# PM7364

# FREEDM

# PROGRAMMER'S GUIDE

**PRELIMINARY**

**Issue 1: March, 1997**

## CONTENTS

# 1 INTRODUCTION

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

This document is a supplement to the FREEDM Longform Datasheet[1]. Both documents should be studied together to interface the FREEDM 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.

## 1.1 Target Audience

This document has been prepared for readers with some prior knowledge of HDLC and Frame Relay technology. Introductory material on Frame Relay is readily available from several sources.

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

## 1.2 Numbering Conventions

The following numbering conventions are used:

| | |
|---|---|
| binary | 0001B, 1110B |
| decimal | 129, 6, 12 |
| hexadecimal | 0x1FE2, 09FH |

## 1.3 Register Description

Unless specified otherwise, FREEDM 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 - these are the normal mode registers and the PCI configuration registers.

### 1.3.1 Normal Mode Registers

Normal mode registers are used to configure, monitor and control the operation of the FREEDM. Registers must be accessed as 32-bit values with a dword aligned address. A register value is accessed at the PCI interface during a PCI memory 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 versions of the product, unused register bits should be written with logic zero. Reading back unused bits can produce either logic one or a logic zero; Hence unused register bits should be masked off by software during a register read access.

- All configuration bits that can be written into can also be read back. This allows the processor controlling the FREEDM 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 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 operates as intended, reserved register bits must only be written with their default values. Similarly, writing to reserved registers should be avoided.

### 1.3.2 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 are accessed as 8-bit, 16-bit or 32-bit values and register addresses are byte, word, and dword aligned respectively. These registers are accessed via the PCI interface during a PCI configuration access transaction and have the following characteristics:

- Writing values into unused register bits has no effect.

- All configuration bits that can be written into can also be read back. This allows the processor controlling the FREEDM to determine the programming state of the PCI device.

- PCI configuration registers are not cleared upon reset. They are set to the default value on power-up.

- Writing into read-only register bit locations does not affect FREEDM operation.

# 2 FREEDM OVERVIEW

## 2.1 FREEDM Summary

The PM7364 FREEDM is a highly integrated semiconductor device that is ideally suited for Frame Relay applications that require many links to be terminated and HDLC processed within a highly dense design. The functional blocks of the FREEDM are illustrated in figure 1.

As many as 32 bi-directional serial links can be connected to the FREEDM. These are processed by the RCAS and TCAS blocks. Links are individually clocked and can be individually placed in line loopback. The RCAS and TCAS can be interfaced to channelized T1, channelized E1, unchannelized links, or unchannelized links with byte sychronization.

The data stream at each serial port can be assigned to one or more of the FREEDM channels. There are as many as 128 receive channels and 128 transmit channels available for assignment to unchannelized links or time-slots within a channelized link.

Each data stream can be HDLC processed on a channelized basis within the RHDL and THDL. There is an 8 KByte buffer in the RHDL and another 8 KByte buffer in the THDL that must be assigned to FREEDM channels to serve as Channel FIFO's. Each buffer is a group of 512 blocks with 16 bytes per block, and a minimum of 3 blocks must be assigned to a channel during provisioning. This allows flexibility in assignment of a Channel FIFO based on the expected data rate for the channel.

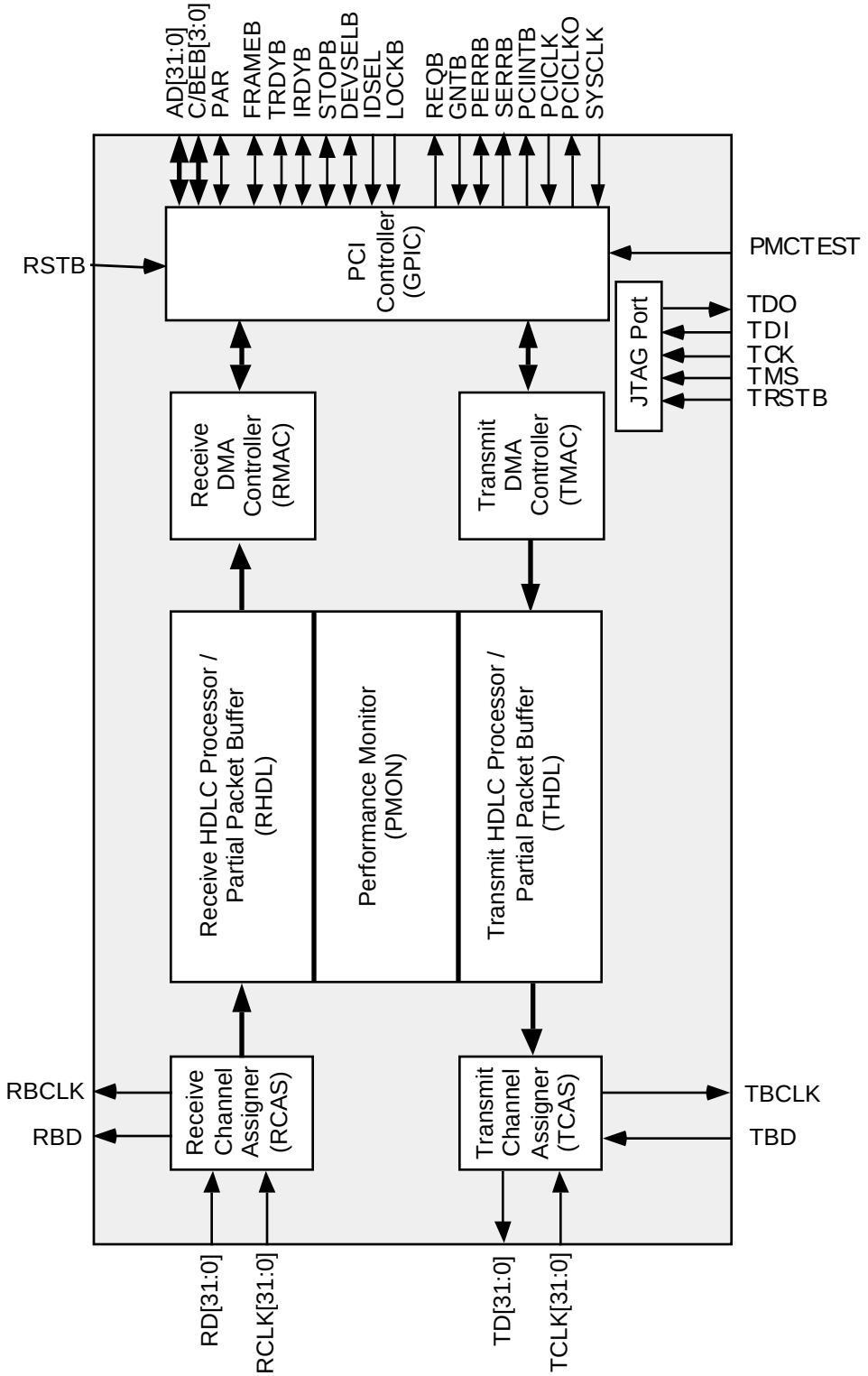
The RHDL and THDL also provide the facility to transfer the raw data stream without HDLC processing.

The FREEDM interfaces to an embedded processor and packet memory via 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 RMAC, the TMAC and the GPIC blocks perform the DMA of buffer data across the PCI local bus.

Each channel provisioned within the FREEDM contends for access to the PCI bus based on its configuration within the RMAC and TMAC 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.

**Fig. 1 FREEDM Block Diagram**



In addition to the line loopback capability, the FREEDM 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. Finally there is an internal diagnostic loopback configuration for each channel, which can be used to diagnose FREEDM operation on a per channel basis.

### 2.2 PCI Interface

Figure 2 shows an address map for a PCI bus which contains one FREEDM device. These data structures are required to interface a FREEDM 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.

When multiple FREEDM's are attached to the bus each FREEDM 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

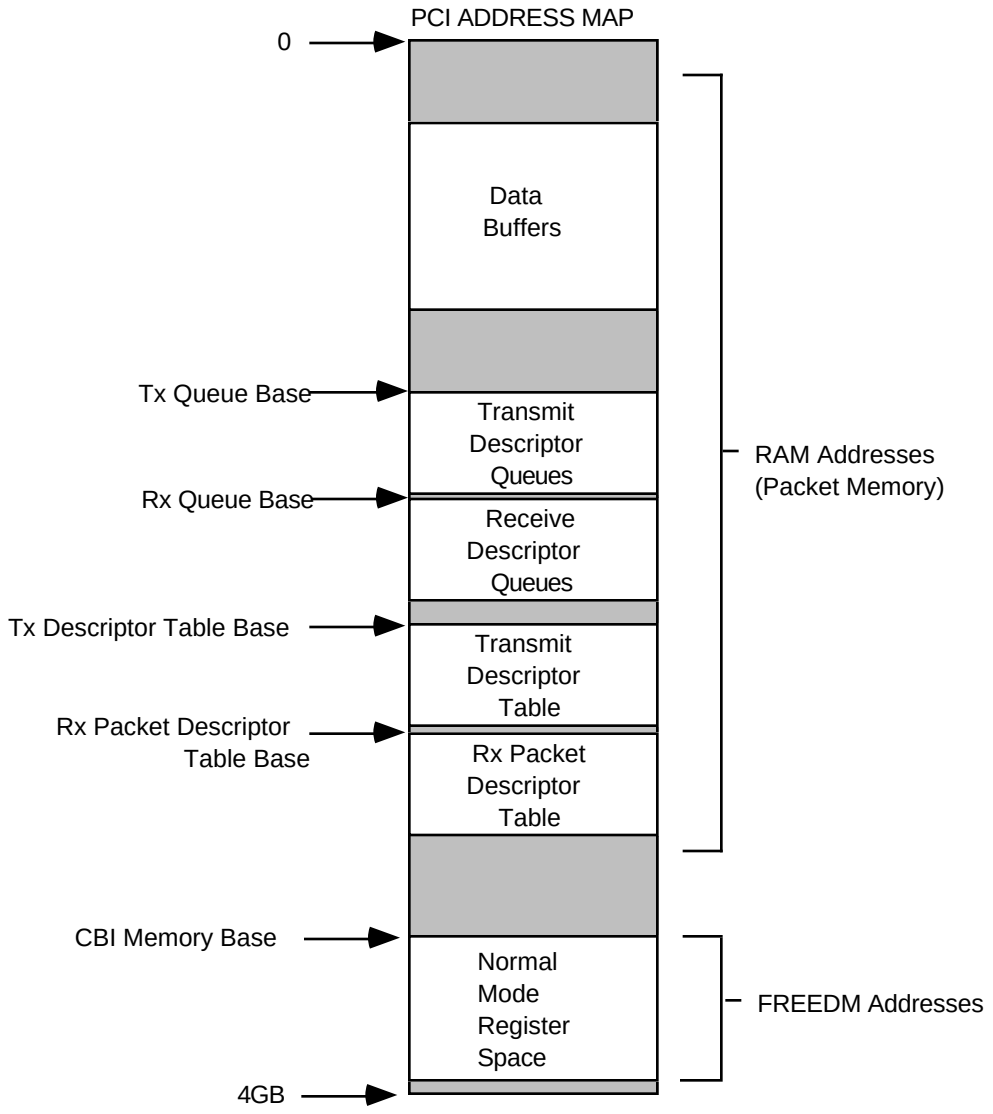
The data structures within RAM are accessed by software running on the embedded processor, or by the FREEDM. The software must specify the location of these data structures by writing base addresses into the appropriate FREEDM registers, before activating the FREEDM.

