リフレクティブメモリ **5565** シリーズ ユーザーガイド

concurrent

●このマニュアルは、VMIVME-5565,VMIPCI-5565,VMIMPC-5565,PCI5565-PIORC,PCIE-5565RC および PMC-5565PIORC Ultrahigh-Speed Fiber-Optic Reflective Memory with Interrupts を翻訳した物をコンカレント RedHawk Realtime Linux 用に開発したデバイスドライバの仕様に合わせて加筆したものです。本マニュアルとハードウェアの記載に食い違いがある場合には、Ultrahigh-Speed Fiber-Optic Reflective Memory with Interrupts の記載が優先します。

●マニュアルの内容は予告無く変更する事があります。

●本インターフェースボードは、GE Fanuc Embedded Systems 製です。また、日本での代理店は、株式会社エルエッチエスとインターニックス株式会社です。

●ここに、記載した商品名は、一般に各社の商標または登録商標です。

１.概要

リフレクティブメモリは、2～256 台のシステム間でメモリを共有するためのインターフェースボードです。

ローカルノード上のメモリに、書き込まれたデータは、瞬時に他のボード上のメモリにコピーされます。この制御は、ハードウェアで行われるため通信に関するソフトウェアのオーバヘッドがいっさいありません。

このリフレクティブメモリ・ボードは、VME,PCI,PMC に対応しています。

１．１　特徴

ハードウェアの特徴

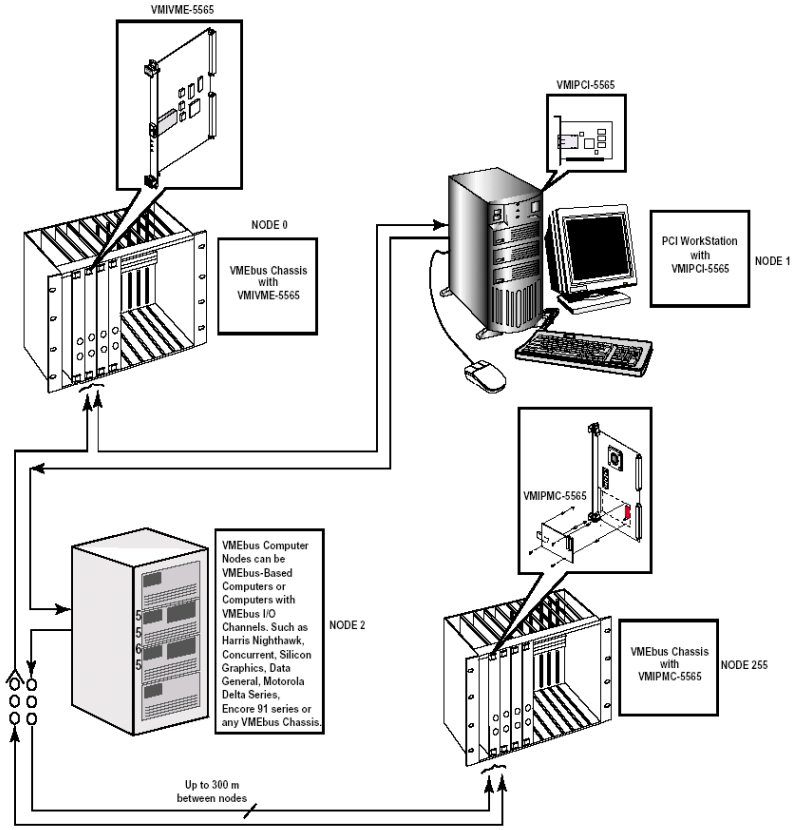
- メモリを共有するため、ソフトウェアのオーバヘッドがありません。
- 指定のノードあるいはブロードキャストで割込みを発生できます、このとき 32 ビットのデータを送信できます。
- ボード上のジャンパを設定することによって、リダンダント通信にする事ができますので、長距離でも安全に通信することが出来ます。
- プログラム入出力時のデータアクセス幅は、8,16,32 ビットアクセスが可能です。
- DMA 転送時は、4 から 64 バイトの動的なパケットを生成します。
- ノード間ケーブル長は、マルチモードファイバーでは 300m、シングルボードファイバーでは 10Km です。
- PCI 64-bit 66 MHz, 3.3 or 5 V の PCI、PCIExpress あるいは PMC ボードです。
- 4 バイトパケットでは、43MB/sec,64 バイトパケットでは 174MB/sec です。
- オンボード・メモリは、128MB です。
- 1K バイトまたは 4K バイト FIFO をオプションで選択できます。

ソフトウェアの特徴

- 独自デバイスドライバとアクセスライブラリをソースコードで供給いたします。
- コンカレント RedHawk Realtime Linux では、割込み信号を指定のプロセッサに直接入力出来ますので、高速な割込み処理が可能です。
- １つのボードを複数のプロセスあるいはスレッドで同時にアクセス出来ます。
- 2 つの DMA エンジンを、read(),と write()に独立して割り付けています。
- DIRECT-DMA 転送をサポートしていますので、ユーザ空間とリフレクティブメモリ間を直接 DMA します。
- 理論性能に近い、write 時 140M バイト/秒、read 時 170M バイト/秒の実転送速度を実現しています。

１．２　機能説明

２つのシャーシ間は、光ドライバ／レシーバをインターフェースに使用したＦＩＦＯメモリによってディジーチェイン接続されます。下図にこのリフレクティブメモリのネットワーク図を示します。



１．３　デバイスドライバとユーザソフトウェア

ユーザソフトウェアは、プログラム起動時のリフレクティブメモリ・ボードに対してボードとボードのリンクを確立するために初期化プログラムを起動（リセットを発行）する必要があります。

リフレクティブメモリ・ボードは、パワーアップ時に、すべての割込みがマスクされるように初期化します。しかし、もし割込み処理を望むならば、初期化時に SIGNAL を定義しなくてはなりません。

リフレクティブメモリ上の FAIL LED は、コントロール・ステータス・レジスタ（ＣＳＲ）の最上位ビットを示しています。FAIL LED は、ボードの状態とレジスタを結び付ける唯一のデバイスで、ソフトウェアによってオン／オフすることができます。

ボード上に自動診断プログラムは実装されていませんが、立ち上げ時にコンフィグレーション異常を検出した場合に、ドライバが FAIL LED をオンします。標準的パワーアップ・モードでは、FAIL LED は、オフされています。

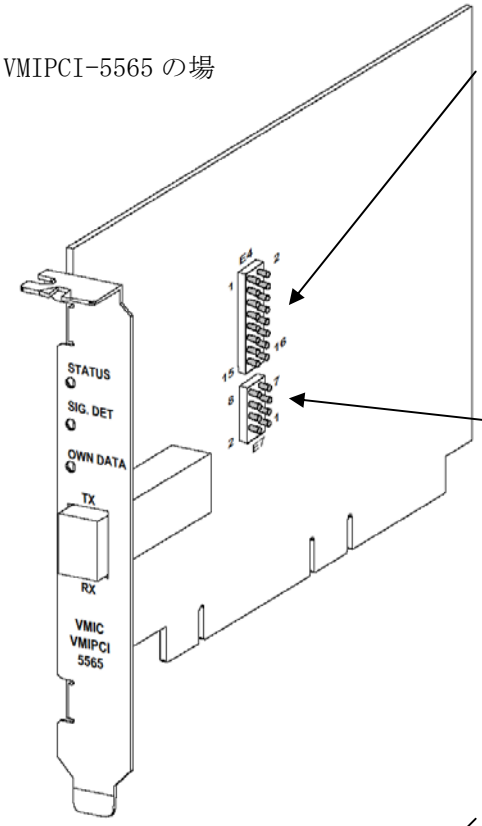
もしも、送信ＦＩＦＯの半分にデータが達し、割込みを許可するようにプログラムしていた場合、リフレクティブメモリ・ボードは、割込みを発生させます。現在のドライバプログラムは、この警告を無視します。もし完全にフルになった送信ＦＩＦＯに対して書込みを試みるとリフレクティブメモリ・ボードは、バスエラー（BERR）を返します。

１．４　ハードウェア設定条件

リフレクティブメモリ・ボードは、以下の条件に従って正しくシステムのジャンパー設定を行なわなくてはなりません。

a. リクティブメモリ・ボード上のジャンパスイッチで、各々ユニークなノードＩＤを設定しなくてはなりません。
b. ユニークなノードＩＤが保持される限りコミュニケーションバス上でのノードＩＤの順序は問いません。
c. すべてのボード上のバス速度の設定は同じになるようにして下さい。さもないと異常な結果になります。パリティなし、あるいはそのほかのハードウェアでチェックできるエラーが接続先に存在する場合でも、リフレクティブメモリボードは、その存在をユーザに報告しません。
d. それぞれのシャーシでのリフレクティブメモリ・ボードがマップされるアドレス空間は違っていてもかまいません。各々のリフレクティブメモリボードのベースアドレスから相対的に同じ位置にデータが現れます。

VMIPCI-5565 の場



VMIPMC-5565 の場

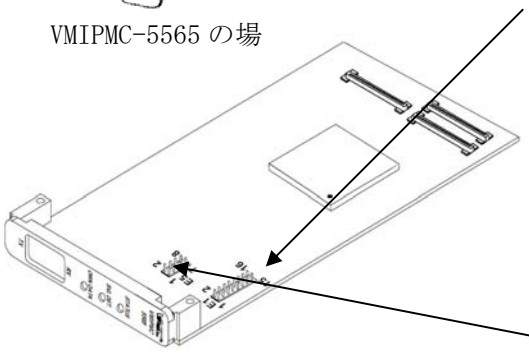


Table 2-1 Example Node ID

E4 1 to 2	E4 3 to 4	E4 5 to 6	E4 7 to 8	E4 9 to 10	E4 11 to 12	E4 13 to 14	E4 15 to 16	Node ID Hex (Dec.)
Installed	Installed	Installed	Installed	Installed	Installed	Installed	Installed	$0 (0)
Removed	Installed	Installed	Installed	Installed	Installed	Installed	Installed	$1 (1)
Installed	Removed	Installed	Installed	Installed	Installed	Installed	Installed	$2 (2)
Installed	Installed	Removed	Installed	Installed	Installed	Installed	Installed	$4 (4)
Installed	Installed	Installed	Removed	Installed	Installed	Installed	Installed	$8 (8)
Installed	Installed	Installed	Installed	Removed	Installed	Installed	Installed	$10 (16)
Installed	Installed	Installed	Installed	Installed	Removed	Installed	Installed	$20 (32)
Installed	Installed	Installed	Installed	Installed	Installed	Removed	Installed	$40 (64)
Installed	Installed	Installed	Installed	Installed	Installed	Installed	Removed	$80 (128)
Removed	Removed	Removed	Removed	Removed	Removed	Removed	Removed	$FF (255)

Table 2-2 E7 Jumper Configuration

E7 Pins	Installed/Omitted	Function/Mode Selected
1 to 2	Installed	Non-redundant (Fast) network transfer mode selected
	Omitted	Redundant network transfer mode selected
3 to 4	Installed	Rogue Master 0 function disabled
	Omitted	Rogue Master 0 function enabled
5 to 6	Installed	Rogue Master 1 function disabled
	Omitted	Rogue Master 1 function enabled
7 to 8	None	These pins are RESERVED, NO jumper shunt should be installed.

Table 2-1 Example Node ID

E1 1 to 2	E1 3 to 4	E1 5 to 6	E1 7 to 8	E1 9 to 10	E1 11 to 12	E1 13 to 14	E1 15 to 16	Node ID Hex (Dec.)
Installed	Installed	Installed	Installed	Installed	Installed	Installed	Installed	$0 (0)
Removed	Installed	Installed	Installed	Installed	Installed	Installed	Installed	$1 (1)
Installed	Removed	Installed	Installed	Installed	Installed	Installed	Installed	$2 (2)
Installed	Installed	Removed	Installed	Installed	Installed	Installed	Installed	$4 (4)
Installed	Installed	Installed	Removed	Installed	Installed	Installed	Installed	$8 (8)
Installed	Installed	Installed	Installed	Removed	Installed	Installed	Installed	$10 (16)
Installed	Installed	Installed	Installed	Installed	Removed	Installed	Installed	$20 (32)
Installed	Installed	Installed	Installed	Installed	Installed	Removed	Installed	$40 (64)
Installed	Installed	Installed	Installed	Installed	Installed	Installed	Removed	$80 (128)
Removed	Removed	Removed	Removed	Removed	Removed	Removed	Removed	$FF (255)

Table 2-2 E5 Jumper Configuration

E5 Pins	Installed/Omitted	Function/Mode Selected
1 to 2	Installed	Non-redundant (Fast) network transfer mode selected
	Omitted	Redundant network transfer mode selected
3 to 4	Installed	Rogue Master 0 function disabled
	Omitted	Rogue Master 0 function enabled
5 to 6	Installed	Rogue Master 1 function disabled
	Omitted	Rogue Master 1 function enabled
7 to 8	None	These pins are RESERVED, NO jumper shunt should be installed.

