External Memory Device Driver Installation on RedHawk Linux for the Generic PCI(e)-Boards Version 10.3 Release Note

August 26,2025

concurrent
REAL-TIME

1. 概要:

このドキュメントは、Concurrent Real Time (CCRT)製 RedHawk Linux 用 extemem デバイスドライバとそのサンプルプログラムの使用方法について記述されています。
extmem デバイスドライバは、単体でも動作しますが、通常は、各 I/O カード毎に作成済みのボードサポートパッケージの共通プラットフォームとして動作します。
　本デバイスドライバは、異なる OS の版でも共通の API を使用しています。

2. インストールに必要な条件:

以下に、インストールに必要な条件を示します.

- RedHawk Release 6.x.y 以上(i386,i686,x86_64 ーキテクチャのみ)
- RedHawk Release 7.x.y 以上(adm64,arm64 アーキテクチャのみ)
- RedHawk Release 8.x.y 以上(adm64,arm64 アーキテクチャのみ)
- RedHawk Release 9.x.y 以上(adm64,arm64 アーキテクチャのみ)

3. インストレーション:

　extmem デバイスドライバは、 CDROM あるいは DVD の rpm/deb フォーマットで供給されますので、以下の手順でインストールしてください。

インストールパッケージはご購入のボードサポートパッケージとアーキテクチャ毎に異なります:

X.Y	デバイスドライババージョン	10.3	2025/08/26 現在
Z	デバイスアクセスライブラリバージョン	A	2025/08/26 現在

バイナリパッケージ
  bin-extmem-X.Y_RH?.?-Z.x86_64.rpm　又は
  bin-extmem-rh?.?_X.Y_amd64.deb　又は
  bin-extmem-rh?.?_X.Y_arm64.deb

(注)RH?.? あるいは rh?.? は、RedHawk のバージョン番号です

開発パッケージ
  dev-extmem-X.Y_RH?.?-Z.x86_64.rpm　又は
  dev-extmem-rh?.?_X.Y_amd64.deb　又は
  dev-extmem-rh?.?_X.Y_arm64.deb

RPM パッケージの場合
```
>  === root ユーザで行ってください ===
# mount - ro /dev/sr0 /mnt
# cd /mnt
# rpm - ivh RPMFILE
# umount /mnt
```

DEB パッケージの場合
```
>  === root ユーザで行ってください ===
$ sudo -s
Password:
```

```
# mount - ro /dev/sr0 /mnt
# cd /mnt
# apt install ./DEBFILE
# umount /mnt
```

RPM パッケージのの実行例

 # rpm -ivh bin-extmem-10.3_RH9.2-A.x86_64.rpm

```
Verifying...                ################################# [100%]
準備しています...           ################################# [100%]
更新中 / インストール中...
   1: bin-extmem-10.3_RH9.2-A   ################################# [100%]
rm -f extmem.4.gz
cp extmem.4.UTF8 extmem.4
cp extmem.4.UTF8 extmem.4
gzip extmem.4
mkdir -p /usr/share/man/ja/man4
cp extmem.4.gz  /usr/share/man/ja/man4/extmem.4.gz
cp sys/extmem.h /usr/include/sys/extmem.h
cp sys/extmem_version.h /usr/include/sys/extmem_version.h
cp sys/libextmem.h /usr/include/sys/libextmem.h
cp libextmem32.a /usr/lib/libextmem.a
cp libextmem32.so /usr/lib/libextmem.so
cp libextmem.a /usr/lib64/libextmem.a
cp libextmem.so /usr/lib64/libextmem.so
rm -f extmem.4.gz
cp extmem.4.UTF8 extmem.4
cp extmem.4.UTF8 extmem.4
gzip extmem.4
mkdir -p /usr/share/man/ja/man4
cp extmem.4.gz  /usr/share/man/ja/man4/extmem.4.gz
cp sys/extmem.h /usr/include/sys/extmem.h
cp sys/extmem_version.h /usr/include/sys/extmem_version.h
cp sys/libextmem.h /usr/include/sys/libextmem.h
cp libextmem32.a /usr/lib/libextmem.a
cp libextmem32.so /usr/lib/libextmem.so
cp libextmem.a /usr/lib64/libextmem.a
cp libextmem.so /usr/lib64/libextmem.so
All the source for this product has been installed.
For explanations -- see Installation Notes.
```