The FREEDM accesses RAM directly using physical addressing whereas the software may use virtual addressing. In some systems which use virtual memory management the software must translate between virtual addresses (ie pointers) and physical addresses. The software must ensure that the values written to FREEDM 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, or contain transmit data which is read by the FREEDM. 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 device. This register space is located in the FREEDM and is mapped into the PCI address space by the software.

**Fig. 2 PCI Address Map**



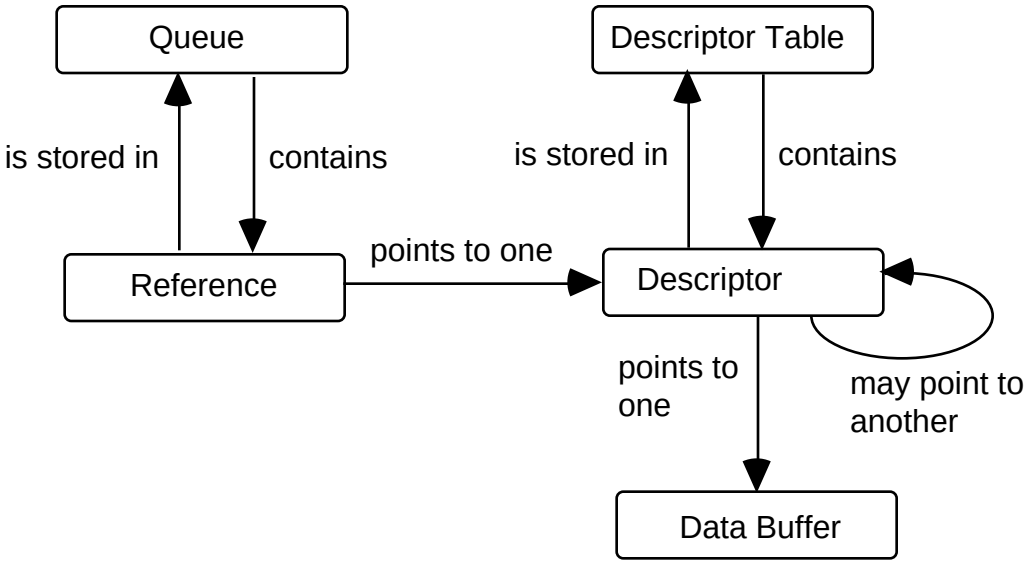

The PCI Configuration Space does not reside in the PCI address map, but it is a requirement for all PCI devices. The Configuration Space is a block of 256 contiguous bytes that reside in the PCI device, and is accessed by the embedded processor in a PCI bus Configuration Read (or Write) transaction, 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 Configuration Space is discussed in a later section of this document.

# 3 DATA STRUCTURES

The RAM data structures accessed by the FREEDM 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.

**Fig. 3 Data Structure Relationship**




These data structures are also accessed by software. The queues specify the data which may be accessed by the FREEDM 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 RPD's. A Transmit Packet consists of a reference pointing to one transmit descriptor (TD), or a linked-list of TD's. Transmit Packets may be linked by software, or by the FREEDM, via seperate fields within each descriptor.

## 3.1 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. There are two descriptor tables. For transmit packets there is the Transmit Descriptor Table and for receive packets there is the Receive Packet Descriptor Table.

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 determines the size of the descriptor table. The relationship is: Size (in bytes) = 16* Number of References.

The table index (reference) is a 14 bit value which limits the size to 16,384 descriptors, or 524288 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 registers must be written with the physical address of the Receive Descriptor Table and the Transmit Descriptor Table, respectively:

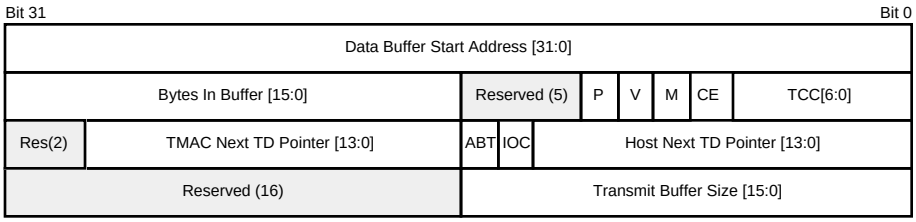
| Bit | Register |
|---|---|
| RPDTB[15:0] | **RMAC Packet Descriptor Table Base LSW** (0x288) |
| RPDTB[31:16] | **RMAC Packet Descriptor Table Base MSW** (0x28C) |
| TDTB[15:0] | **TMAC Transmit Descriptor Table Base LSW** (0x308) |
| TDTB[31:16] | **TMAC Transmit Descriptor Table Base MSW** (0x30C)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

**Note**: RPDTB[3:0] and TDTB[3:0] must be zero, thereby aligning the descriptor tables on 16 byte boundaries.

### 3.2 Transmit Descriptor

A Transmit Descriptor (TD) is a 16 byte data structure that contains a number of fields as shown in figure 4. TD's are used in the transmit direction to describe packets read from packet memory and transmitted by the FREEDM. Each TD is located in the Transmit Descriptor Table and is indexed from the base address using a TD Reference (TDR).

**Fig. 4 Transmit Descriptor**



| Field | Description |
|---|---|
| Data Buffer Start Address [31:0] | The Data Buffer Start Address[31:0] bits point to the data buffer in packet memory.<br>The Data Buffer Start Address field is valid in all TDs |
| Bytes In Buffer [15:0] | The Bytes In Buffer[15:0] field is used by the software to indicate the total number of bytes to be transmitted in the current TD. If M is 1 (see below), this value must be a non-zero multiple of 4, i.e. the buffer must not be empty and must contain an exact number of Dwords. |
| P | The Priority bit is set by the software 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 TMAC. 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. |
| 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[13:0] field is valid. When V is set to logic 0, the TMAC Next TD Pointer[13:0] field is invalid. The V bit is used by the software to reclaim data buffers in the event that data presented to the TMAC is returned host due to a channel becoming unprovisioned. The V bit is expected to be initialised to logic 0 by the host. |
| M | The More (M) bit is used by the software to support packets that require multiple TDs. If M is set to logic 1, the TMAC assumes that the current TD is just one of several TDs for the current packet. If M is set to logic 0, the TMAC assumes that 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).<br>Note: When M is set to logic 1, the only valid value for CE is logic 0. |

| | |
|---|---|
| CE | The Chain End (CE) bit is used by the software to indicate the end of a linked list of TDs presented to the TMAC. The linked list can contain one or more packets as delineated by the M bit (see above). 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[13:0] field is valid, otherwise the field is not valid.<br>Note: When CE is set to logic 1, the only valid value for M is logic 0.<br>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[6:0] | The Transmit Channel Code (TCC[6:0]) field is used by the software to indicate with which channel a TD is associated. All TD in a chain must be associated with the same channel, i.e. have this field set to the same value. |
| TMAC Next TD Pointer [13:0] | The TMAC Next TD Pointer[13:0] bits are used to store TDRs which permits the TMAC 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. In the case that data presented to the TMAC 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 TMAC and host links in order to recover all the buffers. |
| ABT | The Abort (ABT) bit is used by the software to abort the transmission of a packet. When ABT is set to logic 1, the packet will be aborted. 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 software to instruct the TMAC to interrupt the embedded processor when the current TD's data buffer has been read. When IOC is logic 1, the TMAC will assert an interrupt when the associated data buffer has been read, provided the interrupt is enabled. Additionally, the Free Queue FIFO will be flushed. If IOC is logic zero, the TMAC will not generate an interrupt and the Free Queue FIFO will operate normally. |
| Host Next TD Pointer [13:0] | The Host Next TD Pointer[13:0] bits are used to store TDRs which permits the software 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 software to pass multiple TD packets or multiple packets associated with the same channel to the TMAC . |
| 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 TMAC does not make use of this field.) |

Initialization of a Transmit Descriptor By Software

The following fields of a TD (or a linked-list of TD's) 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 |
| V | 0 |
| M | 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 |
| ABT | value is determined during run time or preconfigured |
| IOC | value is determined during run time or preconfigured |
| Reserved | 0 |
| Host Next TD Pointer | value is determined during run time or preconfigured |

Fields Modified By FREEDM

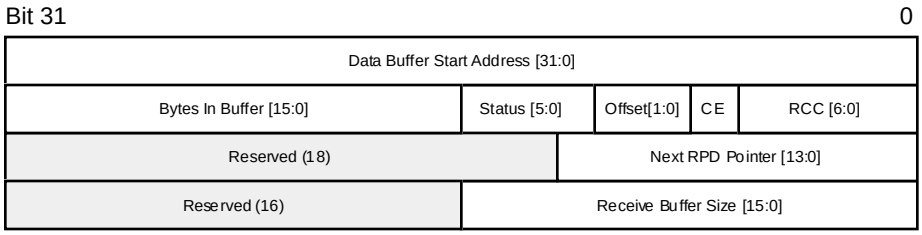
The following fields may be modfied by the FREEDM after it reads the reference from the TDR ready queue, but before the same reference is written to the TDR free queue:

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

### 3.3 Receive Packet Descriptor

A Receive Packet Descriptor (RPD) is a 16 byte data structure that contains a number of fields as shown in figure 5. RPD's are used in the receive direction to describe packets that were 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).

