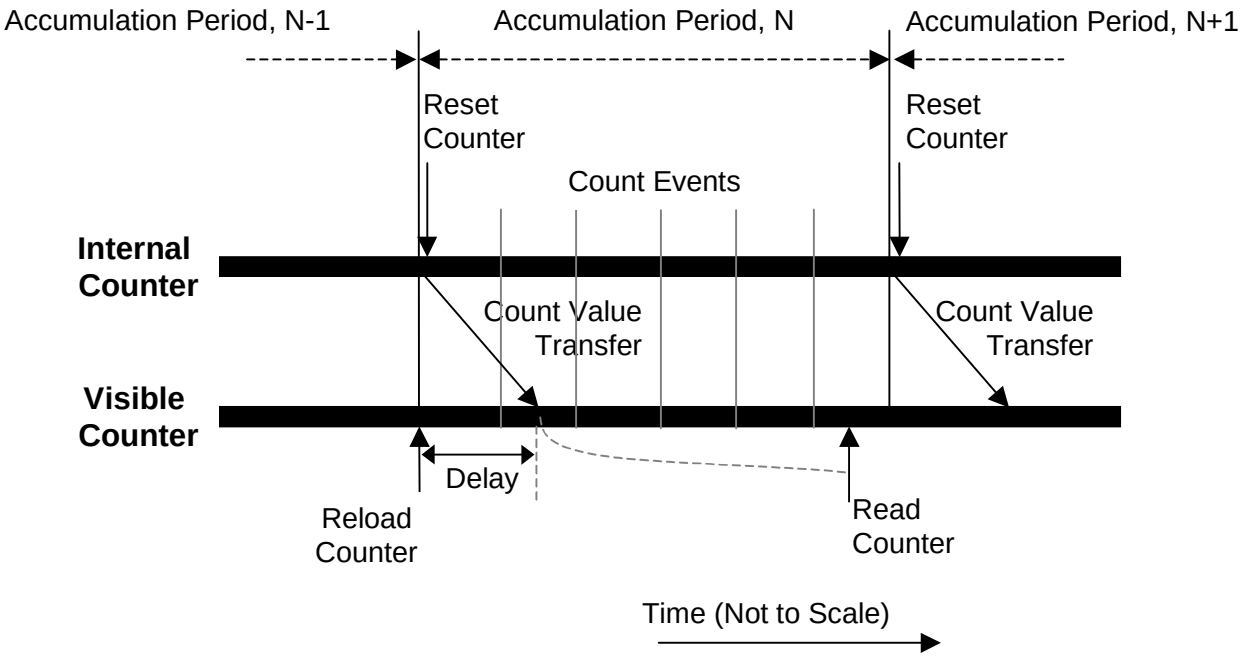
10.10 Performance Counters

The FREEDM-32P672 provides four count registers within the Normal Mode Register Space. These are as follows:

Bit	Register
OF[15:0]	**PMON Receive FIFO Overflow Count** (0x504)
UF[15:0]	**PMON Transmit FIFO Underflow Count** (0x508)
C1[15:0]	**PMON Configurable Count #1** (0x50C)
C2[15:0]	**PMON Configurable Count #2** (0x510)

The software must poll these counters to prevent overflow. Figure 25 illustrates the sequence of events when the counters are polled. The **PMON Status** (0x500) register provides status bits which indicate whether any of the four internal holding counters has overflowed.

Figure 25 – Event Sequence for Polling of Counters



The software initiates a counter reload by writing to the **FREEDM-32P672 Master Clock/Frame Pulse/BERT Activity Monitor and Accumulation Trigger** (0x00C) register. There is a small delay to transfer data from internal counters to the visible counters. The recommended polling strategy is to read the counters first before initiating a reload. Using this strategy, the transfer latency can be ignored.