DEB(adm64 アーキテクチャ)パッケージの実行例

 # apt install ./bin-extmem-rh9.2_10.3_amd64.deb

```
パッケージリストを読み込んでいます... 完了
依存関係ツリーを作成しています
状態情報を読み取っています... 完了
:
```

DEB(arm64 アーキテクチャ)パッケージの実行例

 # apt install ./bin-extmem-rh9.2_10.3_arm64.deb

```
パッケージリストを読み込んでいます... 完了
依存関係ツリーを作成しています
状態情報を読み取っています... 完了
:
```

extmem デバイスドライバは、*/usr/local/CNC/drivers* ディレクトリに展開されます。

次に、ボードあるいは、予約メモリを指定してドライバの組み込みを行います:

extmem で利用するボードのベンダーID および、ボード ID を lspci コマンドで調べます。

 # lspci (display all the PCI devices in the system)
02:0c.0 Network controller: VMIC Unknown device 5565

この情報から、デバイス ID=0x5565 であることがわかります。
次に、先頭の 02:0c.0 に着目して *lspci - n | grep 02:0c.0* の結果を検索してください。

02:0c.0 xxxx: 114a:5565

この情報から、ベンダーID=0x114A デバイス ID=0x5565 であることがわかります。

このデバイスを組み込むために下記の手順を行ってください。
通常、この手順は/etc/rc.local に組み込まれます。

> === root ユーザから ===
> /usr/local/CNC/drivers/extmem/extmem_start ¥
EXTMEM_VID0=0x114A EXTMEM_DID0=0x5565 EXTMEM_INT0=2
> ln -s /dev/extmem0 /dev/pci5565a
詳細は、man extmem のコンフィグレーションシェルスクリプト項目を参照してください。

注意 *:*
x86_64 アーキテクチャの RedHawk7 をお使いの場合で、レガシィ PCI ボードをお使いの場合には、/etc/default/grub の GRUB_CMDLINE_LINUX 行に pci=routeirq を追加してください。

RedHawk6.x までは、直接/etc/grub.conf あるいは/etc/grub2-efi.cfg に記述できますが、RedHawk7.x 以降は、直接編集できません。

下記コマンドを実行することですべてのエントリに追加されます。
 # grub2-mkconfig -o /etc/grub2-efi.cfg

grub2-mkconfig だけだと内容が標準出力に出力されます。

また、/etc/rc.local から組み込むためには、以下のコマンドで/etc/rc.local に実行権を与えてください
 # chmod u+x /etc/rc.d/rc.local

または、予約メモリを使用できます。

 # /usr/local/CNC/drivers/extmem/extmem_start ¥
EXTMEM_RESADR0=RESERVED_MEMORY_ADDRESS ¥
EXTMEM_RESLEN0=RESERVED_MEMORY_SIZE

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 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
 # memexact -x -MS=128M

memmap=exactmap memmap=0x9f000@0x1000 memmap=0xf00000@0x100000 memmap=0x4fe56018@0x411b000
memmap=0x1fc40@0x53f710.38 memmap=0x3c0@0x53f90c58 memmap=0x10440@0x53f910.38
memmap=0x8ba8@0x53fa1458 memmap=0xdad4000@0x58976000 memmap=0x53d000#0x693ea000
memmap=0x3bdc000@0x6c424000 memmap=0xb38000000@0x100000000 memmap=0x8000000$0xc38000000

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$0xc38000000"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_RESADRO *0xc38000000* EXTMEM_RESLENO=*0x8000000*
のように記述することで、予約メモリを設定できます。

(組み込みに成功したかどうかは dmesg で確認してください).