**Fig. 5 Receive Packet Descriptor**




| Field | Description |
|---|---|
| Data Buffer Start Address[31:0] | The Data Buffer Start Address[31:0] bits point to the data buffer in packet memory. This field is expected to be configured by the software during initialisation.<br>The Data Buffer Start Address field is valid in all RPDs. |
| RCC[6:0] | The Receive Channel Code (RCC[6:0]) bits are used by the RMAC to indicate which channel an RPD is associated with.<br>For a linked list of RPDs, all the RPDs' RCC fields are valid, i.e. all contain the same channel value. |
| 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.<br>The CE bit is valid for all RPDs written by the RMAC 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 RMAC 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.<br>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. |
| Status [5:0] | The Status[5:0] bits indicate the status of the received packet.<br>Status[0] Rx buffer overrun<br>Status[1] Packet exceeds max. allowed size<br>Status[2] CRC error<br>Status[3] Packet Length not an exact no. of bytes<br>Status[4] HDLC abort detected<br>Status[5] Unused (set to 0)<br>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.<br>The Bytes in Buffer field is valid in all RPDs. |
| Next RPD Pointer [13:0] | The Next RPD Pointer[13:0] bits store a RPDR which enables the RMAC 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 RMAC links RPDs when more than one buffer is needed to store a packet.<br>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. |
| 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 software during initialisation. The Receive Buffer Size must be a non-zero integer multiple of four.<br>The Receive Buffer Size field is valid in all RPDs. |

**NOTE:** For error checking purposes it is recommended to examine the Bytes in Buffer[15:0] field to ensure that it does not exceed the Recieve Buffer Size[15:0].

Initialization of a RPD By Software

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

RPD Fields Modified By FREEDM

The following fields are modfied by the FREEDM after it reads the reference from the RPDRF large queue, or the RPDRF small queue, but before the same reference is written to the RPDR ready queue:

| Field | Value |
|---|---|
| RCC | value is determined during run time |
| CE | value is determined during run time |
| Status | value is determined during run time |
| Bytes In Buffer | value is determined during run time |
| Next RPD Pointer | value is determined during run time |
| Offset | value is determined during run time |

### 3.4 Data Buffers

In the receive path, the FREEDM writes receive packet data into data buffers. In the transmit path, the FREEDM 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 maximum size of each data buffer is 65536 bytes, and the minimum size is 4 bytes. There is no buffer address alignment or additional size restrictions.

For a receive buffer the following fields of an 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 automatically links RPD's when the receive packet length exceeds the buffer size.

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

### 3.5 References

References are dword structures used to access descriptors within a descriptor table. They also have status bits which are written by the FREEDM after it has processed the packet. These status bits must be read by software to identify whether the FREEDM was successful in processing the packet.

A Transmit Descriptor Reference (TDR) has the following fields:


| Field | Description |
|---|---|
| Status[2:0] | The TMAC fills in the Status field to indicate the results of processing the TD. The encoding is:<br>Status[1:0] Description<br>00 Successful, last or only buffer of packet.<br>01 Successful, buffer of partial packet.<br>10 Failed, unprovisioned channel.<br>11 Failed, malformed packet (e.g. Bytes In Buffer field set to 0).<br>Status[2] Description<br>0 No underflow detected.<br>1 Underflow detected. |
| TDR[13:0] | The TDR[13:0] field contains the offset of the TD returned. |

A Receive Packet Descriptor Reference (RPDR) has the following fields:

Bit 31 Bit 0

| UNUSED | STATUS[1:0] | RPDR[13:0] |
|---|---|---|

| Field | Description |
|---|---|
| Status[1:0] | The encoding for the status field is as follows:<br>00 - Successful reception of packet.<br>01 - Unsuccessful reception of packet.<br>10 - Unprovisioned partial packet.<br>11 - Reserved. |
| RPDR[13:0] | The RPDR[13:0] field defines the offset of the first RPD in a linked chain of RPDs, each pointing to a buffer containing the received data. |

**NOTES**:

- When the FREEDM writes an RPDR into the RPDR Ready queue, or when it writes a TDR into the TDR Free queue, it overwrites unused bits in byte 2 of the reference with a zero value. 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 register. The software should write a non-zero value in byte 2 after reading the reference, and at a later time it can check whether the non-zero value was overwritten by the FREEDM, indicating the FREEDM has written another reference into this queue location.

- The reference, including status bits, is written into a queue by the FREEDM 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.

- Only one RPDR is written into the RPDR Ready queue per receive packet, and this RPDR represents the linked list of RPD's which identify the receive packet.

- 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 TD's there will be multiple TDR's written to the TDR Free queue.

- The Status[2] field of a TDR can be used to identify the occurance of an underflow condition on the channel associated with the TDR. The underflow may or may not have occured on the buffer associated with the TDR read from the TDR Free queue.

TD's or RPD's 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 18 bits of the desc reference to extract
 * the index field. */
#define      RPD_INDEX_MASK    0x00003FFF
#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);
```

### 3.6 Queues

A queue is a FIFO buffer located in fixed memory and it holds a number of references. The FREEDM has 5 queues which must be allocated. There are 2 queues for TDR's and 3 queues for RPDR's. The software must allocate memory for each of these queues.

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 to transmit a packet. The FREEDM 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.

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 reads from the RPDR Free Large queue and the RPDR Free Small queue to get free buffers into which the receive data is written. When the receive operation is complete the FREEDM 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 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 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.

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

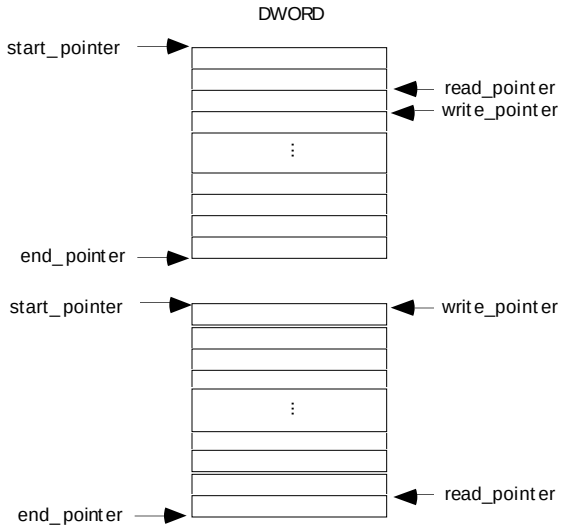
| Queue | Read By | Written By |
|---|---|---|
| TDR Ready | FREEDM | Software |
| TDR Free | Software | FREEDM |
| RPDR Free Small | FREEDM | Software |
| RPDR Free Large | FREEDM | Software |
| RPDR Ready | Software | FREEDM |

There are four indexes for each queue that are used to manage its state. These indexes are located in the FREEDM 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. |

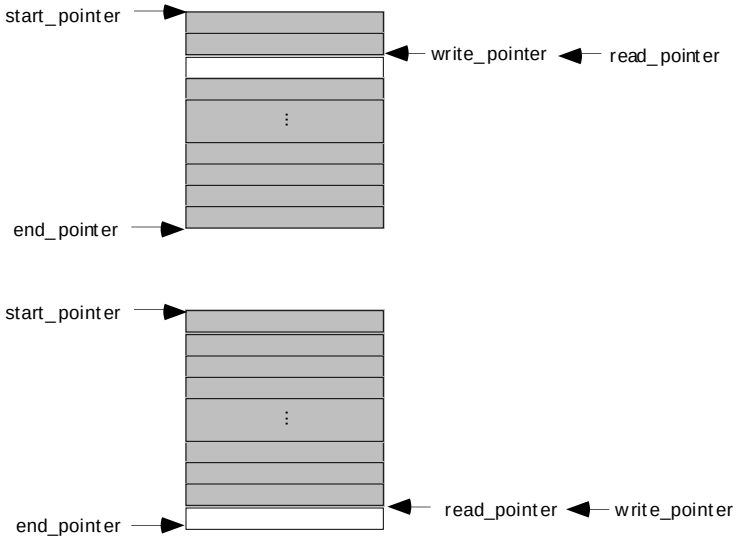
The empty queue states are illustrated in figure 6. The queue is empty when the read index is one location before the write index.

**Fig. 6 Empty Queue States**




The full queue states are illustrated in figure 7. The queue is full when the read index is the same as the write index.

**Fig. 7 Full Queue States**




Allocation of Queues

From figure 7 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. In general, each queue must be large enough to hold one reference per channel that is provisioned.

To obtain the best bus utilization possible 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. 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 transmit packets are located in fixed memory as offsets from a base address. The queues used for receive 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:

| Bit | 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 TDR Ready queue and the TDR free queue must reside within 256K of the TMAC Transmit Queue Base address. The size of each queue is specifed by assignment of the start, write, read and end indexes.

The RPDR Free Large queue, the RPDR Free Small queue and the RPDR Ready queue must reside within 256K of the RMAC Receive 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 |
|---|---|
| TDR Ready | empty |
| TDR Free | empty |
| RPDR Free Small | full |
| RPDR Free Large | full |
| RPDR Ready | empty |

The software must write valid RPD References into the RPDRF Small (and Large) queues.

The software must also write the following FREEDM registers with valid indexes.

| Bit | 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) |
| 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 to write a reference to the queue. This method is recommended when interrupts RPQRDYI and TDQFI are disabled, and processing of the TDR Free queue and the RPDR Ready 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, so
         * the reference is invalid, and PollQueue() does not return
         * a valid reference, so... */
        return FALSE;
     }
     else {
        /* the queue location was overwritten by the FREEDM, so
         * the reference is valid, proceed by overwriting the queue
         * location with an invalid reference. */
        *pQueueElement = INVALID_REFERENCE;

        /* write the FREEDM 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;
     }
}
```

## 4 INTERRUPT ARCHITECTURE

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

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

The FREEDM provides interrupts to the PCI bus via the PCIINTB pin of the FREEDM. 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 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. 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.