Counters are normally configured during initialization. The first configurable count register is assigned by setting one of the following register bits, while setting all other bits to zero:

Bit	Register
RSPE1EN	**FREEDM-32P672 Master Performance Monitor Control** (0x024)
RFCSE1EN	**FREEDM-32P672 Master Performance Monitor Control** (0x024)
RABRT1EN	**FREEDM-32P672 Master Performance Monitor Control** (0x024)

Bit	Register
RLENE1EN	**FREEDM-32P672 Master Performance Monitor Control** (0x024)
RP1EN	**FREEDM-32P672 Master Performance Monitor Control** (0x024)
TABRT1EN	**FREEDM-32P672 Master Performance Monitor Control** (0x024)
TP1EN	**FREEDM-32P672 Master Performance Monitor Control** (0x024)

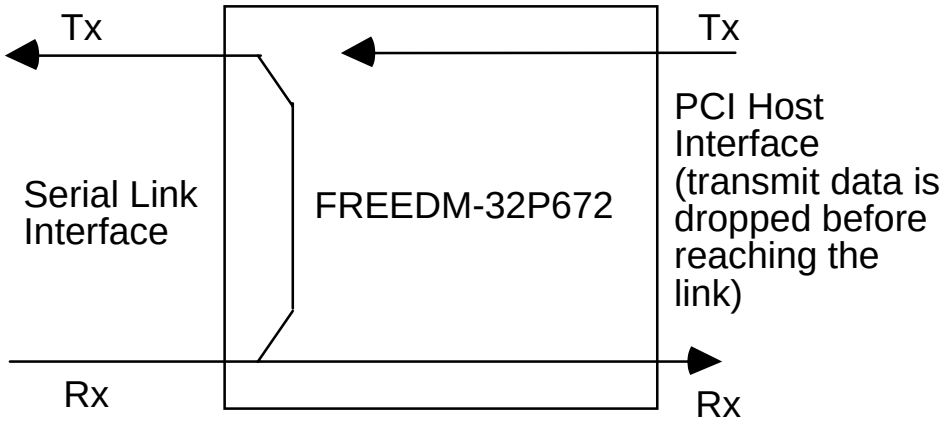
The second configurable count register is assigned by setting one of the following register bits, while setting all other bits to zero:

Bit	Register
RSPE2EN	**FREEDM-32P672 Master Performance Monitor Control** (0x024)
RFCSE2EN	**FREEDM-32P672 Master Performance Monitor Control** (0x024)
RABRT2EN	**FREEDM-32P672 Master Performance Monitor Control** (0x024)
RLENE2EN	**FREEDM-32P672 Master Performance Monitor Control** (0x024)
RP2EN	**FREEDM-32P672 Master Performance Monitor Control** (0x024)
TABRT2EN	**FREEDM-32P672 Master Performance Monitor Control** (0x024)
TP2EN	**FREEDM-32P672 Master Performance Monitor Control** (0x024)

10.11 Line Loopback

Serial ports configured for non H-MVIP traffic can be placed in line loopback. In this configuration, receive data from the serial link is looped back to the transmit serial link as illustrated in Figure 26. Line loopback is not supported for H-MVIP traffic.

Figure 26 – Line Loopback



Non H-MVIP serial ports can be placed in line loopback by setting the appropriate bit within one of the following registers. There are 32 bits corresponding to the 32 serial ports.

Bit	Register
LLBEN[15:0]	**FREEDM-32P672 Master Line Loopback #1** (0x014)
LLBEN[31:16]	**FREEDM-32P672 Master Line Loopback #2** (0x018)

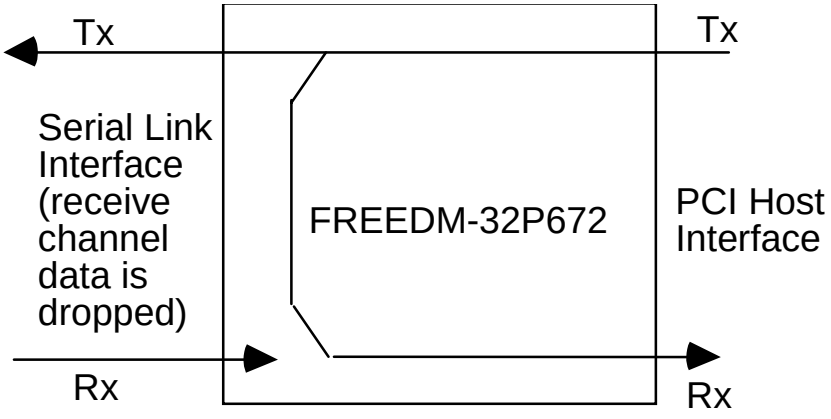
Note: The software should unprovision channels associated with the link that is placed in line loopback mode before placing the link in line loopback. This will prevent the data stream at the serial link from passing through the FREEDM-32P672 to the PCI interface.