> lsmod　　*(this command will display the **extmem** driiver)*

試験プログラムの例は、以下のディレクトリにあります:

```
> === as user ===
> cd /usr/local/CNC/drivers/extmem_driver/resmem
> make clean
> make
> cd ../dio
> make
```

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
Generating grub configuration file ...
:
done
```

KABI の変更あるいは、CUSTOM RedHawk kernel 用にコンパイルする場合

RedHawk カーネルソースコードに対して、パッチを適用した場合や、RedHawk のフレーバーを custom などに変更した場合には、下記の手順でドライバを再構築してください。

```
> === root ユーザで行ってください ===
> cd /usr/local/CNC/drivers/extmem
> make rebuild
```

/usr/local/CNC/drivers/extmem/driver_x86_64 下に再構築されたドライバが生成されます。
開発パッケージでは、以下の手順で、新しいデバイスドライバを構築することが出来ます。

```
> === root ユーザで行ってください ===
> cd /usr/local/CNC/drivers/extmem
> make config
> make
> make install
```

また、IOCTL_EXTMEM_SET_SIGNAL_INFO を併用すると、割り込み原因となったデータを同時に配信することが出来るようになりました。詳細は、IOCTL の(13)に記述されています。
なお、ディフォルト設定では従来の SIGIO 配信のままです。

4. 各リリースの変更点

シグナルイベント配信機構の変更について

Ver8.3 以降の版では、シグナルイベントの配信が SIOGIO のみから IOCTL_EXTMEM_SET_SIGNAL_NUMBER を使用してリアルタイムシグナルを利用して、シグナルイベントを配信可能になりました。
SIGIO では、ユーザプロセスが、extmem 割り込みハンドラの高速な要求に応答できずに、シグナルイベントが失われ、結果としてシグナルハンドラで pending が 0 にならないことがありましたが、リアルタイムシグナルを使用することで、イベントがキューイング配信されるようになりました。

V8.4A の変更点

RedHawk7.3 対応
社名変更(プログラム、ヘッダーおよび、マニュアルと Makefile 等)

- CONCURRENT -> CCRT
- Concurrent Computer Corporation -> Concurrent Real Time Inc
- Ge -> Abaco
- RedHawk7.X のマニュアルディレクトリを/usr/man から/usr/shar/man/ja 下に変更

V8.4B の変更点

7.04 対応

IOCTL_EXTMEM_RESON_DATAH, IOCTL_EXTMEM_RESON_DATAH_AUX, IOCTL_EXTMEM_RESON_DATAL, IOCTL_EXTMEM_RESON_DATAL_AUX,追加

新規 BSP

z30046 pexh291388 pexh284011 pexh224020

V8.4D の変更点

7.5 対応

abaco/pci5576　abaco/pci5587 の削除

queued signal info.si_ptr　に割込み理由の値をセットした。

V8.4E の変更点

queued signal info.si_pid に割込み理由の位置をセットした。

V8.4F の変更点

extmem_start custom サポート

新規 BSP　　pexh2994w　　pexh251500

V8.4G の変更点

extmem_5565.c get_user_pages() パラメータ変更

新規 BSP なし

V8.4H の変更点

iterface/*_mmap.c *_write_data()関数の後に、 200 マイクロ秒のディレイを追加

(仕様的には、120 マイクロ秒)

これがないと、過度的に異なる値(前回値との OR の値)をレジスタから読み出す

新規 BSP　　PEX-251100

V9.0A の変更点

RedHawk 8.0 サポート

sys/extmem.h　で REDHAWK_VERSION を定義した

例えば RedHawk8.0 では、REDHAWK_VERSION は、80 に定義されている

新規 BSP　　なし

V9.1A の変更点

(1) RedHawk8.2 linux kernel5.4 対応

access_ok()のマクロ変更に伴い、ACCESS_OK に変更 extmem_5565.c

do_gettimeofday()の廃止に伴い、jiffies に変更 extmem_main.c extmem_compat.c

(2) 追加

/dev/extmem?を自動的に生成する。ただし、REDHAWK7.5 以上

(3) バグ修正

compat_ioctl()は、32 ビット時と 64 ビット時で command 部分のサイズマクロで値が異る

このマクロによって影響を受けるのは、以下の_IOWR のみ(_IO マクロでは影響を受けない)

```
#define IOCTL32_EXTMEM_RESON_DATAH      _IOWR(IOCTL_EXTMEM,32,int *)
#define IOCTL32_EXTMEM_RESON_DATAH_AUX  _IOWR(IOCTL_EXTMEM,33,int *)
#define IOCTL32_EXTMEM_RESON_DATAL      _IOWR(IOCTL_EXTMEM,34,int *)
#define IOCTL32_EXTMEM_RESON_DATAL_AUX  _IOWR(IOCTL_EXTMEM,35,int *)
```

新規 BSP　　なし

V9.1B の変更点