１．５　旧シリーズのノード **ID** 設定**(**スイッチ **E4** または **E1)**

ボードのノード ID は、コミュニケーションバス上の認証番号を定義します。ＩＤ番号は、コミュニケーションバスでただ一つの番号でなくてはなりません。これは、ノード０から順番にすべてのノードへコミュニケーションバスの操作を行うからです。ノードＩＤはスイッチ S6 で定義します。スイッチは ON すると０になります。このノード番号はコミュニケーションバス上の物理的な位置とは関係していませんので物理的な位置は自由です。

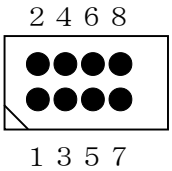
設定例

S6 POS1	**S6 POS2**	**S6 POS3**	**S6 POS4**	**S6 POS5**	**S6 POS6**	**S6 POS7**	**S6 POS8**	**Node ID Hex (Dec.)**
On	On	On	On	On	On	On	On	**$0 (0)**
Off	On	On	On	On	On	On	On	**$1 (1)**
On	Off	On	On	On	On	On	On	**$2 (2)**
On	On	Off	On	On	On	On	On	**$4 (4)**
On	On	On	Off	On	On	On	On	**$8 (8)**
On	On	On	On	Off	On	On	On	**$10 (16)**
On	On	On	On	On	Off	On	On	**$20 (32)**
On	On	On	On	On	On	Off	On	**$40 (64)**
On	On	On	On	On	On	On	Off	**$80 (128)**
Off	Off	Off	Off	Off	Off	Off	Off	**$FF (255)**

OFF(Down) = 1
ON(Up) = 0

１．６　旧シリーズのリダンダントモード設定（ジャンパ **E7** または **E5**）

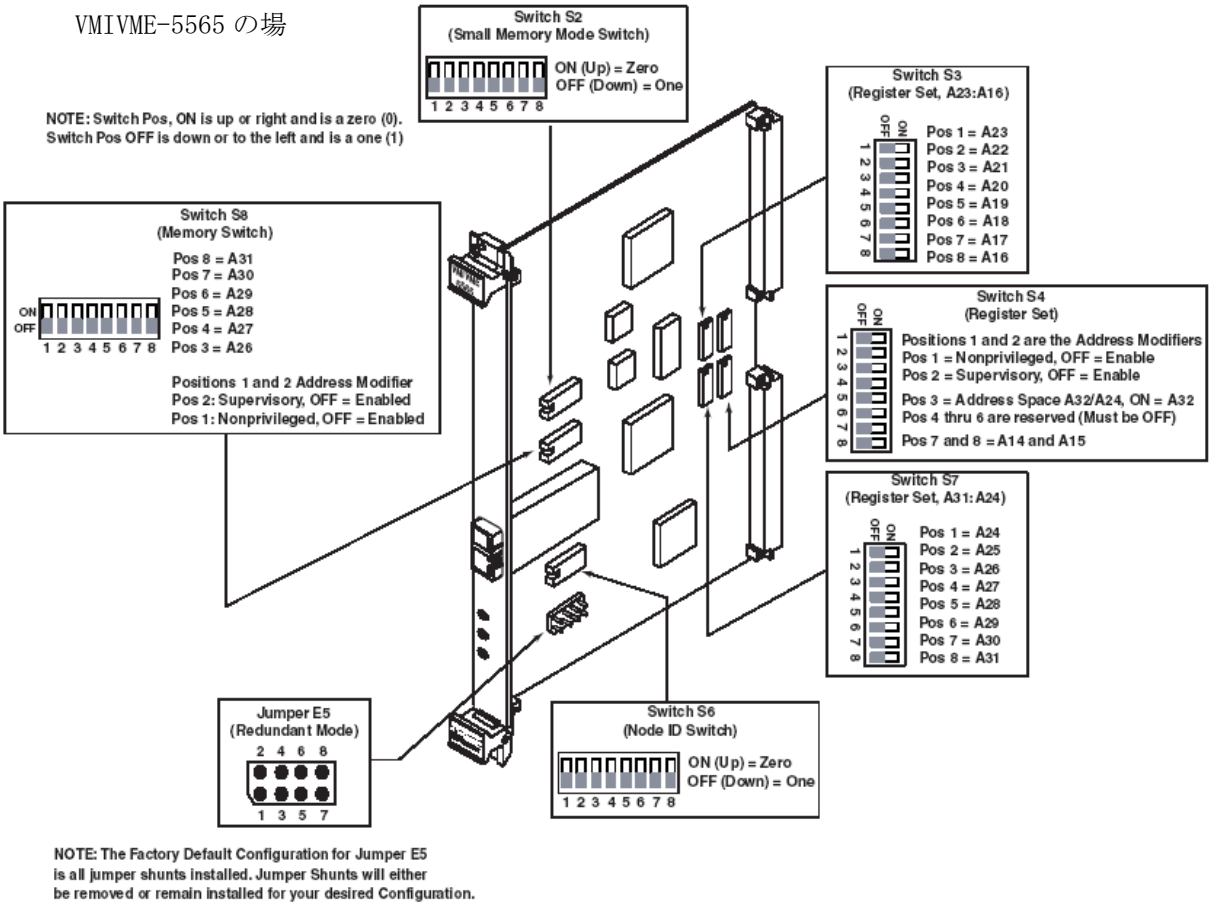
Jumper Position	**Jumper State**	**Function/Mode Selected**
1 and 2	**Installed**	Non-redundant (Fast) transfer mode
	Omitted	Redundant transfer mode
3 and 4	**Installed**	Rogue master 0 disabled
	Omitted	Rogue master 0 enabled
5 and 6	**Installed**	Rogue master 1 disabled
	Omitted	Rogue master 1 enabled
7 and 8	**Omitted**	***Reserved, Not Used***



１．７　VMEbus のベースアドレスの選択

PMCbus，PCIbus の場合には、自動的に設定されますが、VMIVME-5565 のベースアドレスは、ジャンパで設定します。一度確定すると、すべての VMIVME-5565 へのアクセスは、ベースアドレスからの相対アドレスで行われます。また接続されている各々のノードすべてのボードのベースアドレスを同じにする必要はありません。VMEbus アドレスの上の相対アドレスは、ベースアドレスからの相対オフセットのみがコミュニケーションバスに送信されます。

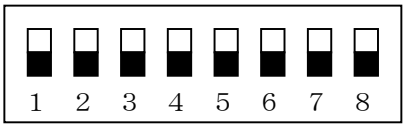
VMIVME-5565 の場



NOTE: The Factory Default Configuration for Jumper E5 is all jumper shunts installed. Jumper Shunts will either be removed or remain installed for your desired Configuration.

１．８　新シリーズのノード ID 設定(スイッチ S2)

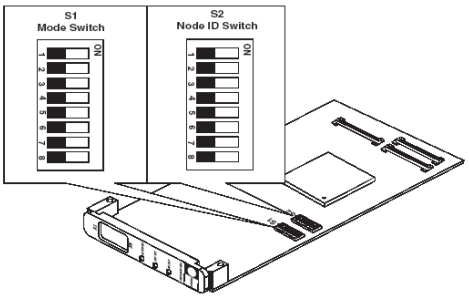
ボードのノード ID は、コミュニケーションバス上の認証番号を定義します。ＩＤ番号は、コミュニケーションバスでただ一つの番号でなくてはなりません。これは、ノード０から順番にすべてのノードへコミュニケーションバスの操作を行うからです。ノードＩＤはスイッチ S2 で定義します。スイッチは ON すると 1 になります。このノード番号はコミュニケーションバス上の物理的な位置とは関係していませんので物理的な位置は自由です。



ON(Up)　　= 1
OFF(Down) = 0

S2 Position 8	S2 Position 7	S2 Position 6	S2 Position 5	S2 Position 4	S2 Position 3	S2 Position 2	S2 Position 1	Node ID Hex (Dec.)
ON	ON	ON	ON	ON	ON	ON	ON	$FF(255)
ON	OFF	OFF	OFF	OFF	OFF	OFF	OFF	$80 (128)
OFF	ON	OFF	OFF	OFF	OFF	OFF	OFF	$40 (64)
OFF	OFF	ON	OFF	OFF	OFF	OFF	OFF	$20 (32)
OFF	OFF	OFF	ON	OFF	OFF	OFF	OFF	$10 (16)
OFF	OFF	OFF	OFF	ON	OFF	OFF	OFF	$8 (8)
OFF	OFF	OFF	OFF	OFF	ON	OFF	OFF	$4 (4)
OFF	OFF	OFF	OFF	OFF	OFF	ON	OFF	$2 (2)
OFF	OFF	OFF	OFF	OFF	OFF	OFF	ON	$1 (1)
OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF	$0 (0)

Factory Default: S2 positions 1 through 8 OFF



S1 S2

PMC-5565PIORC の場合

１．９　新シリーズのリダンダントモード設定(スイッチ S1)

Switch S1 Configuration RFM-5565

	OFF	ON	
Position 1	non-redundant mode	redundant mode	
Position 2	higher performance achievable	low network usage	
Position 3,4	PCI Window Size	3	4
	Default	Off	Off
	64 MByte	On	Off
	16 MByte	Off	On
	2 MByte	On	On
Position 5	disables Rogue Master 0	enables Rogue Master 0	
Position 6	disables Rogue Master 1	enables Rogue Master 1	
Position 7	Reserved		
Position 8	most recent control logic	original factory control logic	

Factory Default: S1 positions 1 through 8 OFF

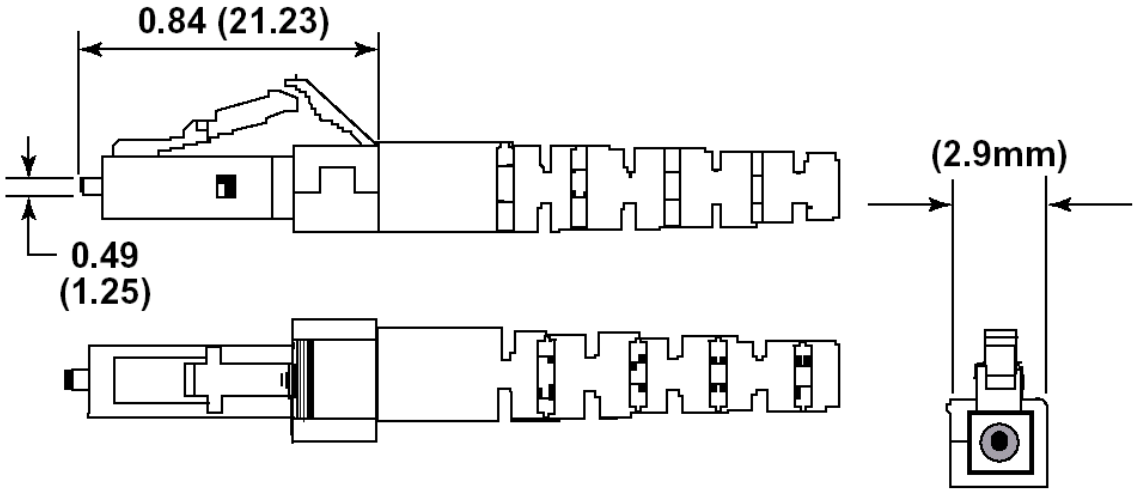
PCIE-5565RC の場合

２．リフレクティブメモリ・ボードの動作

２．１　動作概要

5565 は、256 台までのシステムを１つのコミュニケーションバスに接続することが可能で、１つのシステムが書き込んだデータを自動的にすべてのシステム書き込みます。コミュニケーションバスは、２つの LC タイプコネクタケーブル(NTT 標準 JIS C 5973)によって接続し、アドレス、データ、トークン情報をすべてのボードに供給します。またコミュニケーションバスの制御は、（順番に制御を許可する）トークンを各々のボードが得る事で可能になります。

5565 は、ユーザが任意のメモリ空間を READ/WRITE することが出来ます。ユーザがメモリに書き出すためにトークンを得る必要はありません。すべてのメモリに対する書き出しは、ボード上のＳＲＡＭに格納され、その他のノードに対する書込みは一度 FIFO に書き出され、ボードがトークンを得たときにブロードキャストされます。割込みコマンドは、メモリの所定の場所にデータワードを書き出す事で特定のシステムあるいは、すべてのシステムに発生させることが出来ます。このときのデータワードで割込みを受けるノードＩＤと種類を指定します。割込みは、データを書き込んだ順に送信出力され、システムから受信されますが、もしボードにブロックデータを書き出た後、割込みコマンドを送信すると、データをすべてのボードにブロードキャストした後に割込みコマンドを送信します。



‘LC’タイプマルチモード光ファイバーケーブルコネクタ

２．２　バス速度の設定

コミュニケーションバスの最高転送速度は 174MB/sec ですが、長距離のケーブルを必要とする場合、転送時のデータにエラーが生じる場合があります。このような場合に、リダンダントモード(87MB/sec)でコミュニケーションバス速度を定義する事が出来ます。したがってホストコンピュータからの大きなデータを転送する場合の速度に影響を与えます。しかし、コミュニケーションバスの速度低下は、ホストの応答時間に影響を与えません。ここで注意する事は、コミュニケーションバスに接続されているすべてのボードの速度設定が同様の設定になっていなければならないことです。
また、この速度は、16Lword 転送時の速度なので、DMA 転送時にしか使われません。
通常のプログラム I/O 転送では、4 バイト read/write なので、高速時、43MB/sec リダンダントモード時で 20MB/sec になります。