10.12 Diagnostic Loopback

Each channel of the FREEDM-32P672 can be placed in a diagnostic loopback mode. In this configuration, the transmit data stream is looped back to the receive data stream as illustrated in Figure 27. The pair of transmit/receive channels is configured in diagnostic loopback mode by provisioning both the transmit and the receive channels as specified in section 10.6, except with the CDLBEN bit set high within the **RCAS Indirect Channel Data** (0x104) register.

In diagnostic loopback mode, the transmit channel data is looped back as well as driven onto the transmit serial link. The channel data from the receive serial link is dropped. The TCLK[n] input pin provides the bit timing for the diagnostic loopback mode.

Figure 27 – Diagnostic Loopback



Note: For the diagnostic loopback to pass data through the FREEDM-32P672, the receive line clock must be provided at the RCLK[n] input pin for the link that is placed in diagnostic loopback. The data rate for the diagnostic loopback is the same as the clock rate provided at this pin.

10.13 BERT Port

The FREEDM-32P672 provides pins to transmit/receive a BERT data stream on serial links configured for non H-MVIP traffic. The following register is written to enable the BERT port and transmit/receive the BERT data stream on serial port *n*, where $0 \leq n \leq 31$.

Bit	Register	Value
RBSEL[4:0]	**FREEDM-32P672 Master BERT Control** (0x020)	*n*
RBEN	**FREEDM-32P672 Master BERT Control** (0x020)	1
TBSEL[4:0]	**FREEDM-32P672 Master BERT Control** (0x020)	*n*
TBEN	**FREEDM-32P672 Master BERT Control** (0x020)	1

The BERT port is disabled by programming the following register:

Bit	Register	Value
RBSEL[4:0]	**FREEDM-32P672 Master BERT Control** (0x020)	X
RBEN	**FREEDM-32P672 Master BERT Control** (0x020)	0
TBSEL[4:0]	**FREEDM-32P672 Master BERT Control** (0x020)	X
TBEN	**FREEDM-32P672 Master BERT Control** (0x020)	0

The TDBA bit of the **FREEDM-32P672 Master Clock/Frame Pulse/BERT Activity Monitor and Accumulation Trigger** (0x00C) can be read by software to determine whether transmit data is present at the BERT port.

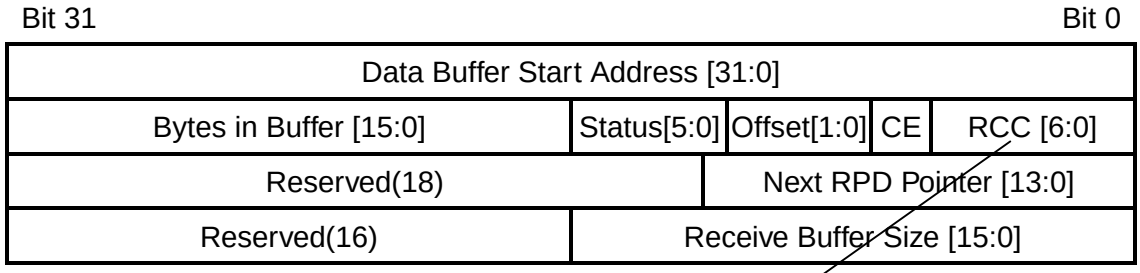
Note: The FREEDM-32P672 channels which are assigned to the same link as the BERT port should be unprovisioned to prevent the receive BERT data stream from passing through the FREEDM-32P672 to the PCI interface.