```
void * x86_64_memcpy(void *dest, const void *src, size_t len);
void * x86_64_memset(void *dest, int val, size_t len);
```

の追加と pci5565_mmap.c pci5565_mtest での利用

新規 BSP　　なし

V9.2A の変更点

RedHawk8.4 のサポートと kdump モジュールの削除

新規 BSP　　なし

バグ修正

abaco/pci5565/pci5565_mmap.c で、割り込みに対する Sender Data 取得が機能していない。
SID より先に ISD を読まないとデータを損失する。

V9.3A の変更点

Ubuntu18.04(arm64)/Ubuntu20.04(amd64) のサポート
sys/extmem.h の REDHAWK_VERSION チェック修正(RHEL_MINOR == 6 は 8.4)
Makefile のオプティマイズから-minline-all-stringops -msse2 を削除
extmem_start 中で、mknod の umask を 022 から 0 に変更したこれにより、/dev/extmem*の権限が -rw-rw-rw に変更された。
EXTMEM_INTn=3 以上で IRQ の強制割当する機能をマニュアルに追加した。
arm64 アーキテクチャである時 I/O ports 領域が[disabled]であっても、
強制的に enable にする機能を追加した。
gcc の警告文に対処
新規 BSP　　なし

V9.3B の変更点

extmem_5565.c
get_user_pages()を pin_user_pages_fast()に変更
この変更を行わないと、5565DMA のオフセットが 0 以外で失敗する。
Before

```
ret = get_user_pages((unsigned long)user_buffer & PAGE_MASK,
        nr_pages, FOLL_FORCE,pages, NULL);
```

After

```
ret = pin_user_pages_fast((unsigned long)user_buffer & PAGE_MASK,
        nr_pages, FOLL_FORCE | FOLL_LONGTERM, pages);
```

abaco/pci5565/Makefile
abaco/pci5565/test1/Makefile
abaco/pci5565/test2/Makefile
OPTIMIZE オプションを下記に変更した
OPTIMIZE = -O3 -mavx -minline-all-stringops -mforce-indirect-call -mmemcpy-strategy=vector_loop:-1:align
abaco/pci5565/test1/pci5565_mtest の結果を示す
Before -O3

```
32 bit write 2615.177979 25.059862 MB/s
32 bit read 16114.190430 4.066975 MB/s

64 bit write 1276.357056 51.346134 MB/s
64 bit read 8069.967773 8.120975 MB/s

memcpy write 10543.628906 6.215697 MB/s                 <====
memcpy read 64483.726562 10.36318 MB/s                  <====
memcpy read/write 76446.320312 0.857281 MB/s            <====

x86_memcpy write 2593.787109 25.266531 MB/s
x86_memcpy read 16130.158203 4.062948 MB/s
x86_memcpy read/write 19131.416016 3.425570 MB/s

x86_64_memcpy write 688.674988 95.162453 MB/s
x86_64_memcpy read 4139.767090 15.830842 MB/s
x86_64_memcpy read/write 76428.460938 0.857482 MB/s     <====

rfmcpy write 823.275024 79.604019 MB/s
rfmcpy read 724.562988 90.449005 MB/s
```

```
                        rfmcpy read/write 2015.977051 32.508308 MB/s
                After -O3 -mavx -minline-all-stringops -mforce-indirect-call -mmemcpy-
strategy=vector_loop:-1:align
                        32 bit write 2615.246094 25.059210 MB/s
                        32 bit read 16125.161133 4.064208 MB/s

                        64 bit write 1276.308960 51.348068 MB/s
                        64 bit read 8077.397949 8.113503 MB/s

                        memcpy write 393.332001 166.617508 MB/s                 <====
                        memcpy read 2172.479004 30.166460 MB/s                  <====
                        memcpy read/write 2607.989990 25.128931 MB/s            <====

                        x86_memcpy write 2593.724121 25.267143 MB/s
                        x86_memcpy read 16133.715820 4.062052 MB/s
                        x86_memcpy read/write 19128.400391 3.426110 MB/s