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 segement 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);
}
```

# 5 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.

## 5.1 Accesssing the PCI Configuration Space

The FREEDM responds to Type 0 configuration cycles for a single function device, as described in the PCI specification[2]. The FREEDM 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, and 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.

## 5.2 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. The reader is refered 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 8.

- Vendor ID
- Device ID
- Command
- 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 Configuration Space. These fields will return 0 when read.

**Fig. 8 FREEDM 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 Base 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 |

## 6 CONFIGURING THE SERIAL LINKS

Each of the 32-bidirectional links are controlled via the RCAS and the TCAS blocks of the FREEDM. The TCAS controls the transmit data stream while the RCAS controls the receive data stream.

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

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

Each of the serial link configurations are discussed seperately in the following sections.

### 6.1 Channelized T1 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 channelized E1 receive link (see figure 9).

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 channelized T1 transmit link (see figure 10).

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

**Fig. 9 Channelized T1 Receive Link Timing**

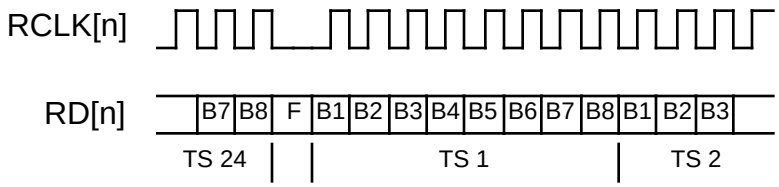
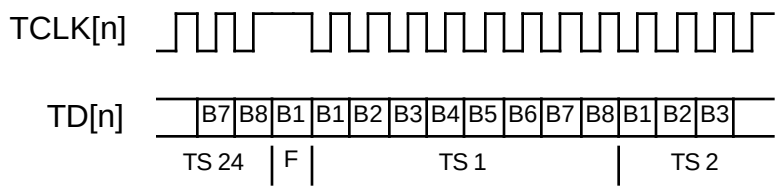


**Fig. 10 Channelized T1 Transmit Link Timing**




To configure the RCAS and TCAS to interface to channelized T1 links the following bits are written:

| Bit | Register | Value |
|---|---|---|
| CEN | **RCAS Link #n Configuration** (see note) | 1 |
| E1 | **RCAS Link #n Configuration** (see note) | 0 |
| BSYNC | **RCAS Link #n Configuration** (see note) | X |
| FTHRES[6:0] | **RCAS Framing Bit Threshold** (0x108) | 0x1F |
| CEN | **TCAS Link #n Configuration** (see note) | 1 |
| E1 | **TCAS Link #n Configuration** (see note) | 0 |
| BSYNC | **TCAS Link #n Configuration** (see note) | X |
| FTHRES[6:0] | **TCAS Framing Bit Threshold** (0x408) | 0x1F |
| 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 0x2FC corresponding to the link being configured for channelized T1 operation. The BSYNC bit is present only in registers with $0 \le n \le 2$. The BSYNC value has no significance for the channelized 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 channelized T1 operation. The BSYNC bit is present only in registers with $0 \le n \le 2$. The BSYNC value has no significance for the channelized mode of operation.

- The FTHRES[6:0] value assumes a SYSCLK of 33Mhz. For other values of SYSCLK the framing threshold value is $1.5 \cdot f_{SYSCLK}/1544000$ (This value ensures the threshold is suitable for both T1 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. The programmer should ensure that these values are suitable for all links attached to a FREEDM.

### 6.2 Channelized 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 channelized E1 receive link (see figure 11).

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 channelized E1 transmit link (see figure 12).

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. The data stream of each direction is processed and clocked independently.

**Fig. 11 Channelized E1 Receive Link Timing**

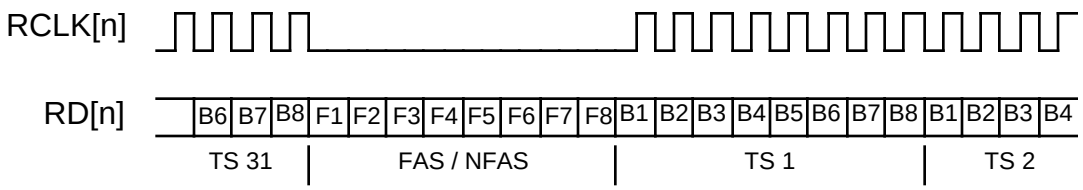
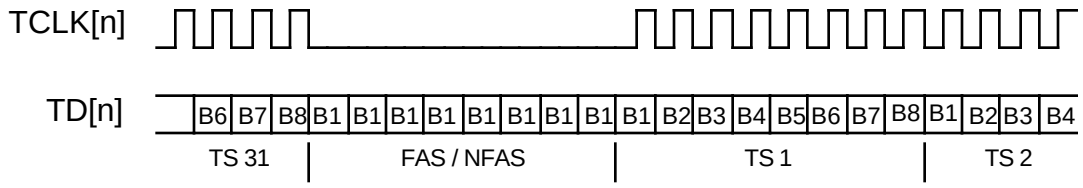



**Fig. 12 Channelized E1 Transmit Link Timing**




To configure the RCAS and TCAS to interface to channelized E1 links the following bits are written:

| Bit | Register | Value |
|---|---|---|
| CEN | **RCAS Link #n Configuration** (see note) | 1 |
| E1 | **RCAS Link #n Configuration** (see note) | 1 |
| BSYNC | **RCAS Link #n Configuration** (see note) | X |
| FTHRES[6:0] | **RCAS Framing Bit Threshold** (0x108) | 0x1F |
| CEN | **TCAS Link #n Configuration** (see note) | 1 |
| E1 | **TCAS Link #n Configuration** (see note) | 1 |
| BSYNC | **TCAS Link #n Configuration** (see note) | X |
| FTHRES[6:0] | **TCAS Framing Bit Threshold** (0x408) | 0x1F |
| 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 0x2FC corresponding to the link being configured for channelized E1 operation. The BSYNC bit is present only in registers with $0 \le n \le 2$. The BSYNC value has no significance for the channelized 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 channelized E1 operation. The BSYNC bit is present only in registers with $0 \le n \le 2$. The BSYNC value has no significance for the channelized mode of operation.

- The FTHRES[6:0] value assumes a SYSCLK of 33Mhz. For other values of SYSCLK the framing threshold value is $1.5 \cdot f_{SYSCLK}/1544000$ (This value ensures the threshold is suitable for both T1 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. The programmer should ensure that these values are suitable for all links attached to a FREEDM.

### 6.3 Unchannelized Links With Byte Synchronization

The bit stream of the unchannelized receive link that is attached to the RD[n] input is sampled on rising edges of the RCLK[n] input (see figure 13). 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 unchannelized transmit link is attached to the TD[n] output and is driven on falling edges of the TCLK[n] input (see figure 14). 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.

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

**Fig. 13 Unchannelized Receive Link Timing With Byte Synchronization**

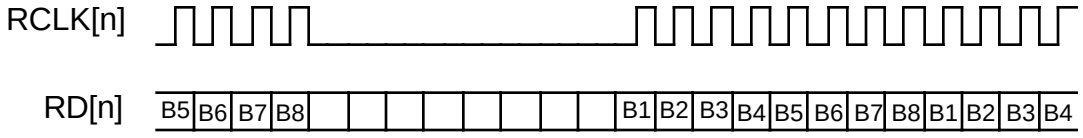
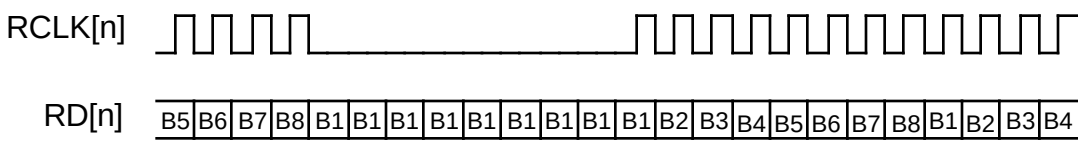


**Fig. 14 Unchannelized Transmit Link Timing With Byte Synchronization**




To configure the RCAS and TCAS to interface to byte sychronized unchannelized links the following bits are written:

| Bit | Register | Value |
|---|---|---|
| CEN | **RCAS Link #n Configuration** (see note) | 0 |
| E1 | **RCAS Link #n Configuration** (see note) | 0 |
| BSYNC | **RCAS Link #n Configuration** (see note) | 1 |
| FTHRES[6:0] | **RCAS Framing Bit Threshold** (0x108) | (see note) |
| CEN | **TCAS Link #n Configuration** (see note) | 0 |
| E1 | **TCAS Link #n Configuration** (see note) | 0 |
| BSYNC | **TCAS Link #n Configuration** (see note) | 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 0x2FC corresponding to the link being configured. The unchannelized 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 0x180 through 0x2FC corresponding to the link being configured. The unchannelized link with byte synchronization is only supported in registers with $0 \leq n \leq 2$.

- The FTHRES[6:0] value 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. The programmer should ensure that these values are suitable for all links attached to a FREEDM.

### 6.4 Unchannelized Links Without Synchronization

The bit stream of the unchannelized receive link that is attached to the RD[n] input is sampled on rising edges of the RCLK[n] input (see figure 15). The receive data is not aligned to the RCLK[n] input.

The unchannelized transmit link is attached to the TD[n] output and is driven on falling edges of the TCLK[n] input (see figure 16). 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.

**Fig. 15 Unchannelized Receive Link Timing Without Synchronization**

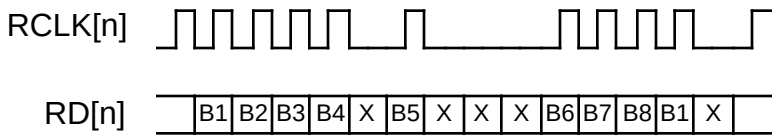
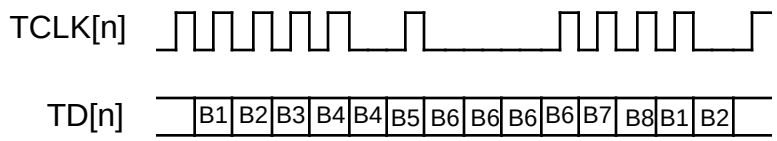



**Fig. 16 Unchannelized Transmit Link Timing Without Synchronization**




To configure the RCAS and TCAS to interface to unchannelized links the following bits are written:

| Bit | Register | Value |
|---|---|---|
| CEN | **RCAS Link #n Configuration** (see note) | 0 |
| E1 | **RCAS Link #n Configuration** (see note) | 0 |
| BSYNC | **RCAS Link #n Configuration** (see note) | 0 |
| CEN | **TCAS Link #n Configuration** (see note) | 0 |
| E1 | **TCAS Link #n Configuration** (see note) | 0 |
| BSYNC | **TCAS Link #n Configuration** (see note) | 0 |
| 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 0x2FC corresponding to the link being configured for unchannelized operation.
- The **TCAS Link #n Configuration** register must be chosen from the range 0x480 through 0x4FC corresponding to the link being configured for unchannelized 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) register has no effect on operation of an unchannelized link with BSYNC low, or on links with no BSYNC bit.

# 7 CONFIGURING THE PCI INTERFACE

Configuration of the PCI interface involves initialization of data structures, which is covered in section 3, and mapping of the Normal Mode Register Space, which is covered in section 9.1. This section covers configuration and control of the DMA activities. These are accessible within the RMAC, TMAC and GPIC block registers.

## 7.1 Configuring the Receive DMA Controller (RMAC)

The RMAC is the DMA controller which writes receive data into packet memory. It sources data from the RHDL and requests the GPIC to write the data across the PCI bus and into packet memory.

The RMAC 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) | 11B |
| RPQ_RDYN[2:0] | **RMAC Control** (0x280) | 000B |
| RPQ_LFN[1:0] | **RMAC Control** (0x280) | 00B |
| RPQ_SFN[1:0] | **RMAC Control** (0x280) | 00B |

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

Activation of the RMAC

By default, the RMAC 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 RMAC does not accept data from the RHDL and does not write data to host memory. |