APPENDIX A – RECEIVE PACKET DESCRIPTOR CHANGES

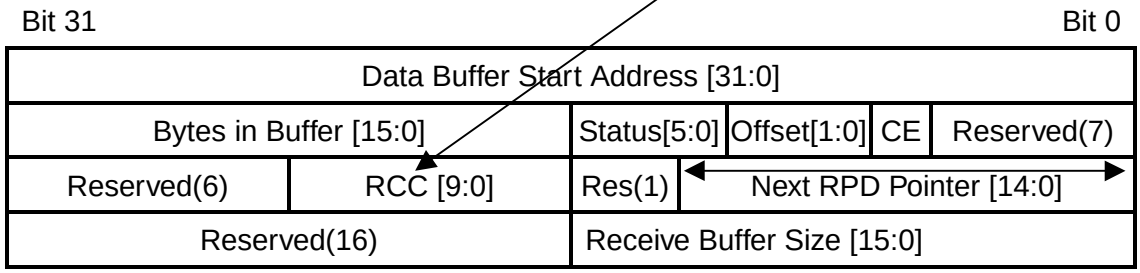
Figure 28 illustrates the changes made to the Receive Packet Descriptor (RPD) from the FREEDM-32 to the FREEDM32-P672.

Figure 28 – Changes to Receive Packet Descriptor

FREEDM-32



FREEDM-32P672



The Receive Channel Code (RCC) field increased from 7 bits to 10 bits as a result of the increase in addressable HDLC channels from 128 to 672 for the FREEDM-32P672. This field has also been relocated.

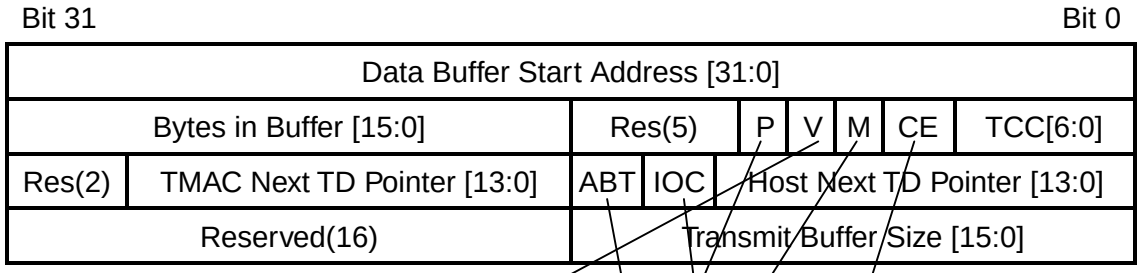
The Next RPD Pointer field increased from 14 bits to 15 bits as a result of the increase in maximum number of addressable descriptors from 16K to 32K for the FREEDM-32P672.

APPENDIX B – TRANSMIT DESCRIPTOR CHANGES

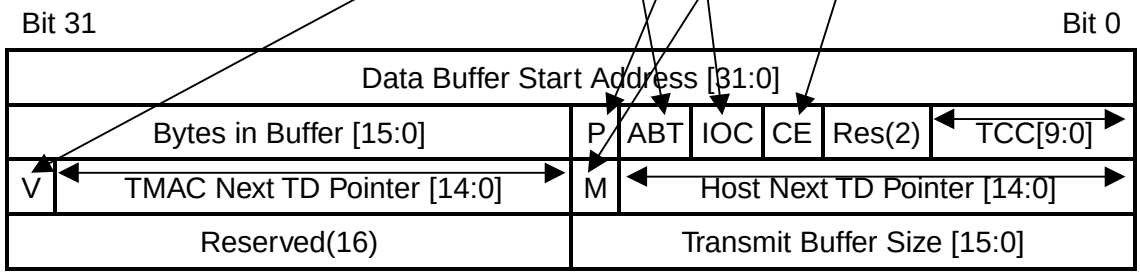
Figure 29 illustrates the changes made to the Transmit Descriptor (TD) from the FREEDM-32 to the FREEDM32-P672.

Figure 29 – Changes to Transmit Descriptor

FREEDM-32



FREEDM-32P672



The Transmit Channel Code (TCC) field increased from 7 bits to 10 bits as a result of the increase in addressable HDLC channels from 128 to 672 for the FREEDM-32P672.