２．３　Rouge Packet(はぐれパケット)削除オペレーション

Rouge Packet(はぐれパケット)は、ネットワーク上でノードに属さないパケットです。
リフレクティブメモリの基本的なオペレーションでは、ホストからメモリ書き込みに応えてネットワークに１パケットを送信します。
パケットは、それが出発点のノードに戻るまで、ネットワークのすべてのノードに転送されます。
出発点のノードはパケットをネットワークから取り除くための必要条件です。
しかしながら、パケットが、別のノードを通過するとき、あるいは出発点のノードが誤作動し始めた場合に、出発点のノードはパケットをそれ自身のものとして認知し損ね、ネットワークからパケットを削除できないでしょう。
この場合パケットは永久にネットワークを通過し続けるでしょう。
Rouge Packet は極めてまれです。それらの存在は厳しい環境でコンポーネントの障害か、誤作動しているボードを示します。 通常、解決方法は誤作動しているボードを隔離して置き換え、そして環境を改善することです。
しかしながら、Rouge Packet がネットワークから取り除かれるなら、メンテナンスのためにシステムを止めるより、むしろ散発的なはぐれパケットを大目に見ることが必要な場合もあります。
Rouge Packet フォールトに耐性を提供するために、 VMIVME-5565 は２つの Rogue Master の１つとして機能することができます。 それは１つの Rogue Master ノードを通過するときに、Rogue Master が通過するパケットを変更します。 パケットが２番目の Rogue Master のところに戻るとき、Rogue Master はそれがはぐれパケットであることを認識して、そしてそれをリングから取り除きます。
Rouge Packet が検出されるとき、Rouge Packet フォールトフラグがローカル割り込みステータスレジスタ（LISR）にセットされます。
オプション設定で、Rouge Packet フォールトビットの発生により、ホストに VMEbus 割り込みを発生させることが出来ます。
Rogue Master 0 と Rogue Master 1 が、お互いをチェックするために提供されます。

２．４　割込みのハンドリング

割込みのハンドリングは、ボード上のバス割込みモジュールが行います。リモートシステムからの割込み処理にバス割込みモジュールは、INT1,INT2,INT3,INT4 を使用しています。すべての割込みは、電源投入時に禁止されるようにマスクされ、プログラムの制御下で開始する事ができます。
本デバイスドライバでは、任意の割り込みを４つのシグナルに割り当てることで処理を行います。
この割り当ては、アプリケーションプログラムで行います。

３．アプリケーション・プログラミング・インターフェース
３．１　リフレクティブメモリライブラリ

libvmic は、external memory ドライバを利用したリフレクティブメモリボードアクセスライブラリです。

本ライブラリとデバイスドライバは、コンカレント社が独自にリアルタイム用途に開発したものであり、GE Fanuc 社から供給されるデバイスドライバとは API と性能が異なることに注意してください。

割り込みハンドラの登録

```
int pci5565_setup_signal
      (
         int fd,
         PCI5565R *dev,
         PCI5565R *intr,
         void (*interrupt_hadler)(int)
      );
```

戻り値
　　エラーなら-1 成功なら 0

引数
　fd　ファイルディスクリプタ番号
　dev pci5565 のデバイスレジスタへのポインタ
　intr pci5565 の割り込みレジスタへのポインタ
　void (*interrupt_hadler)(int) 割り込みハンドラ

共有モードの場合でも、任意のユーザが実行できるが、
最後に登録したプロセスにのみ シグナルは配信される

デバイスの非初期化処理

```
int pci5565_uninit(
         int fd,
         PCI5565M *mem,
         PCI5565R *dev,
         PCI5565R *intr
      );
```

戻り値
　　エラーなら-1 成功なら 0

引数
　fd　ファイルディスクリプタ番号
　mem pci5565 のメモリへのポインタ
　dev pci5565 のデバイスレジスタへのポインタ(pci5565 のみ)
　intr pci5565 の割り込みレジスタへのポインタ(pci5565 のみ)

レジスタの初期化部分は、共有モードの場合最初のユーザのみが実行できる

デバイスの初期化処理

```
int pci5565_init
      (
        int fd,
        PCI5565M **mem,
        int *mem_size,
        PCI5565R **dev,
        PCI5565R **intr,
        int option
      );
```

戻り値

エラーなら-1 成功なら 0

引数

fd　ファイルディスクリプタ番号
mem pci5565 のメモリへのポインタが返される
mem_size pci5565 のメモリのサイズが返される
dev　pci5565 のデバイスレジスタへのポインタが返される
intr pci5565 の割り込みレジスタへのポインタが返される
option　1 を指定すると以下の情報が表示される

```
mapped 4096 bytes,at 0xdfdfddc0
and mapped 1074442991 bytes,at 0xd0000000
NodeID 0 real size 128MB rev 7
Redundant Mode Disabled(Installed E7:1-2)
Rogue Master 0 Disabled(Installed E7:3-4)
Rogue Master 1 Disabled(Installed E7:5-6)
Rx Signal Detect
Rx FIFO is Ok
Sync Signal is Ok
DATA is Ok
OWN DATA not Detect
```

レジスタの初期化部分は、共有モードの場合最初のユーザのみが実行できる

他ノードに対しての割り込みを発生させる

```
int pci5565_raise_signal
      (
        PCI5565R *dev,
        int sig_num,
        int target,
        unsigned int data
      );
```

戻り値

エラーなら-1 成功なら 0

引数

dev pci5565 のデバイスレジスタへのポインタが返される

sig_num 割り込みの種類 以下のいずれかを指定する

PCI5565_RNR0	リセット割り込み
PCI5565_NIC1	割り込み 1
PCI5565_NIC2	割り込み 2
PCI5565_NIC3	割り込み 3
PCI5565_NIC4	割り込み 4
PCI5565_NICALL	ブロードキャスト割り込み

target　割り込みをかけるターゲットノード番号

data　割り込みをかけたノードに引き渡すデータ

割り込みサービス関数 割り込んだ際のレジスタの値を戻す

```
int pci5565_intr_service
      (
        int fd,
        unsigned int *icsr,
        unsigned int *lisr
      );
```

戻り値

エラーなら-1 成功なら 0

引数

fd　ファイルディスクリプタ番号

pci5565 の場合

icsr,lsir 値を戻す変数

割り込みを禁止する

```
int pci5565_disable_intrrupt
      (
        PCI5565R *dev,
        PCI5565R *intr
      );
```

戻り値
　　エラーなら-1 成功なら 0
引数
　dev pci5565 のデバイスレジスタへのポインタ
　intr pci5565 の割り込みレジスタへのポインタ

割り込みを許可する

```
int pci5565_enable_intrrupt
      (
        PCI5565R *dev,
        PCI5565R *intr,
        int option
      );
```

戻り値
　　エラーなら-1 成功なら 0
引数
　dev pci5565 のデバイスレジスタへのポインタ
　intr pci5565 の割り込みレジスタへのポインタ
　option 通常は０を指定する。
　１を指定した場合は、他ノードからの割り込み以外の割り込みが有効になる。

割り込みを再び有効にする

```
int pci5565_rearm_interrupt
      (
        PCI5565R *dev
      );
```

戻り値
　　エラーなら-1 成功なら 0
引数
　dev pci5565 のデバイスレジスタへのポインタ

名前

rfmcpy - リフレクティブメモリ領域をコピーする。

書式

```
#include <extmem.h>
#include <libvmic.h>

void *rfmcpy(void *dest, const void *src, size_t n);
```

説明

rfmcpy()はリフレクティブメモリ、あるいはメインメモリの領域 src の先頭 n バイトをリフレクティブメモリ あるいはメインメモリのメモリ 領 域 dest に read()/write() を使ってコピーする。コピー元の領域とコピー先の領域が重なってはならない。
src と dest の両方が、リフレクティブメモリを示している場合には、4096 バイトのバッファを利用したコピーを繰り返す。
サイズが１０２４バイト以下の場合と、read()/write()/lseek()でエラーが発生した場合や、 src と dest の両方が、メインメモリの領域を示している場合には、memcpy()を呼び出す。

返り値

rfmcpy() は dest へのポインタを返す。

準拠

SVr4, 4.3BSD, C99

関連項目

read(2), write(2), lseek(2), memcpy(3),

SEE ALSO

/usr/local/CNC/drivers/extmem/vmic 下のプログラム

AUTHORS

３．２　extmem デバイスドライバインターフェース

extmem デバイスドライバは、デバイスをユーザ空間から直接デバイスをアクセスすることの出来る汎用デバイスドライバです。

書式

#include <sys/extmem.h>

機能解説

/dev/extmem[0-9]は、PCIbus 上のボードをアプリケーションプログラムから 10 デバイスまで、直接アクセスする機能を提供します。 /dev/resmem[0-5] は、 システム起動時の reserved memory をアプリケーションプログラムから 6 セグメントまで、分割アクセスする機能を提供します。

また、このデバイスドライバでは、１つのボードあたり 6 つのベースアドレスを定 義する事と、ボードで発生した割り込みをハンドリングすることができるように設計されています。デバイスドライバは、open/close/mmap 時ボードに アク セスしませんので、そのすべての動作はユーザプログラムに委ねられます。
extmem で利用するデバイスの MEMORY デバイスは、 mmap(2) を使いユーザ空間にマ ッピ ン グ し 。 直接アクセスすることができますが、 I/O デバイスは、ioctl() を使いカーネル空間で、BUS WINDOW モードのプログラムでアクセス します。しかし、例外的に、root ユーザだけは、iopl()を用いてユーザ空間に I/O デバイスを直接マッピングしてアクセスすることができます。

OPEN/CLOSE

extmem は、通常のデバイスファイルと同様に open/close 可能です。デバイスは、 実使用の前に必ずユーザーが初期化する必要があります。ディフォルトでは、非共有モードですが、下記の IOCTL_EXTMEM_SHARED を発行すると、複数のユーザ でデバイスを共有できます。但し、レジスタなどの初期化の責任はユーザに任されます。デバイスドライバでは最初に open したプロセスが最後に close することを仮定しています。

MMAP

```
#define roundup(x,y)    (((int)(x) + y -1)& ~(y-1))
typedef struct
{
           union {
                volatile unsigned char             reg08[4096];
                volatile unsigned short int     reg16[2048];
                volatile unsigned long int         reg32[1024];
           } access;
} extmem;

if ((fd=open("/dev/extmem0",O_RDWR)) < 0)
{
        fprintf(stderr,"can' t open /dev/extmem/0");
        exit(0);
}
addr = (extmem*)mmap(NULL,roundup(sizeof(extmem),
sysconf(_SC_PAGESIZE)),PROT_READ|PROT_WRITE,MAP_SHARED,fd, 0L);
if (addr==(extmem*)(-1))
{
         exit(0);
}
```

READ/WRITE

PCI5565,PMC5565 のみ動作し、ダイレクト転送を行います。リフレクティブメモリ上の位置は、lseek()関数で決定します。

read()/write()後は、位置が size 分移動していることに注意してください。

IOCTLS

extmem は次の ioctl() 呼び出しを処理します。

1)デバイス情報をＯＳから得ます。

```
ret = ioctl(fd,IOCTL_EXTMEM_GET_DRIVER_INFO,extmem_driver_info_t*);

typedef struct
{
      struct
      {
            int      type;  /* 0:未使用
                            1:I/O 領域
                            2:MEMORY 領域*/
            int      address;
            int      offset;
            int      size;
      } region[PCI_MAX_NUM_REGS];
      int      irq;         /* IRQ の値 */
} extmem_driver_info_t;
```

2)BUS WINDOW プログラムを単一に実行します。

```
ret = ioctl(fd,IOCTL_EXTMEM_SINGLE,extmem_busw_command_t *);
typedef struct
{
      /* bus window structure */
      int   window;
      long operation;            /* read registers, write registers */
      long offset;               /* offset from board base address of register */
      unsigned int *data_ptr; /* address in user space for data, */
                                  * or immediate data for MODE_W_I */
} extmem_busw_command_t;
```

3)BUS WINDOW プログラムを複数連続に実行します。

```
ret = ioctl(fd,IOCTL_EXTMEM_BUSWINDOW,extmem_buswindow_t *);
/* BUSWINDOW ioctl takes a string of commands to read/write the board */
typedef struct
{
         int count;                     /* number of commands to follow */
         extmem_busw_command_t *busw_command;/* array of busw_commands */
} extmem_buswindow_t;
```

4)割り込みの発生を SIGIO で通知させます。

```
ret = ioctl(fd,IOCTL_EXTMEM_REQ_INT_NOTIFY,extmem_int_info_t);
typedef struct
{
      unsigned long  init;//0 = 禁止, else 許可
      unsigned long  irq; //無視する
} extmem_int_info_t;
```

許可

```
int_enable.init = 1;
ioctl(fd,IOCTL_EXTMEM_REQ_INT_NOTIFY,&int_enable);
```

禁止

```
int_enable.init = 0;
ioctl(fd,IOCTL_EXTMEM_REQ_INT_NOTIFY,&int_enable);
```

5)SIGIO の発生回数を得ます。

システムに保持されている割り込み発生回数は、この関数を呼び出すたびに、１減じられその値が戻されます。
したがって、発生した回数だけ呼び出す必要があります。

```
ret = ioctl(fd,IOCTL_EXTMEM_REQ_INT_STATUS,extmem_int_info_t);
typedef struct
{
        unsigned long  init;//無視する
        unsigned long  irq; //SIGIO の発生回数
} extmem_int_info_t;
```

6)割り込みの発生を AST で通知させます。

```
ret = ioctl(fd,IOCTL_EXTMEM_DCLAST,extmem_ast_t *);
typedef struct
{
           volatile unsigned long     timeout; // 0 = 禁止, else 許可
           volatile unsigned long     pending;
} extmem_ast_t;
```

許可

```
ast_enable.timeout = 1;
ioctl(fd,IOCTL_EXTMEM_REQ_INT_NOTIFY,&int_enable);
```

禁止

```
ast_enable.timeout = 0;
ioctl(fd,IOCTL_EXTMEM_REQ_INT_NOTIFY,&int_enable);
```

7)ソフトウェア AST の発生

```
ret = ioctl(fd,IOCTL_EXTMEM_DELAST,NULL);
```

8)AST の発生をマイクロ秒のタイムアウト指定で待ちます。

```
ret = ioctl(fd,IOCTL_EXTMEM_PAUAST,extmem_ast_t *);
typedef struct
{
     volatile unsigned long  timeout;
     volatile unsigned long  pending;
} extmem_ast_t;
```