| 1 | The RMAC accepts data from the RHDL and writes it to host memory. |

Free Buffer Cache Enable

The FREEDM reads from packet memory to obtain unused receive buffers and stores them in a cache if it is enabled. The access can read just one RPDR, or six RPDR's if the cache is enabled. There is a seperate cache for the small buffers and the large buffers which can be individualy enabled.

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

Raw Data Notification

The RAWMAX[1:0] field determines notification of receive occurs. 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 RPDR's written to the RPDR Ready queue by the FREEDM before an RPQRDYI interrupt is activated. 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 indicates the number of RPDR's read from the RPDRF Large queue by the FREEDM before an RPQLFI interrupt is activated. 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 indicates the number of RPDR's read from the RPDRF Small queue by the FREEDM before an RPQSFI interrupt is activated. It essentially controls the frequency of RPQSFI interrupts. When this interrupt occurs the software must replenish the RPDRF Small queue with large buffers. Valid values are:

| RPQ_SFN[1:0] | No of Reads |
|---|---|
| 00 | 1 |
| 01 | 4 |
| 10 | 8 |
| 11 | Reserved |

### 7.2 Configuring the Transmit DMA Controller (TMAC)

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

The TMAC 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) | 000B |
| TDQ_FRN[1:0] | **TMAC Control** (0x300) | 00B |

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

Activation of the TMAC

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

Free Buffer Cache Enable

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

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

TDQFI Interrupt Frequency

The TDQ_FRN[1:0] field sets the number of TDR's read from the cache and written to the TDR Free queue by the FREEDM before a TDQFI interrupt 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 RPDR Ready queue in order to confirm a transmit packet was transmitted, and so that the buffers can be reused. Valid values are:

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

### 7.3 Configuring the General Purpose PCI Controller (GPIC)

The GPIC provides the interface to a 32-bit PCI bus operating at up to 33 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 RMAC or TMAC are translated into PCI bus activity by the GPIC. Except for the PCI Configuration Space registers and parity checking, the GPIC does not perform operations on the PCI bus data.

The GPIC is configured by programming bits within the **GPIC Control** (0x040) register. The default configuration is as follows:

| Bit | Register | Value |
|---|---|---|
| Reserved | **GPIC Control** (0x040) | 0 |
| LENDIAN | **GPIC Control** (0x040) | 1 |
| SOE_E | **GPIC Control** (0x040) | 0 |
| PONS_E | **GPIC Control** (0x040) | 0 |
| RPWTH[4:0] | **GPIC Control** (0x040) | 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 encoding of this bit is:

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

Big Endian Format

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

Little Endian Format

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

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 at the PCI bus interface.

Threshold for Early Bus Arbitration

The Receive Packet Write Threshold (RPWTH[4:0]) bits control early arbitration for the PCI bus. When RPWTH[4:0] is non-zero the GPIC begins requesting access to the PCI bus when the number of dwords specified by RPWTH[4:0] is available from the RMAC.

# 8 HDLC AND CHANNEL FIFO CONFIGURATION

The FREEDM processes the data stream in the receive direction via the RHDL block and it processes the data stream in the transmit direction via the THDL block. Each of these blocks must be configured via the Normal Mode Register Space.

## 8.1 Configuring the RHDL

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

| Bit | Register | Value |
|---|---|---|
| MAX[15:0] | **RHDL Maximum Packet Length** (0x224) | 0xFFFF |
| LENABRT | **RHDL Configuration** (0x220) | 0 |
| TSTD | **RHDL Configuration** (0x220) | 0 |
| Reserved[2:0] | **RHDL Configuration** (0x220) | 0x7 |

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

Maximum Packet Length

The RHDL may be configured to abort packets which exceed a maximum length of $n$, where $0 \le n \le 0xFFFF$. The following bits are written to enable or disable this feature:

| LENABRT | MAX[15:0] | Function |
|---|---|---|
| 1 | $n$ | Enables aborts when the maximum packet length is exceeded. |
| 0 | X | Disables checking of maximum packet length. |

Datacom/Telecom Bit Order

The RHDL 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 is the first HDLC bit received. |
| 1 | Telecom standard - most significant bit is the first HDLC bit received. |

### 8.2 Configuring the THDL

The THDL 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[2:0] | **THDL Configuration** (0x3B0) | 0 |
| 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[2:0] and data is formatted in datacom bit ordering.

Enabling Burst DMA Transfer

The THDL may be configured to combine XFER[2:0] sized block transfers into a single burst DMA transfer. The following bits are written to enable or disable this feature:

| BURST[2:0] | BURSTEN | Function |
|---|---|---|
| (0 through 7 is valid) | 1 | The THDL may combine several XFER[2:0] sized amounts in a single DMA transaction. BURST[2:0] configures the maximum number of blocks that can be burst in a single DMA transaction, where BURST[2:0]=0 is one block and BURST[2:0]=7 is eight blocks. |
| X | 0 | DMA transactions are not greater than XFER[2:0] in size. |

Datacom/Telecom Bit Order

The THDL may be configured to reverse the order of bits within each 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 is the first HDLC bit transmitted. |
| 1 | Telecom standard - most significant bit is the first HDLC bit transmitted. |

BIT8

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

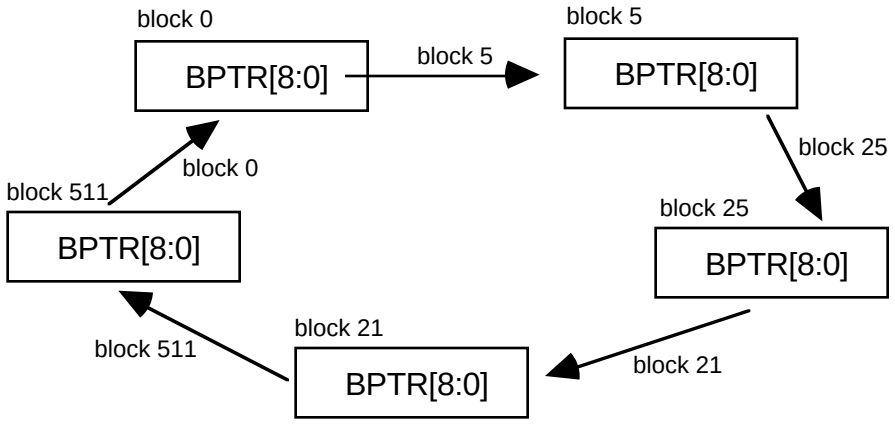
| BIT8 | Function |
|---|---|
| 0 | Channels configured for 7BIT will transmit a zero on the least significant bit of each octet. |
| 1 | Channels configured for 7BIT will transmit a one on the least significant bit of each octet. |

### 8.3 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 512 blocks (8 KBytes) available to assign among the receive channels, and another 512 blocks (8 KBytes) available to assign among the transmit channels.

A FIFO is created by assigning a circular linked list of blocks as shown in figure 17. This shows a channel FIFO consisting of 5 blocks. The quantity of buffers and the arrangement of links is choosen 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.

**Fig. 17 Specifying a Channel FIFO**



### 8.3.1 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[8:0] | **RHDL Indirect Block Data** (0x214) | (0 through 0x1FF is valid) |

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

| Bit | Register | Value |
|---|---|---|
| BUSY | **RHDL Indirect Block Select** (0x210) | X |
| BRWB | **RHDL Indirect Block Select** (0x210) | 0 |
| BLOCK[8:0] | **RHDL Indirect Block Select** (0x210) | (0 through 0x1FF is valid) |

### 8.3.2 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[8:0] | **THDL Indirect Block Data** (0x3A4) | (0 through 0x1FF is valid) |

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

| Bit | Register | Value |
|---|---|---|
| BUSY | **THDL Indirect Block Select** (0x3A0) | X |
| BRWB | **THDL Indirect Block Select** (0x3A0) | 0 |
| BLOCK[8:0] | **THDL Indirect Block Select** (0x3A0) | (0 through 0x1FF is valid) |

### 8.4 RHDL Channel Configuration

The RHDL 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) |
| CRC[1:0] | **RHDL Indirect Channel Data Register #1** (0x204) |
| XFER[2:0] | **RHDL Indirect Channel Data Register #2** (0x208) |
| OFFSET[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) |

Delineation

The data bits from the RCAS 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 directly to the Partial Packet Buffer without CRC verification. |
| 1 | Data is processed for flag sequence delineation, bit de-stuffing and CRC verification. |

Strip FCS Bit

The indirect frame check sequence bit (STRIP) enables the RHDL to remove the FCS data before writing to the channel FIFO. 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. |

CRC Algorithm

The RHDL 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 |
| 00B | 1 | No CRC verification |
| 01B | 1 | CRC-CCITT verification |
| 10B | 1 | CRC-32 verification |
| 11B | 1 | Reserved |

DMA Transfer Size

The channel transfer size is required to limit the size of data read from the Partial Packet Buffer during a single transaction. The channel transfer size also specifies the amount of data read from the Partial Packet Buffer, when the buffer does not contain an end of packet. After the transfer size is filled in the channel FIFO the RMAC will attempt to DMA the data across the PCI bus to the RAM. 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[2:0] | Function |
|---|---|
| (valid values are 0 through 7) | Specifies the data transfer size in blocks, where blocks = XFER[2:0] +1, and there are 16 bytes per block. |

Insertion of Offset Bytes

The RHDL 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 |
|---|---|
| 00B | RHDL does not insert offset bytes |
| 01B | RHDL inserts 1 offset byte per packet |
| 10B | RHDL inserts 2 offset bytes per packet |