                        x86_64_memcpy write 688.7310.38 95.154709 MB/s
                        x86_64_memcpy read 4146.976074 15.803323 MB/s
                        x86_64_memcpy read/write 2603.000000 25.177103 MB/s     <====

                        rfmcpy write 824.487976 79.486908 MB/s
                        rfmcpy read 724.353027 90.475220 MB/s
                        rfmcpy read/write 2005.026978 32.685844 MB/s
```

Note:

-mstringop-strategy=alg

文字列操作のインライン化に使用する特定のアルゴリズムの内部決定ヒューリスティックをオーバーライドします。

alg に指定できる値は次のとおりです。

'rep_byte'

'rep_4byte'

'rep_8byte'

指定されたサイズの i386 rep プレフィックスを使用して展開します。

'byte_loop'

'loop'

'unrolled_loop'

インライン ループに展開します。

'libcall'

常にライブラリ呼び出しを使用します

'unrolled_loop'

'vector_loop'

-mmemcpy-strategy=strategy

strategy は、「alg:max size:dest align」トリプレットのカンマ区切りのリストです。

内部決定ヒューリスティックをオーバーライドして、__builtin_memcpy をインライン化する必要があるかどうか、およびコピー操作の予想サイズがわかっている場合に使用するインライン アルゴリズムを決定します。

alg は「-mstringop-strategy」で指定されるものと同様です。

'max size'は、インライン アルゴリズム alg が許可される最大バイト サイズを指定します。

libcall の場合、'max size'は -1 にする必要があります。
リスト内のトリプレットの'max size'は、昇順で指定する必要があります。
alg の最小バイト サイズは、最初のトリプレットでは 0 で、前の範囲では max_size + 1 です。

max_size コンポーネントは、インライン アルゴリズム alg が許可される最大バイトサイズを指定すると言われていますが、
dest_align コンポーネントは文書化されていません

下記に、調査結果を示します。

```
-mmemcpy-strategy=libcall:-1:noalign
-mmemcpy-strategy=vector_loop:3000:align,libcall:-1:align
-mmemcpy-strategy=libcall:-1:align
-mmemcpy-strategy=vector_loop:2000:align,libcall:-1:align
-mmemcpy-strategy=rep_8byte:-1:noalign
-mmemcpy-strategy=vector_loop:-1:align
```

出典

https://gcc.gnu.org/onlinedocs/gcc-8.4.0/gcc.pdf
https://gcc.gnu.org/legacy-ml/gcc/2019-01/msg00147.html

新規 BSP

なし

V9.3C の変更点

Rocky の dnf バージョンアップにより、/usr/include/linux/version.h の
RHEL_MINOR の値 4 から 5 以上（2023.01.24 現在 7）にアップグレードされたため、以下のマクロを sys/extmem.h に追加した

```
#if RHEL_MINOR == 5
#define REDHAWK_VERSION 84
#endif
#if RHEL_MINOR == 6
#define REDHAWK_VERSION 84
#endif
#if RHEL_MINOR == 7
#define REDHAWK_VERSION 84
#endif
#if RHEL_MINOR == 8
#define REDHAWK_VERSION 84
#endif
#if RHEL_MINOR == 9
#define REDHAWK_VERSION 84
#endif
#if RHEL_MINOR == 10
#define REDHAWK_VERSION 84
#endif
```

interface/pex361116/pex361116_libs.c pex361116_uninit()関数の unmap 順を修正

新規 BSP

pex340416

V9.4A の変更点

MSI のサポート
EXTMEM_MSI*=1

新規 BSP
なし

V9.4B の変更点

8.x の pci5565 の Makefile で gcc オプション
-mforce-indirect-call -mmemcpy-strategy=vector_loop:-1:align
を追加したが、このオプションは 8.x 以外では利用できないため、pci5565/Makefile の条件設定を追加した

7.2 fips モードに対応した
extmem_start.bin extmem_start Makefile.abaco Makefile が変更されている

driver_x86_64/extmem.fips.7.2.ko は、 シグニチャが付与されていないが、
driver_x86_64.signing/extmem.fips.7.2.ko は、 以下のシグニチャが付与されている