The TMAC Next TD Pointer and Host Next TD Pointer fields increased from 14 bits to 15 bits as a result of the increase in maximum number of addressable descriptors from 16K to 32K for the FREEDM-32P672.

The following control bits have been relocated: P, V, M, CE, ABT and IOC. Please see section 4.3 for a description of these fields.

APPENDIX C – MOVED NORMAL MODE REGISTERS

The following registers have been moved when comparing its FREEDM-32 location to its FREEDM-32P672 location.

Register	FREEDM-32 PCI Offset	FREEDM-32P672 PCI Offset
GPIC Control	0x040	0x080
GPIC Reserved	0x044 – 0x07C	0x084 – 0x0FC

APPENDIX D – NORMAL MODE REGISTER BIT CHANGES

The following normal mode registers have changed at the bit level from the FREEDM-32 to the FREEDM-32P672. Unless otherwise specified, register names, locations and comments refer to FREEDM-32P672 registers.

Register 0x00C : FREEDM-32P672 Master Clock / Frame Pulse /BERT Activity Monitor and Accumulation Trigger

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
11	TFPA[3]	X	Unused	X	Transmit 2.048 Mbps H-MVIP frame pulse activity bits.
10	TFPA[2]	X	Unused	X	
9	TFPA[1]	X	Unused	X	
8	TFPA[0]	X	Unused	X	
7	RFPA[3]	X	Unused	X	Receive 2.048 Mbps H-MVIP frame pulse activity bits.
6	RFPA[2]	X	Unused	X	
5	RFPA[1]	X	Unused	X	
4	RFPA[0]	X	Unused	X	
3	TFP8A	X	Unused	X	Transmit 8.192 Mbps H-MVIP frame pulse activity bit.
2	RFP8A	X	Unused	X	Receive 8.192 Mbps H-MVIP frame pulse activity bit.

Register 0x010 : FREEDM-32P672 Master Link Activity Monitor

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
15	TLGA[7]	X	TLGA[7]	X	Now also monitors TMV8DC input.
11	TLGA[3]	X	TLGA[3]	X	Now also monitors TMVCK[3] input.
10	TLGA[2]	X	TLGA[2]	X	Now also monitors TMVCK[2] input.
9	TLGA[1]	X	TLGA[1]	X	Now also monitors TMVCK[1] input.
8	TLGA[0]	X	TLGA[0]	X	Now also monitors TMVCK[0] input.
7	RLGA[7]	X	RLGA[7]	X	Now also monitors RMVCK[3] and RMV8DC inputs.
6	RLGA[6]	X	RLGA[6]	X	
5	RLGA[5]	X	RLGA[5]	X	Now also monitors RMVCK[2] and RMV8DC inputs.
4	RLGA[4]	X	RLGA[4]	X	
3	RLGA[3]	X	RLGA[3]	X	Now also monitors RMVCK[1] and RMV8DC inputs.
2	RLGA[2]	X	RLGA[2]	X	
1	RLGA[1]	X	RLGA[1]	X	Now also monitors RMVCK[0] and RMV8DC inputs.
0	RLGA[0]	X	RLGA[0]	X	

Register 0x01C : FREEDM-32P672 Reserved

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
0	Reserved	0	Not Specified	X	Reserved bit must be set low for correct operation of FREEDM-32P672.

Register 0x080 : GPIC Control

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
13	RPWTH[5]	0	Unused	X	Increase in size of Receive Packet Write Threshold.