| 11B | RHDL inserts 3 offset bytes per packet |

HDLC Data Inversion

The INVERT bit configures the RHDL to logically invert the incoming HDLC stream from the RCAS 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 Signalling

The 7BIT enable configures the RHDL 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. |

### 8.5 THDL Channel Configuration

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

| Bit | Register |
|---|---|
| DELIN | **THDL Indirect Channel Data Register #1** (0x384) |
| IDLE | **THDL Indirect Channel Data Register #1** (0x384) |
| CRC[1:0] | **THDL Indirect Channel Data Register #1** (0x384) |
| FLEN[8:0] | **THDL Indirect Channel Data Register #2** (0x384) |
| DFCS | **THDL Indirect Channel Data Register #2** (0x384) |
| INVERT | **THDL Indirect Channel Data Register #2** (0x384) |
| PRIORITYB | **THDL Indirect Channel Data Register #2** (0x384) |
| 7BIT | **THDL Indirect Channel Data Register #2** (0x384) |
| XFER[2:0] | **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) |
| TRANS | **THDL Indirect Channel Data Register #3** (0x38C) |

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 | Packet data is written directly to the outgoing data stream without processing. |
| 1 | Data is processed for flag sequence insertion, bit stuffing and CRC generation before being transmitted on the outgoing data stream. |

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

CRC Algorithm

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

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

Channel FIFO Length

The FLEN[8:0] field specifies number of blocks in the circular linked list of blocks for the channel being provisioned.

| FLEN[8:0] | Function |
|---|---|
| (valid values are 0 through 511) | Specifies the Channel FIFO size in blocks, where blocks =FLEN[8:0] +1, and there are 16 bytes per block. |

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 is filled to the expedite level, 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 is filled to the expedite level, and the last byte of the packet has not been placed into the Channel FIFO. |
| 1 | The channel is inhibited from making expedited DMA requests. It has lower priority than channels with PRIORITYB=1 when the channel with PRIORITYB=1 has a partial packet in its Channel FIFO and the Channel FIFO is at an expedited level. |

Robbed Bit Signalling

The least significant stuff enable bit (7BIT) configures the THDL to stuff the least significant bit of each octet assigned to the transmit channel in the corresponding transmit link.

| 7BIT | Function |
|---|---|
| 0 | Stuffing is disabled |
| 1 | Stuffing is enabled |

DMA Transfer Size

The channel transfer size is required to limit the size of data read from packet memory and copied into the Channel FIFO during a single transaction. The channel transfer size also specifies the maximum amount of data read from a transmit buffer during a DMA transaction. 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[2:0] | Function |
|---|---|
| (valid values are 0 through 7) | Specifies the data transfer size in blocks, where blocks = XFER[2:0] +1, and there are 16 bytes per block. |

**NOTE:** To prevent lockup the transfer size should be less than or equal to the start transmission level set by the LEVEL[3:0] and TRANS fields. Alternatively, the channel transfer size can be set, such that the total number of blocks in the Channel FIFO minus the start transmission level is an integer multiple of the channel transfer size.

Specifying The Number of Flag or Idle Bytes Inserted Between Frames

The THDL can be configured to insert either flag or idle bytes into the data stream between HDLC packets. The number of these is programmed as follows:

| FLAG[2:0] | Function |
|---|---|
| 000B | 1 flag (or 0 idle bytes) is inserted |
| 001B | 2 flags (or 0 idle bytes) is inserted |
| 010B | 4 flags (or 2 idle bytes) is inserted |
| 011B | 8 flags (or 6 idle bytes) is inserted |
| 100B | 16 flags (or 14 idle bytes) is inserted |
| 101B | 32 flags (or 30 idle bytes) is inserted |
| 110B | 64 flags (or 62 idle bytes) is inserted |
| 111B | 128 flags (or 126 idle bytes) is inserted |

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

The expedite trigger level specifies the Channel FIFO minimum empty space in order for the channel to request expedited DMA transactions. When there as many or more than the expedite level of empty blocks in the Channel FIFO the channel may request expedited DMA, provided the PRIORITYB bit is set.

The THDL starts to transmit a packet when the Channel FIFO empty space is less than or equal to the start transmission level.

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

| LEVEL[3:0] | Expedite 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<br>(256 bytes free) | 12 Blocks<br>(192 bytes free) | 8 Blocks<br>(128 bytes free) |
| 0111 | 24 Blocks<br>(384 bytes free) | 16 Blocks<br>(256 bytes free) | 12 Blocks<br>(192 bytes free) |
| 1000 | 32 Blocks<br>(512 bytes free) | 24 Blocks<br>(384 bytes free) | 16 Blocks<br>(256 bytes free) |
| 1001 | 48 Blocks<br>(768 bytes free) | 32 Blocks<br>(512 bytes free) | 24 Blocks<br>(384 bytes free) |
| 1010 | 64 Blocks<br>(1 Kbytes free) | 48 Blocks<br>(768 bytes free) | 32 Blocks<br>(512 bytes free) |
| 1011 | 96 Blocks<br>(1.5 Kbytes free) | 64 Blocks<br>(1 Kbytes free) | 48 Blocks<br>(768 bytes free) |
| 1100 | 128 Blocks<br>(2 Kbytes free) | 96 Blocks<br>(1.5 Kbytes free) | 64 Blocks<br>(1 Kbytes free) |
| 1101 | 192 Blocks<br>(3 Kbytes free) | 128 Blocks<br>(2 Kbytes free) | 96 Blocks<br>(1.5 Kbytes free) |
| 1110 | 256 Blocks<br>(4 Kbytes free) | 192 Blocks<br>(3 Kbytes free) | 128 Blocks<br>(2 Kbytes free) |
| 1111 | 384 Blocks<br>(6 Kbytes free) | 256 Blocks<br>(4 Kbytes free) | 192 Blocks<br>(3 Kbytes free) |

# 9 FREEDM OPERATIONAL PROCEDURES

## 9.1 Device Identification, Location and System Resource Assignment

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

Identifying and Locating a FREEDM

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 5.1 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 Device ID (0x7364) 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 specifies a revision identifier within the REVID[7:0] field which may be useful to distinguish between future revisions of the FREEDM.

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 value read back determines the size of memory to assign to the base address register. The software scans the value starting from the most significan bit to determine the first 0 bit. For example, a device that wants a 1 M address space would build the top 12 bits and hardwire the others to zero. The four least significant bits are read-only and are not used to determine the memory requirement.

For a FREEDM 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 4K bytes.

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 onto the PCI Bus

Following assignment of the memory base address the software must enable the FREEDM to respond to PCI memory accesses and to participate on the PCI bus as a bus master. Additionally, the FREEDM can be enabled to report system and parity errors. The 16-bit **Command** (0x6) register of the PCI Configuration Space is written as follows:

| Bit | Word Sized Configuration Register | Value |
|---|---|---|
| MCNTRL | **Command** (0x6) | 1 |
| MSTREN | **Command** (0x6) | 1 |
| PERREN | **Command** (0x6) | 1 |
| SERREN | **Command** (0x6) | 1 |

Initializing the Configuration Space Registers

In addition to enabling the FREEDM onto the PCI bus the following register bits must also be initialized.

| Bit | Byte Sized Configuration Register | Value |
|---|---|---|
| CLSIZE[7:0] | **Cache Line Size** (0xC) | (see note) |
| LT[7:0] | **Latency Timer** (0xD) | (see note) |
| INTLIN[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 uses the memory read/write multiple command if the data transfer size is greater than the cache line size. It will use the memory read/write 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/write 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 RMAC and TMAC 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.

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

### 9.2 Reset

This section describes the procedure to reset the FREEDM via software. The FREEDM 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 are:

1) If the FREEDM was active before the reset procedure then the deactivation procedure must be done. (see section 9.5).

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

This reset procedure has the following effects:

- If the RESET bit is logic one, the entire FREEDM 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 out of reset. Holding the FREEDM 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 Configuration Space 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.

### 9.3 Initialization

This section describes the procedure to initialize the FREEDM. 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 3.

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

The steps to initialize a FREEDM 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 3.1 and 3.6.

2) Assign start, read, write and end indexes for all of the queues. The register accesses are described in section 3.6.

3) Configure the serial links. The register accesses are described in section 6.

4) Configure the GPIC interface. The register accesses are described in section 7.3.

5) Configure HDLC processing of the RHDL and the THDL blocks. The register accesses are described in section 8.1 and 8.2.