```
author:         Concurrent Real Time Inc.
license:        Concurrent Real Time
srcversion:     2F2C7CC38AE0054DCA92BEA
depends:
vermagic:       4.1.15-rt17-RedHawk-7.2-fips SMP preempt mod_unload
signer:         Concurrent Real-Time: RedHawk Linux kernel signing key
sig_key:        87:F9:C6:28:A5:DE:00:AB:F5:70:F0:A4:E5:5C:6F:35:4B:7F:13:FC
sig_hashalgo:   sha512
```

シグニチャの付与には、キーが必要であるがこれは提供されない。
なお、キーをユーザが付与するためには、fips カーネルを再コンパイルの上、以下のコマンドを実行する必要がある。

```
/lib/modules/4.1.15-rt17-RedHawk-7.2-fips/build/scripts/sign-file sha512 ¥
    /lib/modules/4.1.15-rt17-RedHawk-7.2-fips/build/signing_key.priv ¥
    /lib/modules/4.1.15-rt17-RedHawk-7.2-fips/build/signing_key.x509 ¥
    /usr/local/CNC/drivers/extmem/extmem.fips.7.2.ko
```

新規 BSP
なし

V10.0A の変更点

CentOS7.5 , Rocky8 , Rocky9 kernel のサポート(非売品)
RedHawk7.5 がコンパイル出来無いバグを修正
RedHawk9.2 のサポート
(1) kene.org が、以下のカーネル関数を GPL 以外で利用不可にしたため