Register 0x100 : RCAS Indirect Link and Time-slot Select

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
12	Reserved	0	LINK[4]	0	Reserved bits must be set low for correct operation of FREEDM-32P672.
11	Reserved	0	LINK[3]	0	
10	LINK[4]	0	LINK[2]	0	
9	LINK[3]	0	LINK[1]	0	
8	LINK[2]	0	LINK[0]	0	
7	LINK[1]	0	Unused	X	
6	LINK[0]	0	Unused	X	

Register 0x104 : RCAS Indirect Channel Data

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
15	CDLBEN	0	Unused	X	
14	PROV	0	Unused	X	
9	CHAN[9]	0	CDLBEN	0	Increase in size of CHAN from 7 bits to 10 bits as the result of increase in HDLC channels from 128 to 672.
8	CHAN[8]	0	PROV	0	
7	CHAN[7]	0	Unused	X	

Register 0x108 : RCAS Framing Bit Threshold

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
6	FTHRES[6]	0	FTHRES[6]	0	
5	FTHRES[5]	1	FTHRES[5]	1	
4	FTHRES[4]	0	FTHRES[4]	1	Change of default value.
3	FTHRES[3]	0	FTHRES[3]	1	Change of default value.
2	FTHRES[2]	1	FTHRES[2]	1	
1	FTHRES[1]	0	FTHRES[1]	1	Change of default value.
0	FTHRES[0]	1	FTHRES[0]	1	

Register 0x10C : RCAS Channel Disable

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
9	DCHAN[9]	0	Unused	X	Increase in size of DCHAN from 7 bits to 10 bits as the result of increase in HDLC channels from 128 to 672.
8	DCHAN[8]	0	Unused	X	
7	DCHAN[7]	0	Unused	X	

Registers 0x180 – 0x188 : RCAS Links #0 to #2 Configuration

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
4	BSYNC	0	Unused	X	
2	MODE[2]	0	BSYNC	0	New mode select bits; see table below. E1 is no longer used. CEN is no longer used.
1	MODE[1]	0	E1	0	
0	MODE[0]	0	CEN	0	

Registers 0x18C – 0x1FC : RCAS Links #3 to #31 Configuration

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
2	MODE[2]	0	Unused	X	New mode select bits; see table below. E1 is no longer used. CEN is no longer used.
1	MODE[1]	0	E1	0	
0	MODE[0]	0	CEN	0	

The mode select bits are encoded as follows to configure the serial links of the FREEDM-32P672:

MODE[2:0]	Link Configuration
000	Unchannelised
001	Channelised T1/J1 (24 time slots labeled 1-24)
010	Channelised E1 (31 time slots labeled 1-31)
011	2 Mbps H-MVIP (32 time slots labeled 0-31)
100	Reserved
101	Reserved
110	Reserved
111	8 Mbps H-MVIP (128 time slots mapped to time-slots 0 through 31 of links 4m, 4m+1, 4m+2 and 4m+3)

Register 0x200 : RHDL Indirect Channel Select

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
9	CHAN[9]	0	Unused	X	Increase in size of CHAN from 7 bits to 10 bits as the result of increase in HDLC channels from 128 to 672.
8	CHAN[8]	0	Unused	X	
7	CHAN[7]	0	Unused	X	

Register 0x204 : RHDL Indirect Channel Data #1

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
14	STRIP	0	CRC[1]	0	CRC[1] moved to bit 11 of Register 0x208 of the FREEDM-32P672.
13	DELIN	0	CRC[0]	0	CRC[0] moved to bit 10 of Register 0x208 of the FREEDM-32P672.
12	TAVAIL	X	STRIP	0	
11	Reserved	X	DELIN	0	Reserved bit must be set low for correct operation of the FREEDM-32P672.
10	FPTR[10]	X	TAVAIL	X	Increase in size of FPTR from 9 bits to 11 bits as the result of increase in addressable descriptors from 512 to 2048.
9	FPTR[9]	X	Unused	X	

Register 0x208 : RHDL Indirect Channel Data #2

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
11	CRC[1]	0	Unused	X	CRC[1] moved from bit 14 of Register 0x204 of the FREEDM-32.
10	CRC[0]	0	Unused	X	CRC[0] moved from bit 13 of Register 0x204 of the FREEDM-32.
3	XFER[3]	0	Unused	X	XFER increased from 3 bits to 4 bits to support larger data transfers.