### 9.4 Activation Procedure

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

The activation procedure normally follows the initialization procedure.

The steps to activate a FREEDM are:

1) Enable interrupt 'E' bits as described in section 4.

2) Enable the FREEDM DMA activity by setting the ENABLE bits of the RMAC and TMAC as described in sections 7.1 and 7.2.

3) The SYSCLKA and the TDBA in the **FREEDM Master Clock / 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. A low value indicates a failure in clocking that is provided at the SYSCLK input pin of the FREEDM.

6) The TLGA[7:0] and the RLGA[7:0] bits in the **FREEDM 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 should be read high. A low value indicates a failure in clocking that is provided at the TCLK or RCLK input pins.[1]

### 9.5 Deactivation Procedure

The deactivation procedure is required to place the FREEDM 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 was actively transfering packets, or to gracefully shut down the FREEDM.

The steps to deactivate a FREEDM are:

1) Disable interrupt 'E' bits as described in section 4.

[1]Each RLGA[7:0] or TLGA[7:0] bit corresponds to a group of 4 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.

2) Disable the FREEDM DMA activity by programming the ENABLE bits to zero in the RMAC and TMAC as described in sections 8.1 and 8.2.

3) Continue by performing the software reset procedure.

### 9.6 Provisioning a Channel

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

#### 9.6.1 Receive Channel Provisioning

The steps to provision a receive channel $RCC$, where $0 \le RCC \le 127$ are:

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

| Bit | Register | Value |
|---|---|---|
| CHAN[6:0] | **RCAS Channel Disable** (0x10C) | $RCC$ |
| CHDIS | **RCAS Channel Disable** (0x10C) | 1 |

2) Program the Channel FIFO as described in section 8.3.1.

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 8.4. In writing the **RHDL Indirect Channel Data Register #1**, ensure the PROV bit is set, and ensure the FPTR[8:0] bits identify a block within the circular linked list of buffers of step 2.

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

| Bit | Register | Value |
|---|---|---|
| CHAN[6:0] | **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 RCAS channel that is provisioned. Write the following register:

| Bit | Register | Value |
|---|---|---|
| CHAN[6:0] | **RCAS Indirect Channel Data** (0x104) | *RCC* |
| PROV | **RCAS Indirect Channel Data** (0x104) | 1 |
| CDLBEN | **RCAS Indirect Channel Data** (0x104) | 0 |

8) For a **channelized** 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] is 1 through 24 for a T1 link, and 1 through 31 for an E1 link. For an **unchannelized** 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 ensures that the indirect RAM has been updated.

| Bit | Register | Value |
|---|---|---|
| BUSY | **RCAS Indirect Link and TIme-slot Select** (0x100) | X |
| 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 is valid) |
| RWB | **RCAS Indirect Link and TIme-slot Select** (0x100) | 0 |

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

| Bit | Register | Value |
|---|---|---|
| CHAN[6:0] | **RCAS Channel Disable** (0x10C) | *RCC* |
| CHDIS | **RCAS Channel Disable** (0x10C) | 0 |

**Warning**:

- The programmer must ensure 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.

- 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.

### 9.6.2 Transmit Channel Provisioning

The steps to provision a transmit channel $TCC$, where $0 \le TCC \le 127$ are:

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

| Bit | Register | Value |
|---|---|---|
| DCHAN[6:0] | **TCAS Channel Disable** (0x410) | $TCC$ |
| CHDIS | **TCAS Channel Disable** (0x410) | 1 |

2) Program the Channel FIFO as described in section 8.3.2 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 8.4. In writing the **RHDL Indirect Channel Data Register #1**, ensure the PROV bit is set, and ensure the FPTR[8:0] bits identify a block within the circular linked list of buffers of step 2.

5) Specify the THDL 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[6:0] | **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 TCAS channel to provision. Write the following register:

| Bit | Register | Value |
|---|---|---|
| CHAN[6:0] | **TCAS Indirect Channel Data** (0x404) | $TCC$ |
| PROV | **TCAS Indirect Channel Data** (0x404) | 1 |

8) For a **channelized** 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] is 1 through 24 for a T1 link, and 1 through 31 for an E1 link. For an **unchannelized** 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 ensures that the indirect RAM has been updated.

| Bit | Register | Value |
|---|---|---|
| BUSY | **TCAS Indirect Link and TIme-slot Select** (0x400) | X |
| 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 is valid) |
| RWB | **TCAS Indirect Link and TIme-slot Select** (0x400) | 0 |

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

| Bit | Register | Value |
|---|---|---|
| CHAN[6:0] | **TMAC Indirect Channel Provisioning** (0x304) | *TCC* |
| PROV | **TMAC Indirect Channel Provisioning** (0x304) | 1 |
| RWB | **TMAC Indirect Channel Provisioning** (0x304) | 0 |
| BUSY | **TMAC Indirect Channel Provisioning** (0x304) | X |

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

| Bit | Register | Value |
|---|---|---|
| CHAN[6:0] | **TCAS Channel Disable** (0x410) | *TCC* |
| CHDIS | **TCAS Channel Disable** (0x410) | 0 |

**Warning**:

- The programmer must ensure 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.

- 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.

### 9.7 Unprovisioning a Channel

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

#### 9.7.1 Receive Channel Unprovisioning

The steps to unprovision a receive channel $RCC$, where $0 \le RCC \le 127$ are:

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

| Bit | Register | Value |
|---|---|---|
| CHAN[6:0] | **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 RCAS channel to unprovisioned. Write the following register:

| Bit | Register | Value |
|---|---|---|
| CHAN[6:0] | **RCAS Indirect Channel Data** (0x104) | *RCC* |
| PROV | **RCAS Indirect Channel Data** (0x104) | 0 |
| CDLBEN | **RCAS Indirect Channel Data** (0x104) | X |

4) For a **channelized** link, specify the time-slots which are being unassigned on this channel by writing the following register once for each time-slot that is unassigned. Valid values for TSLOT[4:0] is 1 through 24 for a T1 link, and 1 through 31 for an E1 link. For an **unchannelized** 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 ensures that the indirect RAM has been updated.

| Bit | Register | Value |
|---|---|---|
| BUSY | **RCAS Indirect Link and TIme-slot Select** (0x100) | X |
| 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 is valid) |
| RWB | **RCAS Indirect Link and TIme-slot Select** (0x100) | 0 |

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 RHDL 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[6:0] | **RHDL Indirect Channel Select** (0x200) | *RCC* |
| CRWB | **RHDL Indirect Channel Select** (0x200) | 1 |
| BUSY | **RHDL Indirect Channel Select** (0x200) | X |

7) Read the RHDL 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, and keeping the same FPTR[8:0] bits.

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

| Bit | Register | Value |
|---|---|---|
| CHAN[6:0] | **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.

11) Ensure 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 it is low before proceeding with step 12.

| Bit | Register | Value |
|---|---|---|
| CHAN[6:0] | **RMAC Indirect Channel Provisioning** (0x284) | *RCC* |
| PROV | **RMAC Indirect Channel Provisioning** (0x284) | 0 |
| RWB | **RMAC Indirect Channel Provisioning** (0x284) | 0 |
| BUSY | **RMAC Indirect Channel Provisioning** (0x284) | X |

12) Enable FREEDM processing of the unprovisioned channel. Write the following bits:

| Bit | Register | Value |
|---|---|---|
| CHAN[6:0] | **RCAS Channel Disable** (0x10C) | *RCC* |
| CHDIS | **RCAS Channel Disable** (0x10C) | 0 |

**Warning**:

- 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 100 msec through 1 msec.

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

### 9.7.2 Transmit Channel Unprovisioning

The steps to unprovision a transmit channel $TCC$, where $0 \le TCC \le 127$ are:

1) Unprovision the TMAC 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[6:0] | **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 the previous indirect RAM access has completed and that a new indirect RAM access can be started.

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

| Bit | Register | Value |
|---|---|---|
| DCHAN[6:0] | **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 TCAS channel to unprovision. Write the following register:

| Bit | Register | Value |
|---|---|---|
| CHAN[6:0] | **TCAS Indirect Channel Data** (0x404) | *TCC* |
| PROV | **TCAS Indirect Channel Data** (0x404) | 0 |

6) For a **channelized** link, specify the time-slots to be unassigned by writing the following register once for each time-slot. Valid values for TSLOT[4:0] is 1 through 24 for a T1 link, and 1 through 31 for an E1 link. For an **unchannelized** 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 ensures that the indirect RAM has been updated.

| Bit | Register | Value |
|---|---|---|
| BUSY | **RCAS Indirect Link and TIme-slot Select** (0x100) | X |
| 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 is valid) |