す で に 発生した割り込みがある場合には即時に復帰し発生していない場合には timeout 時間まで発生を待ちます。
ast.timeout は、timeout 時間をマイクロ秒で指示しますが、その精度はミリ 秒です。
関 数の戻り値として、 ret == 0 の場合には、割り込みの発生があり、ret == -1 の場合には、timeout になったことを示します。
パラメータ ast.timeout には、 割り込みが発生するまでの時間がマイクロ秒 で戻されますが、この時間は、マイクロ秒の精度を持っています。
ペンディングされている割り込みの発生回数は、ast.pending に戻ってきます。
システムに保持されている割り込み発生回数は、この関数を呼び出すたびに 、１減じられその値が戻されます。

したがって、発生した回数だけ呼び出す必要があります。

9）カーネル空間で実行される BUS WINDOW プログラムを登録します。

```
ret = ioctl(fd,IOCTL_EXTMEM_BUSWINDOW_INTR0,extmem_buswindow_t *);
ret = ioctl(fd,IOCTL_EXTMEM_BUSWINDOW_CLOSE,extmem_buswindow_t *);
ret = ioctl(fd,IOCTL_EXTMEM_BUSWINDOW_IINF0,extmem_buswindow_t *);
ret = ioctl(fd,IOCTL_EXTMEM_RESON_POSITION,int *);
ret = ioctl(fd,IOCTL_EXTMEM_RESON_MASK,int *);
```

この処理は、割り込み処理と close 処理があります。この場合の BUS WINDOW プログラムでは、MODE_R_I, MODE_W_I, MODE_S_I, MODE_C_I のイミディエートモードしか使えません。また、ユーザが値を戻すためには、IOCTL_EXTMEM_BUSWINDOW_IINF0 を使う必要があります。IOCTL_EXTMEM_BUSWINDOW_IINF0 と IOCTL_EXTMEM_BUSWINDOW_INTR0 は対で、同じサイズの prog 領域を使わなくてはなりません。この理由は、カーネル空間の 割り 込み処理から読み書きするため、すべての領域をドライバの中に持つ必要があるからです。 IOCTL_EXTMEM_RESON_POSITION は、 割り込みルーチンで、共有割 り込みの場合に自身の割り込みであるか判別するために使うポジションを指示します。通常最初の処理であるため、0 を指示します。この結果と IOCTL_EXTMEM_RESON_MASK の論理積が 0 であると、ユーザ割り込みハンドラは起動しません。 IOCTL_EXTMEM_RESON_MASK のディフォルト値は 0xFFFFFFFF です。また、割り込みが共有でない場合には、この処理は必要ではありません。

以下に vmipci5565 で使われた例を示します。

```
        extmem_busw_command_t   prog[3];
        extmem_buswindow_t      wcmd;

        prog[0].window=PCIBAR1;
        prog[0].operation=MODE_LONG|MODE_R_I;
        prog[0].offset = PCI5565_ICSR_OFFSET;
        prog[0].data_ptr = 0;

        prog[1].window=PCIBAR2;
        prog[1].operation=MODE_LONG|MODE_R_I;
        prog[1].offset = PCI5565_LISR_OFFSET;
        prog[1].data_ptr = 0;

        prog[2].window=PCIBAR2;
        prog[2].operation=MODE_LONG|MODE_C_I;
        prog[2].offset = PCI5565_LISR_OFFSET;
        prog[2].data_ptr = (unsigned int*)(
                                PCI5565_LICR_LOCAL_MEMORY_PARITY|
                                PCI5565_LICR_MEMRY_WRITE_INHIBIT|
                                PCI5565_LICR_LATCHED_SYNC_LOSS|
                                PCI5565_LICR_RX_FIFO_FULL|
                                PCI5565_LICR_BAD_DATA|
                                PCI5565_LICR_ROGUE_PACKET_FAULT|
                                PCI5565_LICR_INTERRUPTR|
                                0);
        wcmd.count=3;
        wcmd.busw_command=prog;
        if(ioctl(fd,IOCTL_EXTMEM_BUSWINDOW_INTR0,&wcmd)<0)
        {
                fprintf(stderr,"extmem        infomation    fail   %s0,str-
error(errno));
                return(-1);
```

```
        }
        pos = 1;
        if(ioctl(fd,IOCTL_EXTMEM_RESON_POSITION,&pos)<0)
        {
                fprintf(stderr,"extmem          fd,IOCTL_EXTMEM_RESON_POSI-
TION,&pos fail %s0,strerror(errno));
                return(-1);
        }
        mask =  PCI5565_LICR_LOCAL_MEMORY_PARITY|
                PCI5565_LICR_MEMRY_WRITE_INHIBIT|
                PCI5565_LICR_LATCHED_SYNC_LOSS|
                PCI5565_LICR_RX_FIFO_FULL|
                PCI5565_LICR_BAD_DATA|
                PCI5565_LICR_ROGUE_PACKET_FAULT|
                PCI5565_LICR_INTERRUPTR;
        if(ioctl(fd,IOCTL_EXTMEM_RESON_MASK,&mask)<0)
        {
                fprintf(stderr,"extmem  fd,IOCTL_EXTMEM_RESON_MASK fail
%s0,strerror(errno));
                return(-1);
        }
```

この処理は、割り込み時に、PCI5565_ICSR_OFFSET,PCI5565_LISR_OFFSET の値を読み込みそれぞれ、prog[0].data_ptr,prog[1].data_ptr に格納します。その後、PCI5565_ICSR_OFFSET,PCI5565_LISR_OFFSET の値を再度読み込んで、prog[2].data_ptr に指示したビットをクリアして、再び書き出して、割り込みの停止処理を行います。

prog[1]の処理は、一見無駄に見えますが、PCI5565_LISR_OFFSET のビットクリアする前の値を保存するために、必ず必要です。この後、シグナルハンドラから、アプリケーションで

```
        extmem_busw_command_t   prog[3];
        extmem_buswindow_t      wcmd;

        wcmd.count=3;
        wcmd.busw_command=prog;
        if(ioctl(fd,IOCTL_EXTMEM_BUSWINDOW_IINF0,&wcmd)<0)
        {
                fprintf(stderr,"extmem       infomation    fail    %s0,str-
error(errno));
                return(-1);
        }
        *icsr = (unsigned int)prog[0].data_ptr;
        *lisr = (unsigned int)prog[1].data_ptr;
```

を実行することにより、prog[0].data_ptr,prog[1].data_ptr に格納された値を読み込みます。このとき、prog[2].data_ptr の値は、prog[1].data_ptr のビットクリアされた値になっています。

以下の処理は、close 時に行われますので、アプリケーションプログラムがコアダンプして、意図しない、終了を行った場合に、自動処理されます。ここでは、割り込みを全て禁止することにより、その後の意図しない割り込みの発生を防ごうとしています。

```
        extmem_busw_command_t   prog[0];
        extmem_buswindow_t      wcmd;

        prog[0].window=PCIBAR2;
        prog[0].operation=MODE_LONG|MODE_W_I;
        prog[0].offset = PCI5565_LIER_OFFSET;
```

```
        prog[0].data_ptr = 0;
        wcmd.count=1;
        wcmd.busw_command=&prog[0];
        if(ioctl(fd,IOCTL_EXTMEM_BUSWINDOW_CLOSE,&wcmd)<0)
        {
                fprintf(stderr,"extmem       infomation    fail    %s0,str-
error(errno));
                return(-1);
        }
```

この処理の値を戻す事はできません。

10)デバイスドライバをシェアードモードにする

```
ret = ioctl(fd,IOCTL_EXTMEM_SHARED,NULL);
```

11)デバイスドライバをプライベートモードにする

```
ret = ioctl(fd,IOCTL_EXTMEM_PRIVATE,NULL);
```

12)デバイスドライバを共有しているユーザ数を得る

```
ret = ioctl(fd,IOCTL_EXTMEM_GET_SHARE_USERS,int *arg);
```

使用方法

extmem の ioctl()の使用は、主にＯＳのコンフィグレーションパラメータを得る事と割り込みの制御にあります。

extmem は、OS が自動的に割り当てた、6 つのベースアドレスを以下の コードで読みだしその全ての領域を利用できるようになっています。

```
if(ioctl(fd,IOCTL_EXTMEM_GET_DRIVER_INFO,&info)<0)
{
      fprintf(stderr,"Can not get parameter0);
      return(-1);
}
for(bar=0;bar<PCI_MAX_NUM_REGS;bar++)
{
      switch(info.region[bar].type)
      {
            case 0:/* 使われていない */
            break;
            case 1:/* I/O 空間に割り当てられたもの */
                  printf("PCIBAR %d I/O Region addr 0x%08x offset 0x%08x %10d bytes0,
                  bar,
                  info.region[bar].address,
                  info.region[bar].offset,
                  info.region[bar].size);
            break;
            case 2:/* Memory 空間に割り当てられたもの */
                  printf("PCIBAR %d MEM Region addr 0x%08x offset 0x%08x %10d bytes0,
                  bar,
                  info.region[bar].address,
                  info.region[bar].offset,
                  info.region[bar].size);
            break;
      }
}
```

こ の後、システムがコンフィグレーションしたデバイスのアドレスとサイズを以下のように、メモリマップします。

```
base_address = 0x1000 * PCIBAR0;
ptr = mmap(0,info.region[PCIBAR0].size,PROT_READ|PROT_WRITE,MAP_SHARED,fd,base_address);
if (ptr==MAP_FAILED)
{
      fprintf(stderr,"mmap %s0,get_syserr(errno));
```

```
        return(-1);
    }
    ptr += info.region[PCIBAR0].offset;
```

このとき、mmap()の最後のパラメータである offset を、ベースアドレスの指 定に 利用していることに注意してください。また、必ずドライバが指示したオフセットを加算してください。
extmem コンフィグレーション時に割り込みを利用するように定義した場合には、発生した割り込み信号を、アプリケーションでシグナルあるいは、AST として以下の手順のようにハンドリングしなくてはなりません。

a)シグナルの定義

```
    if (sigprocmask(0,NULL,&set)==(-1))
    {
            fprintf(stderr, "Cannot get sigprocmask - %s0,strerror(errno));
            return(-1);
    }
    sigdelset(&set,SIGIO);
    if (sigprocmask(SIG_SETMASK,&set,&oset)==(-1))
    {
            fprintf(stderr, "Cannot set sigprocmask - %s0,strerror(errno));
            return(-1);
    }

    sigemptyset(&newact.sa_mask);
    sigaddset(&newact.sa_mask,SIGIO);
    newact.sa_handler = interrupt_hadler;
    newact.sa_flags = SA_SIGINFO|SA_RESTART;
    if (sigaction(signo, &newact, 0)==(-1))
    {
            fprintf(stderr, "Cannot set sigaction - %s0,strerror(errno));
            return(-1);
    }
    if (fcntl (fd, F_SETOWN, getpid()) != 0)
    {
            fprintf (stderr,"Error %s : Process ID = %d 0,strerror(errno), getpid());
            return(-1);
    }
    /* Register signal */
    flags = fcntl (fd, F_GETFL);
    if ( flags == -1)
    {
            fprintf (stderr,"Error : %s0,strerror(errno));
            return(-1);
    }
    if ( fcntl(fd, F_SETFL, flags | FASYNC) == -1 )
    {
            fprintf (stderr,"Error : %s0,strerror(errno));
            return(-1);
    }
    int_enable.init = 1;
    ioctl(fd,IOCTL_EXTMEM_REQ_INT_NOTIFY,&int_enable);
```

シグナルは、extmem デバイスドライバの割り込みルーチンか ら 、linux 標 準の kill_fasyc()を使って非同期イベントを通知します。
しかしこの kill_fasync()は、イベントをキューイングすることができません。したがって、signal の発生回数と、割り込みの発生回数は必ずしも等しくあ りません。
extmem デバイスドライバでは、発生した割り込みの回数をカウントしていますので、 ioctl(fd,IOCTL_EXTMEM_REQ_INT_STATUS,extmem_int_info_t) を 使 って SIGIO の発生回数を得ることができますが、複数回発生した signal は、遅れて

配信されることがあり、結果としてイベント回数０の結果を得ることもあり ますので注意してください。

b)BUS WINDOW プログラム

BUS WINDOW モードは、一般ユーザがアプリケーションプログラムからデバイスの I/O 空間にアクセスする唯一方法です。 BUS WINDOW プロ グラ ムは、以下に示すように prog.window に、PCI のベースアドレスを指示し、 prog.offset に、ベースアドレスからのオフセットを指示します。 このとき、アクセスするビット幅や方向を、 以下のテーブルにしたがって prog.operation で指示します

MODE_R　prog.data_ptr に指示した変数に値を読み込む
prog.data_ptr=&read_data;で
値は read_data に格納される。
MODE_R_I　　prog.data_ptr に値を読み込む
値は、prog.data_ptr に格納される。
MODE_W　prog.data_ptr に指示した変数の値を書き出す
prog.data_ptr=&write_data;で
書き出される値は write_data である。
MODE_W_I　　prog.data_ptr の値を書き出す
MODE_S_I　　読みだした値に指示した値をセットし、再び書き出す
prog.data_ptr=0x80000000;で値は
temp_value <-入力ポート
temp_value |= (prog.data_ptr);
出力ポート<-temp_value;
になる。
MODE_C_I　　読みだした値に指示した値をクリアし、再び書き出す
prog.data_ptr=0x80000000;で値は
temp_value <-入力ポート
temp_value &= ~(prog.data_ptr);
出力ポート<-temp_value;
になる。