```
ERROR: modpost: GPL-incompatible module extmem.ko uses GPL-only symbol 'pin_user_pages_fast'
ERROR: modpost: GPL-incompatible module extmem.ko uses GPL-only symbol '__class_create'
ERROR: modpost: GPL-incompatible module extmem.ko uses GPL-only symbol 'class_destroy'
```

ERROR: modpost: GPL-incompatible module extmem.ko uses GPL-only symbol 'device_create'
ERROR: modpost: GPL-incompatible module extmem.ko uses GPL-only symbol 'device_destroy'
GPL 関数の利用を停止した
このための見た目の不利益は無いが、extmem_start でノードを作成することが必須になった。

(2) RedHawk9.2 では Debug カーネルが無くなった。
(3) 32 ビットライブラリ生成時のコンパイルオプションに -fPIC 追加
(4) extmem_start 時のカラーを変更
(5) 全 Makefile の修正
(6) 全 extmem_start の修正

新規 BSP
なし
サンプルプログラム追加
./pexh291388/pexh291388_500usec.c

V10.1A の変更点

V10.1A 用 リリースノート 2025.03.11
V10.1A の変更点
MSI Interrupt の初期化関数を追加 extmem_ht_enable_msi_mapping() extmem_enable_ht()
32bitDMA を可能にするメモリ領域を確保する EXTMEM_DMA0=サイズ を追加
将来への拡張ファイル extmem_dma32.c を追加(中身は無)
キュードシグナルのフィールドを変更(info.si_pid ->info.si_errno)
Active Low の利用していない AUX を Active High へ変更
IOCTL_EXTMEM_RESON_POSITIONL_AUX -> IOCTL_EXTMEM_RESON_POSITION2_AUX
IOCTL_EXTMEM_RESON_MASKL_AUX -> IOCTL_EXTMEM_RESON_MASK2_AUX
IOCTL_EXTMEM_RESON_DATAL_AUX -> IOCTL_EXTMEM_RESON_DATA2_AUX

新規 BSP
sca010
変更 BSP
z30046 (要 V10.3A) IOCTL_EXTMEM_RESON_POSITIONL_AUX -> IOCTL_EXTMEM_RESON_POSITIONL
IOCTL_EXTMEM_RESON_MASKL_AUX -> IOCTL_EXTMEM_RESON_MASKL
IOCTL_EXTMEM_RESON_DATAL_AUX -> IOCTL_EXTMEM_RESON_DATAL
PEXH321416N MSI 動作確認
PEX361116 DA/AD/32bitDMA サポート(要 V10.1A)
PEX361116N DA/AD/32bitDMA サポート(要 V10.1A) MSI 動作確認(但し N の無いカードとので混在は、MSI の指定は出来無い)

上記追加、変更 API を利用するカードの rpm build に係る REQUIRES フィールドを変更した
rpm.build.script/
├── bin-extmem-pack
├── bin-target-pack 変更
├── dev-abaco-pack
├── dev-extmem-pack
├── dev-full-pack
└── dev-target-pack 変更
RedHawk7.2-fips カーネルのシグニチャー付与を自動化した

V10.2A の変更点

5565 の Spurious Interrupt(!ICSR_INTERRUPT_INPUT_ACTIVE) チェックを追加
extmem_5565_dma_handler() extmem_irq_handler()

従来 necrfm_irq_handler()関数中で直接 RMS のキュードシグナルの配信を行っていた。
この方式では、デバイスを利用しているプロセスに、シグナルが配信されるが、
read/write 中にシグナルを受診すると、DMA が停止する不具合が確認された。
read/write の DMA 完了割り込み待ち関数である、wait_event_interruptible()がシグナルを受信した場合に、
関数が終了し、その際戻り値が、ERESTART を返す。
従って、wait_event_interruptible()が ERESTART を返す場合にリトライすることで対処する。

新規 BSP

なし

変更 BSP

なし

V10.3A の変更点

V10.3A 用　リリースノート 2025.08.26
V10.3A の変更点
　　Octagon 社の PCIE5565APIORC の DMA サポート
　　Abaco 社の PCI5565 の DMA 定義(101-102: EXTMEM_DMA_VID0 = 0x114A, EXTMEM_DMA_DID0 = 0x5565)を不要にした
新規 BSP
　　pcie5565(Octagon 社の PCIE5565APIORC)
変更 BSP
　　abaco/pci5565/pci5565_mmap.c　　DMA メモリテスト追加、clear/fill 機能を DMA 化した
　　abaco/pci5565/test1/pci5565_stest.c　　DMA メモリテスト追加
　　abaco/pci5565/pci5565_libs.c　　Octagon/Abaco 共通チェック追加

既知の問題

DMA アクセスを行うデバイスからのトラフィックが非常に高い場合に、以下のようなエラーがコンソールに出力される場合がある。

pci 0000:65:00.0: swiotlb buffer is full (sz: 4096 bytes), total 32768 (slots), used 32768 (slots)

このエラーの原因は、バススロットから、Octagon PCIE5565 と特定することが出来る。

lspci -s 0000:65:00.0
0000:65:00.0 Network controller: Device 1f8a:5565

このエラーメッセージは、swiotlb バッファが一杯になってしまったことを示している。この問題を解決するためには、swiotlb バッファのサイズを増やす必要がある。ただし、どれだけ増やすべきかは、ハードウェアとワークロードによって異なる。
ディフォルトでは、下記のように swiotlb バッファは 64MB である

dmesg | grep "software IO TLB"
[　0.077301] software IO TLB: area num 8.
[　14.793560] software IO TLB: mapped [mem 0x000000006244a000-0x000000006644a000] (64MB)

下記コマンドで、256MB に設定することが出来る。

blscfg --kopt-add="swiotlb=131072"

再起動後、下記のように swiotlb バッファは１２８MB になる。
dmesg | grep "software IO TLB"
[　0.076448] software IO TLB: area num 8.
[　14.815386] software IO TLB: mapped [mem 0x0000000033000000-0x0000000043000000] (256MB)

5. マニュアル

extmem(4)
Kernel Interfaces Manual
extmem(4)

名前

extmem - external & reserved memory device driver with interrupt
version 10.3

書式

#include <sys/extmem.h>

gcc [options ...] file -lextmem ...

機能解説

/dev/extmem[0-25] は、PCIbus 上のボードをアプリケーションプログラムから 26 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[10.34];
                 volatile unsigned long long int reg64[ 512];
            } access;
} extmem;

if ((fd=open("/dev/extmem0",O_RDWR)) < 0)
{
        fprintf(stderr,"can't open /dev/extmem/0¥n");
        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、スキャッター/ギャザー型の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。

0)PCI5565,PMC5565 のみ動作し、リザーブドメモリとの 物理アドレス指定のダイレクト転送を行います。

```
ret = ioctl(fd,IOCTL_EXTMEM_5565_READ,extmem_5565_resmem_transfer_t*);
ret = ioctl(fd,IOCTL_EXTMEM_5565_WRITE,extmem_5565_resmem_transfer_t*);

typedef struct
{
        unsigned long long buffer; /* PCIbus 物理アドレス */
        unsigned int offset;  /* リフレクティブメモリオフセット */
        size_t count;                 /* 転送サイズ */
} extmem_5565_resmem_transfer_t;
```

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)割り込みの発生をシグナルで通知させます。