Register 0x210 : RHDL Indirect Block Select

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
11	Reserved	X	Unused	X	Reserved bit must be set low for correct operation of FREEDM-32P672.
10	BLOCK[10]	X	Unused	X	BLOCK increased from 9 bits to 11 bits as the result of increase in addressable blocks from 512 to 2048.
9	BLOCK[9]	X	Unused	X	

Register 0x214 : RHDL Indirect Block Data

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
11	Reserved	X	Unused	X	Reserved bit must be set low for correct operation of FREEDM-32P672.
10	BPTR[10]	X	Unused	X	BPTR increased from 9 bits to 11 bits as the result of increase in addressable blocks from 512 to 2048.
9	BPTR[9]	X	Unused	X	

Register 0x220 : RHDL Configuration

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
2	Unused	X	Reserved[2]	1	These reserved bits are no longer used in the FREEDM-32P672.
1	Unused	X	Reserved[1]	1	
0	Unused	X	Reserved[0]	1	

Register 0x280 : RMAC Control

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
12	Reserved	0	Unused	X	Reserved bit must be set low for correct operation of FREEDM-32P672.

Register 0x284 : RMAC Indirect Channel Provisioning

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
13	PROV	1	Unused	X	
9	CHAN[9]	0	Unused	X	CHAN increased from 7 bits to 10 bits as the result of increase in HDLC channels from 128 to 672.
8	CHAN[8]	0	Unused	X	
7	CHAN[7]	0	PROV	1	

Register 0x300 : TMAC Control

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
7	FQFLUSH	0	Unused	X	New feature. Can force a flush of the TD Free queue with FQFLUSH bit.

Register 0x304 : TMAC Indirect Channel Provisioning

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
13	PROV	0	Unused	X	
9	CHAN[9]	0	Unused	X	CHAN increased from 7 bits to 10 bits as the result of increase in HDLC channels from 128 to 672.
8	CHAN[8]	0	Unused	X	
7	CHAN[7]	0	PROV	0	

Register 0x380 : THDL Indirect Channel Select

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
9	CHAN[9]	0	Unused	X	Increase in size of CHAN from 7 bits to 10 bits as the result of increase in HDLC channels from 128 to 672.
8	CHAN[8]	0	Unused	X	
7	CHAN[7]	0	Unused	X	

Register 0x384 : THDL Indirect Channel Data #1

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
12	DELIN	X	IDLE	0	IDLE moved to bit 14 of Register 0x38C of the FREEDM-32P672.
11	Reserved	X	DELIN	0	Reserved bit must be set low for correct operation of FREEDM-32P672.
10	FPTR[10]	0	Unused	X	Increase in size of FPTR from 9 bits to 11 bits as the result of increase in addressable descriptors from 512 to 2048.
9	FPTR[9]	0	Unused	X	

Register 0x388 : THDL Indirect Channel Data #2

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
11	Reserved	0	Unused	X	Reserved bit must be set low for correct operation of FREEDM-32P672.
10	FLEN[10]	0	Unused	X	Increase in size of FLEN from 9 bits to 11 bits as the result of increase in addressable descriptors from 512 to 2048.
9	FLEN[9]	0	Unused	X	

Register 0x38C : THDL Indirect Channel Data #3

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
14	IDLE	0	Unused	X	IDLE moved from bit 12 of Register 0x384 of the FREEDM-32.
3	XFER[3]	0	Unused	X	XFER increased from 3 bits to 4 bits to support larger data transfers.

Register 0x3A0 : THDL Indirect Block Select

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
11	Reserved	X	Unused	X	Reserved bit must be set low for correct operation of FREEDM-32P672.
10	BLOCK[10]	0	Unused	X	BLOCK increased from 9 bits to 11 bits as the result of increase in addressable blocks from 512 to 2048.
9	BLOCK[9]	0	Unused	X	

Register 0x3A4 : THDL Indirect Block Data

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
11	Reserved	X	Unused	X	Reserved bit must be set low for correct operation of FREEDM-32P672.
10	BPTR[10]	0	Unused	X	BPTR increased from 9 bits to 11 bits as the result of increase in addressable blocks from 512 to 2048.
9	BPTR[9]	0	Unused	X	

Register 0x3B0 : THDL Configuration

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
3	BURST[3]	0	Unused	X	BURST increased from 3 bits to 4 bits to increase maximum amount of data requested.