この prog.operation で指示した値に、データの幅を論理和でデータ幅を指示します。

MODE_BYTE　　8bits データして扱う。
MODE_BYTE2　16bits データを２回の 8bits 操作として扱う。
MODE_WORD　　16bits データして扱う。
MODE_WORD2　32bits データを２回の 16bits 操作として扱う。
MODE_LONG　　32bits データして扱う。

以下にサンプルを示します。

```
static extmem_busw_command_t      prog[16];
static extmem_buswindow_t         wcmd;
int pc=0,ret;

/* base address 0
prog[pc].window=0;                /* base address 0 */
prog[pc].operation=MODE_LONG |MODE_R;   /* 32bit read */
prog[pc].offset=0x0;                    /* offset 0 */
prog[pc].data_ptr=&read_data;
```

```
    pc++;

    :
    :
    :

    prog[pc].window=1;                /* base address 1 */
    prog[pc].operation=MODE_WORD |MODE_W;   /* 16 bits write */
    prog[pc].offset=0x10;          /* offset 0x10 */
    prog[pc].data_ptr=&write_data;
    pc++;
    wcmd.count = pc;
    wcmd.busw_command = prog;
    ret = ioctl(fd,IOCTL_EXTMEM_BUSWINDOW, &wcmd );
```

c) resmem プログラム

予約メモリは、システムブート時にメインメモリの一部を予約して linux に使用されないようにする手法です。予約された領域は、linux の管理から除外されますので、ユーザが自由に利用できます。また、連続的なメモリ領域として存在しますので、1回の DMA で全てのデータをコピーする事ができ高速です。予約の方法は通常 /etc/grub.conf に mem=最終アドレスの様に指示します。

実際のメモリは/proc/iomem を見る事で確認できます。

```
# cat /proc/iomem
00000000-0009ffff : System RAM
000a0000-000bffff : Video RAM area
000c0000-000cefff : Video ROM
000cf000-000d8fff : Adapter ROM
000da800-000dbfff : Adapter ROM
000f0000-000fffff : System ROM
00100000-3fe8abff : System RAM <-------- システムメモリの最後
00100000-005596b1 : Kernel code
005596b2-006d9c7f : Kernel data
3fe8ac30-3fe8cbff : ACPI Non-volatile Storage <- 使用されている
3fe8cc00-3fe8ebff : ACPI Tables
3fe8ec00-3fffffff : reserved
        :
        :
        :
        :
```

ここで注意しなければならないのは、ACPI がメモリの最後を既に利用しています。この領域に重ならないようにするために、以下の様に/etc/grub.conf を設定します。

```
    timeout=10              splashimage=(hd0,0)/grub/ccur.xpm.gz
    title RedHawk Linux 2.2 (Trace=Yes, Debug=No)
          root (hd0,0)
          kernel       /vmlinuz-2.6.6-RedHawk-2.2-trace          ro
root=/dev/sda2 mem=0x3e000000 quiet
```

この例では、0x3e000000-0x3effffff を予約メモリとして確保して、0x3f000000-0x3fffffff は、ACPI がそのまま利用できる設定にしてい

ます。結果、

```
#       cat /proc/iomem
00000000-0009ffff : System RAM
000a0000-000bffff : Video RAM area
000c0000-000cefff : Video ROM
000cf000-000d8fff : Adapter ROM
000da800-000dbfff : Adapter ROM
000f0000-000fffff : System ROM
00100000-3dffffff : System RAM <-------- システムメモリの最後
00100000-005596b1 : Kernel code
005596b2-006d9c7f : Kernel data
```

のようになります。このときには、

```
3fe8ac30-3fe8cbff : ACPI Non-volatile Storage
3fe8cc00-3fe8ebff : ACPI Tables
3fe8ec00-3fffffff : reserved
```

の表示はされません。

この設定を、extmem に組み込むには

```
/usr/local/CNC/drivers/extmem/extmem_start
EXTMEM_RESADR0=0x3E000000 EXTMEM_RESLEN0=0x01000000
```

を行います。 結果、

```
# cat /proc/extmem
version: 2.00
built: Mar 10 2005, 18:13:04
boards: 0
resadr0: 0x3e000000
reslen0: 0x01000000
resadr1: 0x00000000
reslen1: 0x00000000
resadr2: 0x00000000
reslen2: 0x00000000
resadr3: 0x00000000
reslen3: 0x00000000
resadr4: 0x00000000
reslen4: 0x00000000
resadr5: 0x00000000
reslen5: 0x00000000
```

のように確認することができます。

アプリケーションから利用する例を以下に示します。

```
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/extmem.h>

main(int argc,char **argv)
{
         extern int optind;
         extern char *optarg;
         register int c;
      int fd,dev=0;
      volatile unsigned char *resbuff;
      extmem_driver_info_t resinfo;
      char resdev[256];
      extern int errno;

         while((c = getopt(argc, argv, "d:")) != EOF)
         {
                  switch(c)
                  {
                           case ' d' :
                                    dev = atoi(optarg);
                           break;
            }
      }
      sprintf(resdev,"/dev/resmem%d",dev);
      if ((fd = open(resdev,O_RDWR))<0)
      {
            fprintf(stderr,"open fail %s0,strerror(errno));
            exit(0);
      }
      if(ioctl(fd,IOCTL_EXTMEM_GET_DRIVER_INFO,&resinfo)<0)
      {
                  fprintf(stderr,"extmem infomation fail %s0,strerror(errno));
            exit(0);
      }

      printf("%s phsical address %x size %x¥n"
            ,resdev, resinfo.region[0].address,resinfo.region[0].size);
      resbuff = (char*)mmap(0,resinfo.region[0].size,PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);

      printf("%x¥n" ,resbuff[0]++);

      munmap((void*)resbuff,resinfo.region[0].size);
      close(fd);
}
```

実行すると最初のデータが常にインクリメントされ表示されます。

```
# ./resmem_ex
/dev/resmem0 phsical address 3e000000 size 1000000
0
# ./resmem_ex
/dev/resmem0 phsical address 3e000000 size 1000000
1
# ./resmem_ex
```

/dev/resmem0 phsical address 3e000000 size 1000000
2

ここで、表示されているアドレスは、PCI バスから認識される物理アドレスですので、DMA コントローラなどに設定すれば、デバイスからメモリに DMA が行われ、マップして領域で、アプリケーションプログラムからアクセスできます。

FILES

/usr/local/CNC/drivers/extmem_driver/extmem.ko
/usr/local/CNC/drivers/extmem_driver/sys/extmem.h
/usr/local/CNC/drivers/extmem_driver/extmem_start

４． extmem コンフィグレーション

リフレクティブメモリのコンフィグレーションは、以下の手順で行います。
以下に、この意味を説明します。

1)定義

insmod 時に、以下のパラメータ定義を、PCI ボードに合わせて行ないます。なお、各パラメータの n は、0-9 を指定します。

EXTMEM_VIDn=ベンダーコード
EXTMEM_DIDn=デバイスコード
EXTMEM_INTn=割り込み
0:割り込みは使用しない
1:IRQ を open 時に割り当てる
2:IRQ を init 時に割り当てる

例：insmod extmem.o EXTMEM_VID0=0x114A EXTMEM_DID0=0x5565 EXTMEM_INT0=2

2) device ファイルの編集

利用するデバイスの数だけ以下の記述を用意して下さい。この例では、2 つのデバイスを /dev/extmem? の名称でコンフィグレーションしていますが、 わかりやすい名称でかまいません major の値は、OS が決定するので、/proc/devices を読みだして extmem の項の値を使ってください。

mknod -mrw /dev/extmem0 c major 0
mknod -mrw /dev/extmem1 c major 1
mknod -mrw /dev/extmem2 c major 2

なお、自動的にこれを行うシェルスクリプトを/usr/local/CNC/drivers/extmem_driver/extmem_start に用意しています。

4.1 extmem を使用した 5565 の組み込み手順

下記手順で展開してください

```
# make
# make install
```

組込み手順

(1) /etc/grub.conf の最後に以下のように pci=routeirq を(必要ならば、vmalloc=256M も)加えてください

```
        title RedHawk Linux 6.0.1-20110817 (Trace=Yes, Debug=No)
    root (hd0,0)
    kernel /vmlinuz-2.6.36.4-RedHawk-6.0.1-trace ro root=/dev/mapper/vg_ihawk-lv_root
rd_LVM_LV=vg_ihawk/lv_root rd_LVM_LV=vg_ihawk/lv_swap \
        rd_NO_LUKS rd_NO_MD rd_NO_DM LANG=ja_JP.UTF-8 KEYBOARDTYPE=pc
KEYTABLE=us quiet crashkernel=90M@48M \
        pci=routeirq
    ~~~~~~~~~~~~
```

(2)/etc/rc.local に以下のシェルを加えてください

```
        /usr/local/CNC/drivers/extmem/extmem_start EXTMEM_VID0=0x114A
EXTMEM_DID0=0x5565 EXTMEM_INT0=2
        if [ -L /dev/pci5565a ]
        then
                rm -f /dev/pci5565a
        fi
        ln -s /dev/extmem0 /dev/pci5565a

        if [ -L /dev/pci5565b ]
        then
                rm -f /dev/pci5565b
        fi
        ln -s /dev/extmem1 /dev/pci5565b
```

サンプルが以下のシェルスクリプトに記述されています。

```
    # /usr/local/CNC/drivers/extmem/ge/pci5565/config_*.sh
```

(3) resmem との併用

予約メモリは、 システムブート時にメインメモリの一部を予約して linux に使用されないようにする手法です。予約 された領域は、linux の管理から除外されますので、ユーザが自由に利用できます。また、連続的なメモリ領域として存在しますので、 1 回の DMA で全てのデータをコピーする事ができ高速です。

予約の方法は

```
        memexact -x -MS=サイズ
```

を実行して出てきた出力を/etc/grup.conf に追記します。

memexact -x -MS=128M を実行した場合の例

```
  memmap=exactmap memmap=0x8d800@0x10000 \
        memmap=0x1ff00000@0x100000 memmap=0x1fe00000@0x20200000 \
        memmap=0x8297c000@0x40200000 memmap=0x8000000$0xc2b7c000 \
                                          ~~~~~~~~~~~~~~~~~~~~
        memmap=0x60000#0xcaf9f000 memmap=0x1000@0xcafff000
```

このとき 、128MB のメモリを予約することを要求していますが、この記述中の "memmap=0x8000000$0xc2b7c000"の部分が、このシステムのメモリ配置に合わせた設定です。

この値を、/etc.rc.local で /usr/local/CNC/drivers/extmem/extmem_start EXTMEM_RESADR0=0xc2b7c000 EXTMEM_RESLEN0=0x8000000 のように記述することで、予約メモリを設定できます。

実際のメモリは/proc/iomem を見る事で確認できます。

(4)削除方法
$ su
# rmmod extmem

(5)アンインストール方法
$ su
# cd /usr/local/CNC/drivers/extmem
# make clean
# cd /usr/local/CNC/drivers/extmem/ge/pci5576
# make clean

(6)試験手順
正しく組み込まれると以下の試験プログラムが正常に実行できます
$ su
# cd /usr/local/CNC/drivers/extmem/ge/pci5565
# ./pci*_init
# ./pci*_mmap

(7)ファイル

README	This file.
Makefile	
config_pci5565.sh	Normal configration sample
config_pci5565.tune.sh	Tune up configration sample
pci5565.3.UTF8	Manual source
pci5565.3.gz	Manual
pci5565_init.c	initial program
pci5565_mmap.c	utility for pci5565
sys/pci5565.h	

64 bit system

libpci5565.so	64 bit access library
libpci5565.a	64 bit access library
libpci5565_32.a	32 bit compatibility access library
libpci5565_32.so	32 bit compatibility access library

32 bit system

libpci5565.so	32 bit access library
libpci5565.a	32 bit access library

pci5565_resmem	sample program for IOCTL DMA
test1	sample program for read/write DMA
test2	a DMA diagnostic tool

4.2 /etc/rc.local への記述例

通常は、/etc/rc.local に以下の記述を追加します

```
/usr/local/CNC/drivers/extmem/extmem_start EXTMEM_VID0=0x114A EXTMEM_DID0=0x5565
EXTMEM_INT0=2
if [ -L /dev/pci5565a ]
then
        rm -f /dev/pci5565a
fi
ln -s /dev/extmem0 /dev/pci5565a

if [ -L /dev/pci5565b ]
then
```

```
        rm -f /dev/pci5565b
fi
ln -s /dev/extmem1 /dev/pci5565b
```

もし、チューニングオプションを使用する場合には、以下のような記述を追加します。

```
/usr/local/CNC/drivers/extmem/extmem_start \
EXTMEM_VID0=0x114A EXTMEM_DID0=0x5565 EXTMEM_INT0=2 EXTMEM_ReadThrottleRate=-2
EXTMEM_WriteThrottleRate=-2 EXTMEM_EmptyThrottleRate=-1 \
EXTMEM_RESADR0=0x31c000000 EXTMEM_RESLEN0=0x10000000

if [ -L /dev/pci5565a ]
then
        rm -f /dev/pci5565a
fi
ln -s /dev/extmem0 /dev/pci5565a

if [ -L /dev/pci5565b ]
then
        rm -f /dev/pci5565b
fi
ln -s /dev/extmem1 /dev/pci5565b
```