| RWB | **RCAS Indirect Link and TIme-slot Select** (0x100) | 0 |

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

| Bit | Register | Value |
|---|---|---|
| CHAN[6:0] | **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 Register #1** (0x384). Then write this register with PROV modified to zero, and keeping the same FPTR[8:0] bits.

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

| Bit | Register | Value |
|---|---|---|
| CHAN[6:0] | **THDL Indirect Channel Select** (0x380) | *TCC* |
| CRWB | **THDL Indirect Channel Select** (0x380) | 0 |
| BUSY | **THDL Indirect Channel Select** (0x380) | X |

11) Enable FREEDM processing of the channel. Write the following bits:

| Bit | Register | Value |
|---|---|---|
| CHAN[6:0] | **TCAS Channel Disable** (0x410) | *TCC* |
| CHDIS | **TCAS Channel Disable** (0x410) | 0 |

**Warning**:

- 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 100 msec through 1 msec.

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

### 9.8 Transmit Sequence

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

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

2) The software writes one or more TDR's to the TDR Ready queue as per the WriteQueue() routine of section 3.6. Each TDR that is written to the TDR Ready queue points to the first TD in chain of TD's which describe the packet(s).

3) The software writes the **TMAC Transmit Descriptor Reference Ready Queue Write** (0x23C) register during the WriteQueue() routine. In response to a change in this register the FREEDM reads packet memory at the TDR Ready queue locations that were written by the software.

4) The TMAC links the first TDR of each packet read from the TDR Ready queue to a linked list maintained by the TMAC. The TMAC linked list is maintained on a channel basis. The TMAC writes the TMAC Next TD Pointer[13:0] field and the V bit field within the first TD of the last packet in the TMAC linked list. There is no linking if the TMAC linked list is empty.

5) The TMAC reads and transmits buffer data according to the Host linked list and the TMAC 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 TMAC 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 TDR's 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. When the TDR cache is full (ie the 6'th TDR has been processed) all TDR's are written to the TDR Free queue.

7) The embedded processor is interrupted if either of the following has occured:

   - one or more TDR are written to the TDR Free queue because the IOC bit was set in the last TD processed by the TMAC. The IOCI status bit is set within the **FREEDM 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 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 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 3.6. The FREEDM returns TDR's 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 occured in processing the individual data buffers of the linked TD's that form the packet.

**NOTE**:

- During the transmit channel unprovisioning procedure a TDR that is being processed by the FREEDM and that belong 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[13:0] of each TD in the chain. It must also unlink descriptors in the host chain, based on the CE bit and the HostNext TD Pointer[13:0], of the unprovisioned TDR. This will ensure that all TD's (and data buffers) of the unprovisioned channel are returned to the host.

- The TDR cache may need to be disabled before performing the device unprovisioning procedure. This is required to ensure the FREEDM immediately places the unprovisioned TDR onto the TDR Free queue. The TDR cache is disabled by setting the CACHE bit to zero within the **TMAC Control** (0x300) register.

### 9.9 Receive Sequence

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

1) The FREEDM 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. RPDR's are read from the queue 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. RPDR's are read from the queue six at a time and stored in the cache.

2) The FREEDM generates an interrupt if the programmed number of RPDR's have been read from the RPDR Small (or Large) queue. The RPQSFI or the RPQLFI status bits of the **FREEDM Master Interrupt Status** (0x008) register indicate which queue must be replenished with free RPDR's by the software. The software can use the WriteQueue() routine of section 3.6 to replenish the queue. The frequency of the interrupt, and the number of RPDR's 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: RCC[6:0], CE, Status[5:0], Bytes in Buffer[15:0], and Next RPD Pointer[13:0].

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

5) The FREEDM 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: RCC[6:0], CE, Status[5:0], Bytes in Buffer[15:0], and Next RPD Pointer[13:0].

8) The STATUS[1:0] field of the RPDR is assigned and the first RPDR in the linked list of RPD's which describe the receive packet data is written to the RPDR Ready queue by the FREEDM.

9) The embedded processor is interrupted if the programmed number of RPDR's have been written to the RPDR Ready queue. The RPQRDYI interrupt status bit is set within the **FREEDM Master Interrupt Status** (0x008) register. The number of RPDR's 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 3.6. 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 RPD's 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 RPDR's 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[13:0] of each RPD in the chain. This will ensure that all RPD's and data buffers are returned to the RPDR Ready queue.

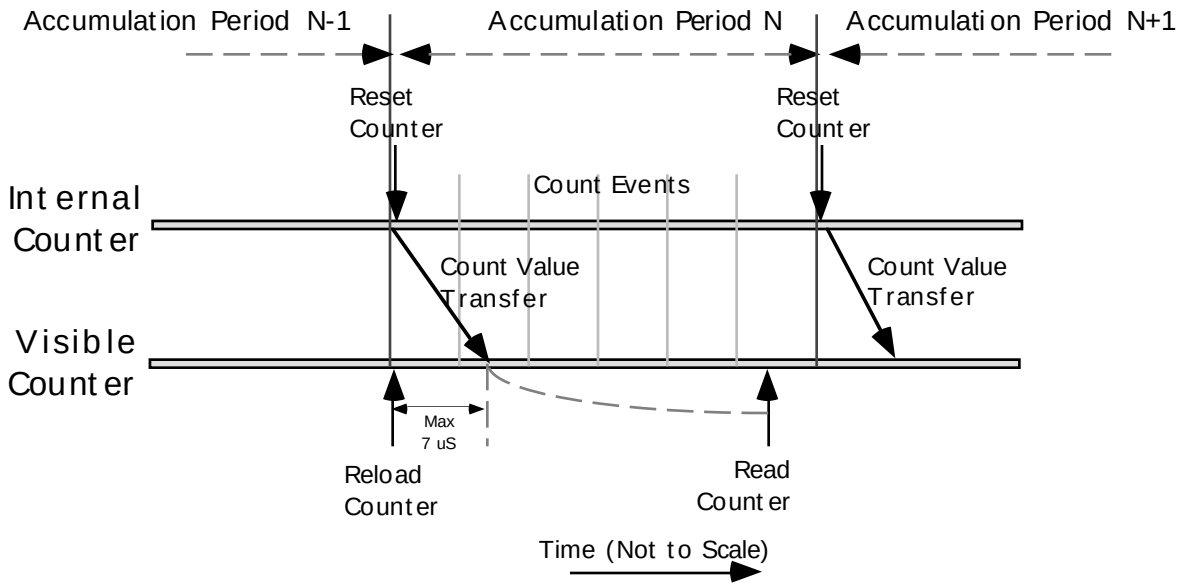
### 9.10 Performance Counters

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

| Bits | 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** (0x504) |
| C2[15:0] | **PMON Configurable Count #2** (0x504) |

The software must poll these counters to prevent overflow. figure 18 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.

**Fig. 18 Event Sequence For Polling of Counters**




The software initiates a counter reload by writing to the **FREEDM Master Clock / 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 configued during initialization. The first configurable count register is assigned by setting one of the following register bits, all other bits must be zero:

| Bits | Register |
|---|---|
| RSPE1EN | **FREEDM Master Performance Monitor Control** (0x024) |
| RFCSE1EN | **FREEDM Master Performance Monitor Control** (0x024) |
| RABRT1EN | **FREEDM Master Performance Monitor Control** (0x024) |
| RLENEIEN | **FREEDM Master Performance Monitor Control** (0x024) |
| RP1EN | **FREEDM Master Performance Monitor Control** (0x024) |
| TABRT1EN | **FREEDM Master Performance Monitor Control** (0x024) |
| TP1EN | **FREEDM Master Performance Monitor Control** (0x024) |

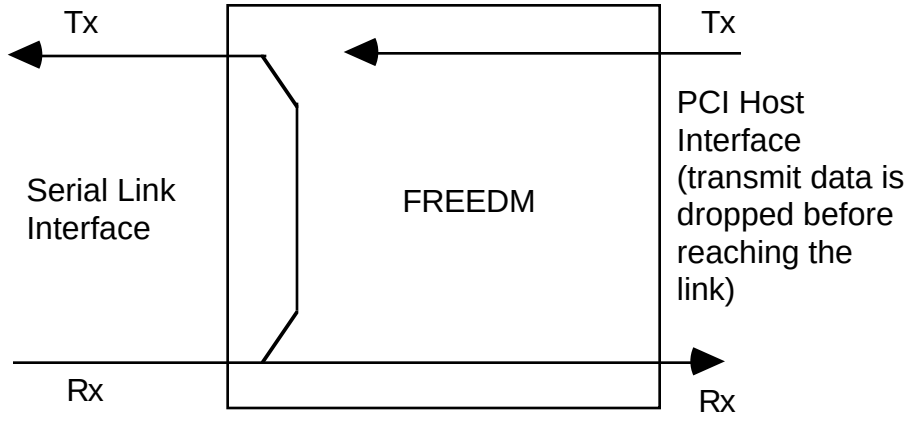
The second configurable count register is assigned by setting one of the following register bits, all other bits must be zero:

| Bits | Register |
|---|---|
| RSP2EN | **FREEDM Master Performance Monitor Control** (0x024) |
| RLEN2EN | **FREEDM Master Performance Monitor Control** (0x024) |
| RABRT2EN | **FREEDM Master Performance Monitor Control** (0x024) |
| RFCS2EN | **FREEDM Master Performance Monitor Control** (0x024) |
| RP2EN | **FREEDM Master Performance Monitor Control** (0x024) |
| TABRT2EN | **FREEDM Master Performance Monitor Control** (0x024) |
| TP2EN | **FREEDM Master Performance Monitor Control** (0x024) |

### 9.11 Line Loopback

Each serial port of the FREEDM can be placed in line loopback. In this configuration receive data from the serial link is loopbacked to the transmit serial link as illustrated in figure 19.

**Fig. 19 Line Loopback**




Each serial port 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.

| Bits | Register |
|---|---|
| LLBEN[15:0] | **FREEDM Master Link Loopback #1** (0x014) |
| LLBEN[31:16] | **FREEDM Master Link Loopback #2** (0x018) |

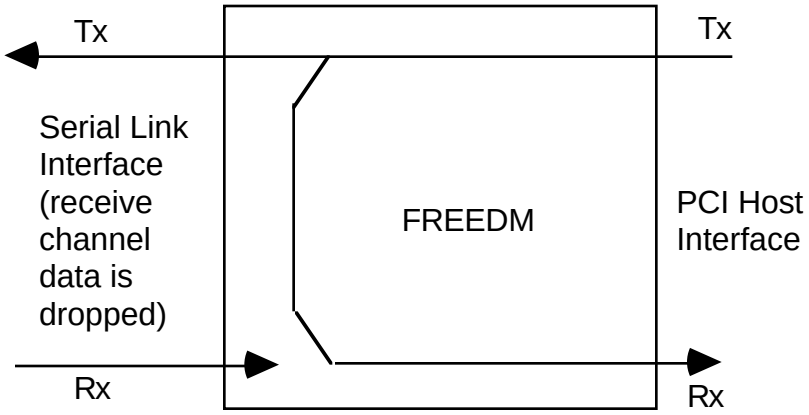
**NOTE**: The software should unprovision channels associated with the link that is placed in line loopback mode before placing the link in line loopback.This will prevent the data stream at the serial link from passing through the FREEDM to the PCI interface.

### 9.12 Diagnostic Loopback

Each channel of the FREEDM 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 20. 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 9.6, 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.

**Fig. 20 Diagnostic Loopback**




### 9.13 BERT Port

The FREEDM provides pins to transmit/receive a BERT data stream on any of the serial links. The following register is written to enable the BERT port and transmit/receive the BERT data stream on serial port $n$, where $0 \le n \le 31$.

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

The BERT port is disabled by programming the following register:

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

The TDBA bit of the **FREEDM Master Clock / 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 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 and the PCI interface.

## REFERENCES

[1] PMC-931127, PMC-Sierra, "Frame Relay Protocol Engine and Datalink Manager" Standard Product Datasheet, July, 1996, Issue 5

[2] PCI SIG, PCI Local Bus Specification, June 1, 1995, Version 2.1

## CONTACTING PMC-SIERRA

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

Telephone: 604-415-6000
Facsimile: 604-415-6200

Product Information: info@pmc-sierra.bc.ca
Applications information: apps@pmc-sierra.bc.ca

World Wide Web Site: http://www.pmc-sierra.com



PMC-970281 Issue date: March, 1997