Register 0x400 : TCAS Indirect Link and Time-slot Select

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
12	Reserved	0	LINK[4]	0	Reserved bits must be set low for correct operation of the FREEDM-32P672.
11	Reserved	0	LINK[3]	0	
10	LINK[4]	0	LINK[2]	0	
9	LINK[3]	0	LINK[1]	0	
8	LINK[2]	0	LINK[0]	0	
7	LINK[1]	0	Unused	X	
6	LINK[0]	0	Unused	X	

Register 0x404 : TCAS Indirect Channel Data

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
15	PROV	0	Unused	X	
9	CHAN[9]	0	Unused	X	Increase in size of CHAN from 7 bits to 10 bits as the result of increase in HDLC channels from 128 to 672.
8	CHAN[8]	0	PROV	0	
7	CHAN[7]	0	Unused	X	

Register 0x408 : TCAS Framing Bit Threshold

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
6	FTHRES[6]	0	FTHRES[6]	0	
5	FTHRES[5]	1	FTHRES[5]	0	Change of default value.
4	FTHRES[4]	0	FTHRES[4]	1	Change of default value.
3	FTHRES[3]	0	FTHRES[3]	1	Change of default value.
2	FTHRES[2]	1	FTHRES[2]	1	
1	FTHRES[1]	0	FTHRES[1]	1	Change of default value.
0	FTHRES[0]	1	FTHRES[0]	1	

Register 0x410 : TCAS Channel Disable

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
9	DCHAN[9]	0	Unused	X	Increase in size of DCHAN from 7 bits to 10 bits as the result of increase in HDLC channels from 128 to 672.
8	DCHAN[8]	0	Unused	X	
7	DCHAN[7]	0	Unused	X	

Registers 0x480 – 0x488 : TCAS Links #0 to #2 Configuration

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
4	BSYNC	0	Unused	X	
2	MODE[2]	0	BSYNC	0	New mode select bits; see table below. E1 is no longer used. CEN is no longer used.
1	MODE[1]	0	E1	0	
0	MODE[0]	0	CEN	0	

Registers 0x48C – 0x4FC : TCAS Links #3 to #31 Configuration

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
2	MODE[2]	0	Unused	X	New mode select bits; see table below. E1 is no longer used. CEN is no longer used.
1	MODE[1]	0	E1	0	
0	MODE[0]	0	CEN	0	

The mode select bits are encoded as follows to configure the serial links of the FREEDM-32P672:

MODE[2:0]	Link Configuration
000	Unchannelised
001	Channelised T1/J1 (24 time slots labeled 1-24)
010	Channelised E1 (31 time slots labeled 1-31)
011	2 Mbps H-MVIP (32 time slots labeled 0-31)
100	Reserved
101	Reserved
110	Reserved
111	8 Mbps H-MVIP (128 time slots mapped to time-slots 0 through 31 of links 4m, 4m+1, 4m+2 and 4m+3)

APPENDIX E – PCI CONFIGURATION REGISTER BIT CHANGES

The following PCI Configuration registers have changed at the bit level from the FREEDM-32 to the FREEDM-32P672. Unless otherwise specified, register names, locations and comments refer to FREEDM-32P672 registers.

Register 0x00 : Vendor Identification/Device Identification

Bits	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
31:16	DEVID[15:0]	7380H	DEVID[15:0]	7364H	New device identification for FREEDM-32P672.

Register 0x04: Command/Status

Bit	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
21	66MHZ_CAPABLE	1	Reserved	0	Hardware wired to one to indicate that GPIC672 is 66 MHz capable.

Register 0x08 : Revision Identifier/Class Code

Bits	FREEDM-32P672		FREEDM-32		Comments
	Function	Default	Function	Default	
7:0	REVID[7:0]	00H	REVID[7:0]	01H	Different revision identification.

CONTACTING PMC-SIERRA, INC.

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

Tel: (604) 415-6000

Fax: (604) 415-6200

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

PMC-990545 (A2) date: June 1999