4.3 チューニングパラメータ

以下のパラメータが調整可能になっています。

EXTMEM_ReadThrottleRate=-1 (READ DMA 終了の監視の待ち時間：μ秒)
EXTMEM_WriteThrottleRate=-1(WRITE DMA 終了の監視の待ち時間：μ秒)
EXTMEM_EmptyThrottleRate=-1(送信 FIFO EMPTY の監視の待ち時間：μ秒)

それぞれ、
正の値の場合には、以下の式の通り動的に DMA の終了時間を計算して待つ

```
        w = (length * 8) / 1000 + ThrottleRate;
```

負の値の場合には、設定した値の絶対値時間で待つようになります。
簡単に言えば、ビジーウェイトの待ち時間の調整値です。
上記値は、jiffies の監視を、1μ秒のループで行う事になります。

５．プログラムの例

```
/*************************************************************************/
/*  init.c --  VMIPCI-5565 */
/*************************************************************************/
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <memory.h>
#include <sys/mman.h>
#include <sys/param.h>
#include <sys/file.h>
#include <sys/types.h>
#include <sys/extmem.h>
#include <signal.h>
#include <stdlib.h>
#include <sys/libvmic.h>

int main(int argc,char **argv)
{
       int    fd, mem_size;
       char devname[1024];
       PCI5565M  *mem;
       PCI5565R  *dev;
       PCI5565R  *intr;

       if ( argc < 2 ) {
              strcpy(devname,"/dev/pci5565a");
       } else {
              strcpy(devname,argv[1]);
       }
       if ((fd = open(devname,O_RDWR)) == -1)
       {
              fprintf(stderr,"Device not found %s(%s)¥n",
              devname,strerror(errno));
                     exit(0);
       }
       printf("%s ",devname);
       if (pci5565_init(fd,&mem,&mem_size,&dev,&intr,1)<0)
       {
              fprintf(stderr,"Device initialize error %s(%s)¥n",
              devname,strerror(errno));
                     exit(0);
       }
       if (pci5565_uninit(fd,mem,dev,intr)<0)
       {
               fprintf(stderr,"%s¥n",strerror(errno));
       }
       close(fd);
}
```

実行結果

```
# pci5565_init
               mapped 4096 bytes,at 0xdfdfddc0
               and mapped 1074442991 bytes,at 0xd0000000
               NodeID 0 real size 128MB rev 7
               Redundant Mode Disabled(Installed E7:1-2)
               Rogue Master 0 Disabled(Installed E7:3-4)
               Rogue Master 1 Disabled(Installed E7:5-6)
               Rx Signal Detect
               Rx FIFO is Ok
               Sync Signal is Ok
               DATA is Ok
               OWN DATA not Detect
```

付録　RFM(PCI556)の転送速度について

１．目的

ＲＦＭ(PCI5565)の転送速度試験を 32bit 版と 64bit 版の RedHawk で行い、両者の性能差を明らかにする。

２．試験に用いたシステム

同一の DELL T5500(4CPU)を用い、試験を行った。下記に OS 及び CPU の情報を以下に示し、異なる部分を赤字で示す。

- **32bit 版**

```
Linux LC-D 2.6.26.8-RedHawk-5.2.3-trace #1 SMP PREEMPT Fri May 1 16:24:26 EDT 2009 i686 i686 i386 GNU/Linux
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model                   : 26
model name      : Intel(R) Xeon(R) CPU          X5560  @ 2.80GHz
stepping        : 5
cpu MHz         : 2793.020
cache size      : 8192 KB
physical id     : 0
siblings        : 4
core id         : 0
cpu cores       : 4
apicid          : 0
initial apicid  : 0
fdiv_bug        : no
hlt_bug         : no
f00f_bug        : no
coma_bug        : no
fpu             : yes
fpu_exception   : yes
cpuid level     : 11
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe nx rdtscp lm
constant_tsc arch_perfmon pebs bts pni monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr dca sse4_1 sse4_2 popcnt lahf_lm ida
bogomips        : 5590.23
clflush size    : 64
power management:
```

- **64bit 版**

```
Linux ihawk64 2.6.23.17-RedHawk-5.1.1-trace #1 SMP PREEMPT Tue Jun 10 20:38:29 EDT 2008 x86_64 x86_64 x86_64 GNU/Linux
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model                   : 26
model name      : Intel(R) Xeon(R) CPU          X5560  @ 2.80GHz
stepping        : 5
cpu MHz         : 2793.000
cache size      : 8192 KB
physical id     : 0
siblings        : 4
core id         : 0
cpu cores       : 4
fpu             : yes
fpu_exception   : yes
cpuid level     : 11
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx rdtscp lm
constant_tsc arch_perfmon pebs bts rep_good pni monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr dca popcnt lahf_lm ida
bogomips        : 5589.99
clflush size    : 64
cache_alignment : 64
address sizes   : 40 bits physical, 48 bits virtual
power management:
```

３．コンパイルオプション、サイズなどの違いについて

試験に用いた 64bit 版 RedHawk(リアルタイム性は無関係のため RedHat と同義)上の gcc-4.1.2 では、-m32 と-m64 で異なるメモリモデルをコンパイルすることが出来る。その異なるメモリモデルで memcpy()関数を利用したプログラムのオブジェクトと、32bit 版 RedHat 上でコンパイルしたオブジェクトとの違いを調査した。

3.1 データサイズの違い

下記に示すように 32bit 版および 32bit エミュレーションと 64bit 版では、データサイズが異なる。

データサイズを調べるプログラム	32bit 版及び 32bit エミュレーションの場合	64bit 版の場合

main()
{
int x;
 	
printf("char %d\n",sizeof(char));
printf("short %d\n",sizeof(short));
printf("long %d\n",sizeof(long));
printf("void* %d\n",sizeof(void*));
printf("int %d\n",sizeof(int));
printf("unsigned int %d\n",sizeof(unsigned int));
printf("unsigned long int %d\n",sizeof(unsigned long int));
printf("long int %d\n",sizeof(long int));
printf("long long %d\n",sizeof(long long));
printf("long long int %d\n",sizeof(long long int));
printf("x %X %d\n",&x,sizeof(&x));
}
	
	char 1
	short 2
	long 4
	void* 4
	int 4
	unsigned int 4
	unsigned long int 4
	long int 4
	long long 8
	long long int 8
	x FFFFD780 4
		
		char 1
		short 2
		long 8
		void* 8
		int 4
		unsigned int 4
		unsigned long int 8
		long int 8
		long long 8
		long long int 8
		x FFFFE5FC 8

3.2 memcpy 試験プログラムのリスト

```
#include <strings.h>
int main()
{
        char src[0x10000];
        char dst[0x10000];
        memcpy(dst,src,0x10000);
}
```

3.3 コンパイルオプションによるオブジェクトの違い

コンパイルを、cc -O3 -S tt.c で行うと、(低速な)内部 memcpy()関数を呼び出すが、-minline-all-stringops を付けると(高速な)インライン関数に展開される。異なるサイズでも、比較を行ったが、サイズ以外の違いはなかった。以下に 32bit 版の結果を示す。

32bit 版の結果

cc -O3 -S tt.c

```
        .file   "tt.c"
        .text
        .p2align 4,,15
.globl main
        .type   main, @function
main:
        leal    4(%esp), %ecx
        andl    $-16, %esp
        pushl   -4(%ecx)
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %ecx
        subl    $131092, %esp
        leal    -131076(%ebp), %edx
        leal    -65540(%ebp), %eax
        movl    $65536, 8(%esp)
        movl    %eax, 4(%esp)
        movl    %edx, (%esp)
        call    memcpy
        addl    $131092, %esp
        popl    %ecx
        popl    %ebp
        leal    -4(%ecx), %esp
        ret
        .size   main, .-main
        .ident  "GCC: (GNU) 4.1.2 20071124 (Red Hat 4.1.2-42)"
        .section        .note.GNU-stack,"",@progbits
```

cc -O3 -S -minline-all-stringops -msse2 tt.c

```
        .file   "tt.c"
        .text
        .p2align 4,,15
.globl main
        .type   main, @function
main:
        leal    4(%esp), %ecx
        andl    $-16, %esp
        pushl   -4(%ecx)
        cld
        pushl   %ebp
        movl    %esp, %ebp
        subl    $131084, %esp
        movl    %ecx, -12(%ebp)
        movl    $16384, %ecx
        movl    %esi, -8(%ebp)
        leal    -65548(%ebp), %esi
        movl    %edi, -4(%ebp)
        leal    -131084(%ebp), %edi
        rep
        movsl
        movl    -12(%ebp), %ecx
        movl    -8(%ebp), %esi
        movl    -4(%ebp), %edi
        movl    %ebp, %esp
        popl    %ebp
        leal    -4(%ecx), %esp
        ret
        .size   main, .-main
        .ident  "GCC: (GNU) 4.1.2 20071124 (Red Hat 4.1.2-42)"
        .section        .note.GNU-stack,"",@progbits
```

同様に 64bitOS の結果を示す。オプションを付けるとインラインに命令が展開されている。

64bit 版の結果

cc -O3 -S tt.c

```
        .file   "tt.c"
        .text
        .p2align 4,,15
.globl main
        .type   main, @function
main:
.LFB2:
        subq    $131080, %rsp
.LCFI0:
        movl    $65536, %edx
        leaq    65536(%rsp), %rsi
        movq    %rsp, %rdi
        call    memcpy
        addq    $131080, %rsp
        ret
.LFE2:
        .size   main, .-main
        .section        .eh_frame,"a",@progbits
.Lframe1:
        .long   .LECIE1-.LSCIE1
.LSCIE1:
        .long   0x0
        .byte   0x1
        .string "zR"
        .uleb128 0x1
        .sleb128 -8
        .byte   0x10
        .uleb128 0x1
        .byte   0x3
        .byte   0xc
        .uleb128 0x7
        .uleb128 0x8
        .byte   0x90
        .uleb128 0x1
        .align 8
.LECIE1:
.LSFDE1:
        .long   .LEFDE1-.LASFDE1
.LASFDE1:
        .long   .LASFDE1-.Lframe1
        .long   .LFB2
        .long   .LFE2-.LFB2
        .uleb128 0x0
        .byte   0x4
        .long   .LCFI0-.LFB2
        .byte   0xe
        .uleb128 0x20010
        .align 8
.LEFDE1:
        .ident  "GCC: (GNU) 4.1.2 20070626 (Red Hat 4.1.2-14)"
        .section        .note.GNU-stack,"",@progbits
```

cc -O3 -S -minline-all-stringops -msse2 tt.c

```
        .file   "tt.c"
        .text
        .p2align 4,,15
.globl main
        .type   main, @function
main:
.LFB2:
        subq    $130960, %rsp
.LCFI0:
        movl    $8192, %ecx
        leaq    -120(%rsp), %rdi
        leaq    65416(%rsp), %rsi
        cld
        rep
        movsq
        addq    $130960, %rsp
        ret
.LFE2:
        .size   main, .-main
        .section        .eh_frame,"a",@progbits
.Lframe1:
        .long   .LECIE1-.LSCIE1
.LSCIE1:
        .long   0x0
        .byte   0x1
        .string "zR"
        .uleb128 0x1
        .sleb128 -8
        .byte   0x10
        .uleb128 0x1
        .byte   0x3
        .byte   0xc
        .uleb128 0x7
        .uleb128 0x8
        .byte   0x90
        .uleb128 0x1
        .align 8
.LECIE1:
.LSFDE1:
        .long   .LEFDE1-.LASFDE1
.LASFDE1:
        .long   .LASFDE1-.Lframe1
        .long   .LFB2
        .long   .LFE2-.LFB2
        .uleb128 0x0
        .byte   0x4
        .long   .LCFI0-.LFB2
        .byte   0xe
        .uleb128 0x1ff98
        .align 8
.LEFDE1:
        .ident  "GCC: (GNU) 4.1.2 20070626 (Red Hat 4.1.2-14)"
        .section        .note.GNU-stack,"",@progbits
```

3.4 64bit メモリモデルと 32bit エミュレーションメモリモデルの違いについて

下記に示すように、両者のオブジェクトは全く異なる。なお、64bit メモリモデルのアセンブラコードを 32bitOS 上では、再コンパイルすることは出来ない。

64bit メモリモデル

```
# cc -O3 -S -minline-all-stringops -msse2 tt.c
        .file   "tt.c"
        .text
        .p2align 4,,15
.globl main
        .type   main, @function
main:
.LFB2:
        subq    $130960, %rsp
.LCFI0:
        movl    $8192, %ecx
        leaq    -120(%rsp), %rdi
        leaq    65416(%rsp), %rsi
        cld
        rep
        movsq
        addq    $130960, %rsp
        ret
.LFE2:
        .size   main, .-main
        .section        .eh_frame,"a",@progbits
.Lframe1:
        .long   .LECIE1-.LSCIE1
.LSCIE1:
        .long   0x0
        .byte   0x1
        .string "zR"
        .uleb128 0x1
        .sleb128 -8
        .byte   0x10
        .uleb128 0x1
        .byte   0x3
        .byte   0xc
        .uleb128 0x7
        .uleb128 0x8
        .byte   0x90
        .uleb128 0x1
        .align 8
.LECIE1:
.LSFDE1:
        .long   .LEFDE1-.LASFDE1
.LASFDE1:
        .long   .LASFDE1-.Lframe1
        .long   .LFB2
        .long   .LFE2-.LFB2
        .uleb128 0x0
        .byte   0x4
        .long   .LCFI0-.LFB2
        .byte   0xe
        .uleb128 0x1ff98
        .align 8
.LEFDE1:
        .ident  "GCC: (GNU) 4.1.2 20070626 (Red Hat 4.1.2-14)"
        .section        .note.GNU-stack,"",@progbits
```

32bit エミュレーションメモリモデル

```
# cc -O3 -S -minline-all-stringops -msse2 tt.c -m32
        .file   "tt.c"
        .text
        .p2align 4,,15
.globl main
        .type   main, @function
main:
        leal    4(%esp), %ecx
        andl    $-16, %esp
        pushl   -4(%ecx)
        cld
        pushl   %ebp
        movl    %esp, %ebp
        subl    $131084, %esp
        movl    %ecx, -12(%ebp)
        movl    $16384, %ecx
        movl    %esi, -8(%ebp)
        leal    -65548(%ebp), %esi
        movl    %edi, -4(%ebp)
        leal    -131084(%ebp), %edi
        rep
        movsl
        movl    -12(%ebp), %ecx
        movl    -8(%ebp), %esi
        movl    -4(%ebp), %edi
        movl    %ebp, %esp
        popl    %ebp
        leal    -4(%ecx), %esp
        ret
        .size   main, .-main
        .ident  "GCC: (GNU) 4.1.2 20070626 (Red Hat 4.1.2-14)"
        .section        .note.GNU-stack,"",@progbits
```

3.5 32bit ネイティブメモリモデルと 32bit エミュレーションメモリモデルについて

ネイティブ 32bitOS 上でコンパイルしたオブジェクトと 64bitOS 上で、32bit エミュレーションモデルでコンパイルしたオブジェクトは、下記に示すように同一であった。

32bit 版の結果	64bit 版上の 32bit メモリモデルの結果
# cc -O3 -S tt.c	# cc -O3 -S tt.c -m32

.file "tt.c"
.text
.p2align 4,,15
.globl main
.type main, @function
main:
leal 4(%esp), %ecx
andl $-16, %esp
pushl -4(%ecx)
pushl %ebp
movl %esp, %ebp
pushl %ecx
subl $131092, %esp
leal -131076(%ebp), %edx
leal -65540(%ebp), %eax
movl $65536, 8(%esp)
movl %eax, 4(%esp)
movl %edx, (%esp)
call memcpy
addl $131092, %esp
popl %ecx
popl %ebp
leal -4(%ecx), %esp
ret
.size main, .-main
.ident "GCC: (GNU) 4.1.2 20071124 (Red Hat 4.1.2-42)"
.section .note.GNU-stack,"",@progbits
	.file "tt.c"
	.text
	.p2align 4,,15
	.globl main
	.type main, @function
	main:
	leal 4(%esp), %ecx
	andl $-16, %esp
	pushl -4(%ecx)
	pushl %ebp
	movl %esp, %ebp
	pushl %ecx
	subl $131092, %esp
	leal -131076(%ebp), %edx
	leal -65540(%ebp), %eax
	movl $65536, 8(%esp)
	movl %eax, 4(%esp)
	movl %edx, (%esp)
	call memcpy
	addl $131092, %esp
	popl %ecx
	popl %ebp
	leal -4(%ecx), %esp
	ret
	.size main, .-main
	.ident "GCC: (GNU) 4.1.2 20070626 (Red Hat 4.1.2-14)"
	.section .note.GNU-stack,"",@progbits

32bit 版の結果	64bit 版上の 32bit メモリモデルの結果
# cc -O3 -S -minline-all-stringops -msse2 tt.c	# cc -O3 -S -minline-all-stringops -msse2 tt.c -m32

.file "tt.c"
.text
.p2align 4,,15
.globl main
.type main, @function
main:
leal 4(%esp), %ecx
andl $-16, %esp
pushl -4(%ecx)
cld
pushl %ebp
movl %esp, %ebp
subl $131084, %esp
movl %ecx, -12(%ebp)
movl $16384, %ecx
movl %esi, -8(%ebp)
leal -65548(%ebp), %esi
movl %edi, -4(%ebp)
leal -131084(%ebp), %edi
rep
movsl
movl -12(%ebp), %ecx
movl -8(%ebp), %esi
movl -4(%ebp), %edi
movl %ebp, %esp
popl %ebp
leal -4(%ecx), %esp
ret
.size main, .-main
.ident "GCC: (GNU) 4.1.2 20071124 (Red Hat 4.1.2-42)"
.section .note.GNU-stack,"",@progbits
	.file "tt.c"
	.text
	.p2align 4,,15
	.globl main
	.type main, @function
	main:
	leal 4(%esp), %ecx
	andl $-16, %esp
	pushl -4(%ecx)
	cld
	pushl %ebp
	movl %esp, %ebp
	subl $131084, %esp
	movl %ecx, -12(%ebp)
	movl $16384, %ecx
	movl %esi, -8(%ebp)
	leal -65548(%ebp), %esi
	movl %edi, -4(%ebp)
	leal -131084(%ebp), %edi
	rep
	movsl
	movl -12(%ebp), %ecx
	movl -8(%ebp), %esi
	movl -4(%ebp), %edi
	movl %ebp, %esp
	popl %ebp
	leal -4(%ecx), %esp
	ret
	.size main, .-main
	.ident "GCC: (GNU) 4.1.2 20070626 (Red Hat 4.1.2-14)"
	.section .note.GNU-stack,"",@progbits

4 ＲＦＭを使用した転送速度試験について

前節で、調査した結果を踏まえて、RFM(VMIPCI-5565 32bit 版)上でのメモリ間転送試験を行った。

4.1 試験プログラム

試験プログラムは、以下の項目を試験する部分に分かれている。

(A) 32bit プログラムループコピー
(B) 64bit プログラムループコピー
(C) memcpy()コピー
(D) rfmcpy() コピー(注)

また、A,B については、下記①②の試験を行い、C,D については、①②③の試験を行った。

①メインメモリから rfm への転送
②rfm からメインメモリへの転送
③rfm から rfm への転送

（注 1）新規作成関数について

pci5565 の read()/write()を利用できるように、デバイスドライバを改修し、以下、２つの関数を新たに作成しました。

名前

rfmcpy - リフレクティブメモリ領域をコピーする。

書式

#include <extmem.h>
#include <libvmic.h>
void *rfmcpy(void *dest, const void *src, size_t n);

説明

rfmcpy() はリフレクティブメモリ、あるいはメインメモリの領域 src の先頭 n バイトをリフレクティブメモリ あるいはメインメモリのメモリ領域 dest に read()/write()を使ってコピーする。コピー元の領域とコピー先の領域が重なってはならない。
src と dest の両方が、リフレクティブメモリを示している場合には、4096 バイトのバッファを利用したコピーを繰り返す。
サイズが 32 バイト以下の場合と、read()/write()/lseek()でエラーが発生した場合や、 src と dest の両方が、メインメモリの領域を示している場合には、memcpy()を呼び出す。

(注 2)

転送レート向上のために、チューニングパラメータが必要な場合には、extmem 起動パラメータに、下記のように EXTMEM_ReadThrottleRate=値 EXTMEM_WriteThrottleRate=値 EXTMEM_EmptyThrottleRate=値を加えてください。

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <memory.h>
#include <sys/mman.h>
#include <sys/param.h>
#include <sys/file.h>
#include <sys/types.h>
#include <sys/extmem.h>
#include <signal.h>
#include <stdlib.h>
#include <sys/libvmic.h>
int main(int argc, char **argv)
{
        int             fd;
        char devname[1024];
        PCI5565M  *mem,*p;
        PCI5565R  *dev;
        PCI5565R  *intr;
        int i,mem_size,tst_size;
        struct timespec t0,t1;
        float real;

        if ( argc < 2 ) {
                strcpy(devname,"/dev/pci5565a");
        } else {
                strcpy(devname,argv[1]);
        }
        if ((fd = open(devname,O_RDWR)) == -1)
        {
                fprintf(stderr,"Device not found %s(%s)\n",
                devname,strerror(errno));
        exit(0);
        }
        printf("%s ",devname);
        if (pci5565_init(fd,&mem,&mem_size,&dev,&intr,1)<0)
        {
           fprintf(stderr,"Device initialize error %s(%s)\n",
           devname,strerror(errno));
           exit(0);
        }
        p=(PCI5565M*)valloc(mem_size);
        tst_size=0x10000;
        memset((void*)p,0xff,tst_size);
        for(i=0;i<tst_size/4;i++)
        {
          mem->access.d32[i]= i;
        }
        mlockall(MCL_FUTURE|MCL_CURRENT);
        readtime(&t0);
        for(i=0;i<tst_size/4;i++)
        {
           mem->access.d32[i]= p->access.d32[i];
        }
        readtime(&t1);
        adjust(&t0,&t1,&real);
        printf("32 bit write %f %f MB/s\n",real,(float)tst_size/real);

        readtime(&t0);
        for(i=0;i<tst_size/4;i++)
        {
           p->access.d32[i]= mem->access.d32[i];
        }
        readtime(&t1);
        adjust(&t0,&t1,&real);
        printf("32 bit read %f %f MB/s\n",real,(float)tst_size/real);
        readtime(&t0);
        for(i=0;i<tst_size/8;i++)
        {
           mem->access.d64[i]= p->access.d64[i];
        }
        readtime(&t1);
        adjust(&t0,&t1,&real);
        printf("64 bit write %f %f MB/s\n",real,(float)tst_size/real);
        readtime(&t0);
        for(i=0;i<tst_size/8;i++)
        {
           p->access.d64[i]= mem->access.d64[i];
        }
        readtime(&t1);
        adjust(&t0,&t1,&real);
        printf("64 bit read %f %f MB/s\n",real,(float)tst_size/real);
        readtime(&t0);
           memcpy((void*)mem,(void*)p,tst_size);
        readtime(&t1);
        adjust(&t0,&t1,&real);
        printf("memcpy write %f %f MB/s\n",real,(float)tst_size/real);
        readtime(&t0);
           memcpy((void*)p,(void*)mem,tst_size);
        readtime(&t1);
        adjust(&t0,&t1,&real);
        printf("memcpy read %f %f MB/s\n",real,(float)tst_size/real);
        readtime(&t0);
           memcpy((void*)&mem->access.d08[tst_size],(void*)mem,tst_size);
        readtime(&t1);
        adjust(&t0,&t1,&real);
        printf("memcpy read/write %f %f MB/s\n",real,(float)tst_size/real);
        readtime(&t0);
          rfmcpy((void*)mem,(void*)p,tst_size);
        readtime(&t1);
        adjust(&t0,&t1,&real);
        printf("rfmcpy write %f %f MB/s\n",real,(float)tst_size/real);
        readtime(&t0);
           rfmcpy((void*)p,(void*)mem,tst_size);
        readtime(&t1);
        adjust(&t0,&t1,&real);
        printf("rfmcpy read %f %f MB/s\n",real,(float)tst_size/real);
        readtime(&t0);
           rfmcpy((void*)&mem->access.d08[tst_size],(void*)mem,tst_size);
        readtime(&t1);
        adjust(&t0,&t1,&real);
        printf("rfmcpy read/write %f %f MB/s\n",real,(float)tst_size/real);
        if (pci5565_uninit(fd,mem,dev,intr)<0)
        {
                        fprintf(stderr,"%s\n",strerror(errno));
        }
        close(fd);
}
```

A: A 32bit write / A 32bit read
B: B 64bit write / B 64bit read
C: C memcpy write / C memcpy read / C memcpy read/write
D: D rfmcpy write / D rfmcpy read / D rfmcpy read/write

```
	/usr/local/CNC/drivers/extmem_start EXTMEM_VID0=0x114A EXTMEM_DID0=0x5565 EXTMEM_INT0=2
EXTMEM_ReadThrottleRate=-1 EXTMEM_WriteThrottleRate=-1 EXTMEM_EmptyThrottleRate=-1
	if [ -L /dev/pci5565a ]
	then
		rm -f /dev/pci5565a
	fi
	ln -s /dev/extmem0 /dev/pci5565a

	if [ -L /dev/pci5565b ]
	then
		rm -f /dev/pci5565b
	fi
	ln -s /dev/extmem1 /dev/pci5565b
```

EXTMEM_ReadThrottleRate=-1 (READ DMA 終了の監視の待ち時間：μ秒)
EXTMEM_WriteThrottleRate=-1(WRITE DMA 終了の監視の待ち時間：μ秒)
EXTMEM_EmptyThrottleRate=1(送信 FIFO EMPTY の監視の待ち時間：μ秒)

EXTMEM_ReadThrottleRate、EXTMEM_WriteThrottleRate は、
正の値の場合に、以下の式で、動的に DMA の終了時間を計算して待ちます。

```
  w = (length * 8) / 1000 + ThrottleRate;
```

EXTMEM_ReadThrottleRate、EXTMEM_WriteThrottleRate が負の値の場合には、設定した値の絶対値時間で待つようになります。
EXTMEM_EmptyThrottleRate は、正の値の絶対値時間で待つようになります。０以下の場合には、１を設定した場合と同じです。
これらの値は、ビジーウェイトの待ち時間の調整値です。
上記値は、jiffies のタイムアウト監視を、1μ秒の待ちループで行う事になります。

(3)サンプルが以下のシェルスクリプトに記述されています。

```
 # /usr/local/CNC/drivers/extmem/vmic/config_5565.sh
```

ＧＥ純正のドライバと同じＤＭＡ転送メソッドに設定する場合
この場合には、RedHawk のユーティリティ　　memexact -x -MS=サイズを実行して出てきた出力を/etc/grup.conf に追記する。

memexact -x -MS=128M を実行した場合の例

```
	memmap=exactmap memmap=0x8d800@0x10000 memmap=0x1ff00000@0x100000
memmap=0x1fe00000@0x20200000 memmap=0x8297c000@0x40200000
memmap=0x8000000$0xc2b7c000 memmap=0x60000#0xcaf9f000 memmap=0x1000@0xcafff000
```

このとき、128MB のメモリを予約することを要求しているのですが、この記述中の"memmap=0x8000000$0xc2b7c000"の部分が、このシステムのメモリ配置に合わせた設定です。
この値を、/etc.rc.local で

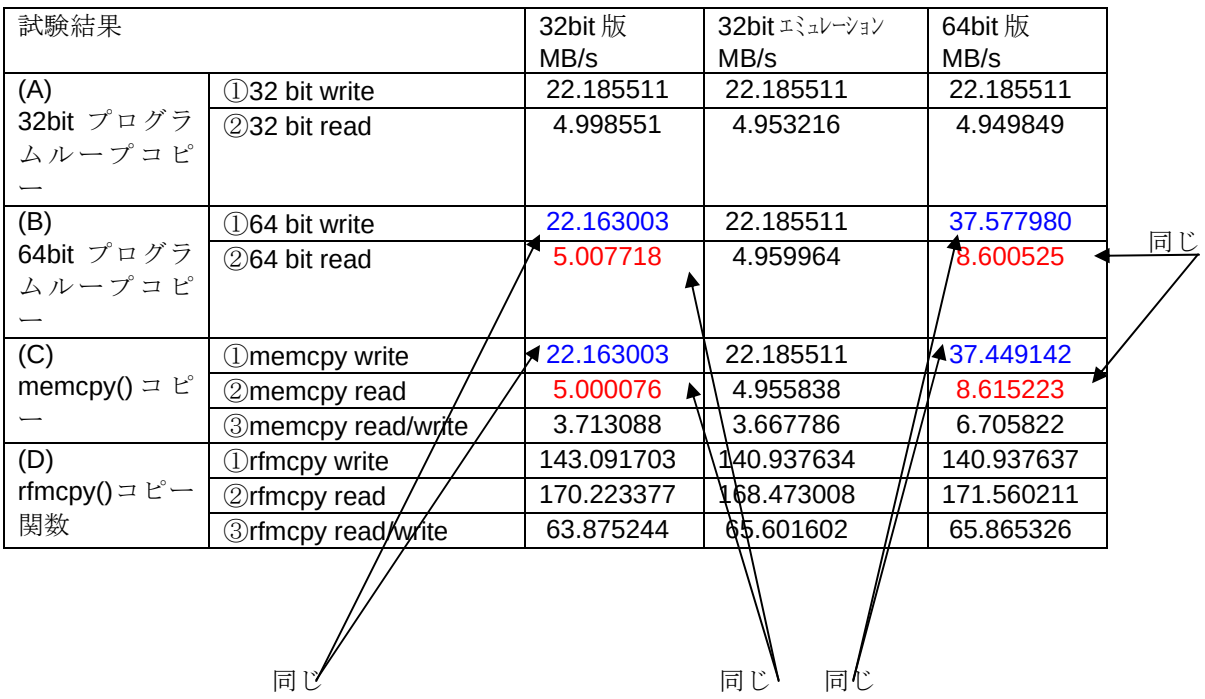
```
/usr/local/CNC/drivers/extmem/extmem_start EXTMEM_VID0=0x114A
EXTMEM_DID0=0x5565 EXTMEM_INT0=2 \
\
EXTMEM_ReadThrottleRate=-2 \
EXTMEM_WriteThrottleRate=-1 \
EXTMEM_EmptyThrottleRate=1 \
EXTMEM_RESADR0=0xc2b7c000 EXTMEM_RESLEN0=0x8000000
```

のように記述することで、予約メモリを設定できます。
このサンプルプログラムは、vmic/pci5565_resmem にあります。

4.2 試験結果

以下に、試験の結果を表にした。
32bit の結果と 32nit エミュレーションモードの結果はほぼ同じであり、誤差部分は数回の試行で変化する範囲にある。
64bit read/write の結果と memcpy()の結果を比較すると同じ OS では、同じ結果になり、32bit と 64bit の差はデータバス幅が原因であることが考えられる。
また、32bitOS と 64bitOS の結果を比較すると、(B)のプログラムコピーによる 64bit read あるいは write と(C)の memcpy()では、read あるいは write の方向に無関係に、約１．７倍の性能差があることを確認できる。

試験結果		32bit 版 MB/s	32bit ｴﾐｭﾚｰｼｮﾝ MB/s	64bit 版 MB/s
(A) 32bit プログラムループコピー	①32 bit write	22.185511	22.185511	22.185511
	②32 bit read	4.998551	4.953216	4.949849
(B) 64bit プログラムループコピー	①64 bit write	22.163003	22.185511	37.577980
	②64 bit read	5.007718	4.959964	8.600525
(C) memcpy() コピー	①memcpy write	22.163003	22.185511	37.449142
	②memcpy read	5.000076	4.955838	8.615223
	③memcpy read/write	3.713088	3.667786	6.705822
(D) rfmcpy()コピー関数	①rfmcpy write	143.091703	140.937634	140.937637
	②rfmcpy read	170.223377	168.473008	171.560211
	③rfmcpy read/write	63.875244	65.601602	65.865326



新規に作成した rfmcpy()関数は、状態によらず同じ性能であり、最も高速である。
この結果から、64 bit RedHawk を利用する場合には、アプリケーションプログラムを 64bit 対応に書き直さないと、性能向上は行われないことが理解できる。

4.3　チューニングパラメータを変更した場合の参考データ。

RFM の転送速度は、システムのメモリバスと PCI バスの速度に依存しています。
参考データとして、下記に、DELL T5500 での速度の結果を示します。

また、GE 純正の DMA 転送は、カーネルに用意した連続領域にあらかじめ用意されたバッファにデータを書き込み、それを 1 回の DMA（スレッショルドサイズ）で転送するものです。　（プログラムでは転送バッファを mmap した後の write になっているはずです。）
したがって、単純なプログラムでは、良いのですが、変数領域が複雑な処理では、DMA バッファへのプログラムコピーを繰り返すため速度が得られません。

このデバイスドライバの DMA 方式は、ユーザ空間からページ単位に直接 DMA する方式なので、どのような変数領域の位置であってもコピー可能です。

もし、同様の GE 純正の DMA を行う場合には、RedHawk の予約メモリと併用することで、同様の方式を行うことが出来ます。

EXTMEM_ReadThrottleRate=-2 EXTMEM_WriteThrottleRate=-1
EXTMEM_EmptyThrottleRate=1
rfmcpy read 1153308.000000 232.752625 MB/s
rfmcpy write 1897455.000000 141.471313 MB/s
rfmcpy read/write 3293432.000000 81.506302 MB/s

EXTMEM_ReadThrottleRate=-11 EXTMEM_WriteThrottleRate=-1
EXTMEM_EmptyThrottleRate=1
rfmcpy read 905300.000000 296.515472 MB/s
rfmcpy write 1893560.000000 141.762314 MB/s
rfmcpy read/write 3644449.000000 73.655975 MB/s

EXTMEM_ReadThrottleRate=-11 EXTMEM_WriteThrottleRate=-2
EXTMEM_EmptyThrottleRate=1
rfmcpy read 905259.000000 296.528900 MB/s
rfmcpy write 1904771.000000 140.927948 MB/s
rfmcpy read/write 3548295.000000 75.651955 MB/s

EXTMEM_ReadThrottleRate=-11 EXTMEM_WriteThrottleRate=-9
EXTMEM_EmptyThrottleRate=1
rfmcpy read 905228.000000 296.539062 MB/s
rfmcpy write 1885471.000000 142.370499 MB/s
rfmcpy read/write 3650314.000000 73.537636 MB/s

EXTMEM_ReadThrottleRate=-11 EXTMEM_WriteThrottleRate=-9
EXTMEM_EmptyThrottleRate=2
rfmcpy read 905201.000000 296.547913 MB/s
rfmcpy write 1895103.000000 141.646896 MB/s
rfmcpy read/write 3645624.000000 73.632240 MB/s
EXTMEM_ReadThrottleRate=-11 EXTMEM_WriteThrottleRate=-9
EXTMEM_EmptyThrottleRate=3
rfmcpy read 905205.000000 296.546600 MB/s
rfmcpy write 2026411.000000 132.468414 MB/s
rfmcpy read/write 3647752.000000 73.589287 MB/s

以下同じ値を適用した場合

#tyr -1
rfmcpy read 1358131.000000 197.650635 MB/s
rfmcpy write 1894377.000000 141.701187 MB/s
rfmcpy read/write 3432523.000000 78.203545 MB/s
#try -2
rfmcpy read 1153327.000000 232.748779 MB/s
rfmcpy write 1905454.000000 140.877426 MB/s
rfmcpy read/write 3209515.000000 83.637390 MB/s
#try -3
rfmcpy read 1230243.000000 218.197098 MB/s
rfmcpy write 1926511.000000 139.337616 MB/s
rfmcpy read/write 3515738.000000 76.352524 MB/s
try -4
rfmcpy read 1164783.000000 230.459625 MB/s
rfmcpy write 1907269.000000 140.743362 MB/s

rfmcpy read/write 3091622.000000 86.826736 MB/s
try -5
rfmcpy read 1349858.000000 198.862000 MB/s
rfmcpy write 1924531.000000 139.480972 MB/s
rfmcpy read/write 3419856.000000 78.493202 MB/s
try -6
rfmcpy read 1037665.000000 258.691833 MB/s
rfmcpy write 1952136.000000 137.508591 MB/s
rfmcpy read/write 3661367.000000 73.315636 MB/s
try -7
rfmcpy read 1197067.000000 224.244308 MB/s
rfmcpy write 1928021.000000 139.228485 MB/s
rfmcpy read/write 4047169.000000 66.326721 MB/s
try -8
rfmcpy read 1299052.000000 206.639496 MB/s
rfmcpy write 1954295.000000 137.356674 MB/s
rfmcpy read/write 3706926.000000 72.414574 MB/s
try -9
rfmcpy read 1457836.000000 184.132828 MB/s
rfmcpy write 1885264.000000 142.386139 MB/s
rfmcpy read/write 3390675.000000 79.168739 MB/s
try -10
rfmcpy read 1557643.000000 172.334396 MB/s
rfmcpy write 1936935.000000 138.587753 MB/s
rfmcpy read/write 3643810.000000 73.668892 MB/s
try -11
rfmcpy read 905264.000000 296.527252 MB/s
rfmcpy write 1934154.000000 138.787018 MB/s
rfmcpy read/write 3909415.000000 68.663841 MB/s
try -12
rfmcpy read 970993.000000 276.454559 MB/s
rfmcpy write 1878926.000000 142.866440 MB/s
rfmcpy read/write 4172657.000000 64.332024 MB/s
try -13
rfmcpy read 1036097.000000 259.083313 MB/s
rfmcpy write 1944587.000000 138.042404 MB/s
rfmcpy read/write 4432692.000000 60.558113 MB/s
try -14
rfmcpy read 1101565.000000 243.685532 MB/s
rfmcpy write 1888707.000000 142.126572 MB/s
rfmcpy read/write 4696049.000000 57.161980 MB/s
try -15
rfmcpy read 1167457.000000 229.931778 MB/s
rfmcpy write 1953709.000000 137.397873 MB/s
rfmcpy read/write 4956979.000000 54.153034 MB/s
try -16
rfmcpy read 1233236.000000 217.667542 MB/s
rfmcpy write 1898434.000000 141.398361 MB/s
rfmcpy read/write 5222706.000000 51.397774 MB/s
try 0
rfmcpy read 2281901.000000 117.636765 MB/s
rfmcpy write 2339619.000000 114.734688 MB/s
rfmcpy read/write 5088836.000000 52.749874 MB/s
try 1
rfmcpy read 2347380.000000 114.355347 MB/s
rfmcpy write 2404943.000000 111.618217 MB/s
rfmcpy read/write 5352785.000000 50.148746 MB/s

try 2
rfmcpy read 2412546.000000 111.266464 MB/s
rfmcpy write 2470150.000000 108.671722 MB/s
rfmcpy read/write 5616497.000000 47.794106 MB/s
try 3
rfmcpy read 2478350.000000 108.312164 MB/s
rfmcpy write 2535724.000000 105.861465 MB/s
rfmcpy read/write 5876856.000000 45.676712 MB/s
try 4
rfmcpy read 2543203.000000 105.550148 MB/s
rfmcpy write 2600990.000000 103.205109 MB/s
rfmcpy read/write 6137612.000000 43.736141 MB/s
try 5
rfmcpy read 2608583.000000 102.904701 MB/s
rfmcpy write 2666321.000000 100.676346 MB/s
rfmcpy read/write 6397232.000000 41.961189 MB/s
try 6
rfmcpy read 2675210.000000 100.341827 MB/s
rfmcpy write 2732697.000000 98.230965 MB/s
rfmcpy read/write 6663985.000000 40.281521 MB/s
try 7
rfmcpy read 2740932.000000 97.935829 MB/s
rfmcpy write 2799164.000000 95.898438 MB/s
rfmcpy read/write 6920579.000000 38.788006 MB/s
try 8
rfmcpy read 2806712.000000 95.640541 MB/s
rfmcpy write 2863597.000000 93.740654 MB/s
rfmcpy read/write 7187287.000000 37.348648 MB/s
try 9
rfmcpy read 2872038.000000 93.465149 MB/s
rfmcpy write 2929552.000000 91.630203 MB/s
rfmcpy read/write 7450416.000000 36.029594 MB/s
try 10
rfmcpy read 2937174.000000 91.392426 MB/s
rfmcpy write 2994970.000000 89.628761 MB/s
rfmcpy read/write 7713869.000000 34.799068 MB