```
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)シグナル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; //シグナルの発生回数
} 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_INTRO,extmem_buswindow_t *);
ret = ioctl(fd,IOCTL_EXTMEM_BUSWINDOW_CLOSE,extmem_buswindow_t *);
ret = ioctl(fd,IOCTL_EXTMEM_BUSWINDOW_IINFO,extmem_buswindow_t *);
ret = ioctl(fd,IOCTL_EXTMEM_RESON_POSITION,int *);
ret = ioctl(fd,IOCTL_EXTMEM_RESON_POSITIONL,int *);
ret = ioctl(fd,IOCTL_EXTMEM_RESON_POSITION_AUX,int *);
ret = ioctl(fd,IOCTL_EXTMEM_RESON_POSITIONL_AUX,int *);
ret = ioctl(fd,IOCTL_EXTMEM_RESON_MASK,int *);
ret = ioctl(fd,IOCTL_EXTMEM_RESON_MASKL,int *);
ret = ioctl(fd,IOCTL_EXTMEM_RESON_MASK_AUX,int *);
```

```
ret = ioctl(fd,IOCTL_EXTMEM_RESON_MASKL_AUX,int *);
ret = ioctl(fd,IOCTL_EXTMEM_KILL,NULL);
```

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_IINFO を使う必要があります。
IOCTL_EXTMEM_BUSWINDOW_IINFO と IOCTL_EXTMEM_BUSWINDOW_INTRO 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_POSI-
TION,IOCTL_EXTMEM_RESON_POSITION_AUX 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,IOCTL_EXTMEM_RESON_MASK_AUX
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,IOCTL_EXTMEM_RESON_MASK_AUX のディフォルト値は 0 です。 割り込み理由の値がアクティブロウである場合には、IOCTL_EXTMEM_RESON_POSI-
TIONL,IOCTL_EXTMEM_RESON_MASKL,IOCTL_EXTMEM_RESON_POSITIONL_AUX,IOCTL_EXTMEM_RESON_MASKL_AUX を利用してください。
IOCTL_EXTMEM_KILL は、ソフトウェアシグナル SIGIO を発生させます。（通常デバッグ目的にしか使用しません）

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_INTRO,&wcmd)<0)
{
        fprintf(stderr,"extmem infomation fail %s¥n",strerror(errno));
        return(-1);
}
pos = 1;
if(ioctl(fd,IOCTL_EXTMEM_RESON_POSITION,&pos)<0)
{
        fprintf(stderr,"extmem fd,IOCTL_EXTMEM_RESON_POSITION,&pos fail %s¥n",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 %s¥n",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_IINFO,&wcmd)<0)
{
        fprintf(stderr,"extmem infomation fail %s¥n",strerror(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 %s¥n",strerror(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);
```

13)割り込みシグナル番号を操作する

```
ret = ioctl(fd,IOCTL_EXTMEM_SET_SIGNAL_NUMBER,int *arg);
ret = ioctl(fd,IOCTL_EXTMEM_GET_SIGNAL_NUMBER,int *arg);
ret = ioctl(fd,IOCTL_EXTMEM_SET_SIGNAL_INFO,int *arg);
ret = ioctl(fd,IOCTL_EXTMEM_RESON_DATAH,int *arg);
ret = ioctl(fd,IOCTL_EXTMEM_RESON_DATAH_AUX,int *arg);
```

```
ret = ioctl(fd,IOCTL_EXTMEM_RESON_DATAL,int *arg);
ret = ioctl(fd,IOCTL_EXTMEM_RESON_DATA2_AUX,int *arg);
```

通常 extmem デバイスドライバは SIGIO を使用して、デバイスドライバからイベント通知を行いますが、これを別のシグナル番号に変更したい場合に利用します。
通常、この呼び出しは、open の直後に行い、close 時には、SIGIO に戻されます。
SIGRTMIN(34)から SIGRTMAX(64)までの値を IOCTL_EXTMEM_SET_SIGNAL_NUMBER で設定すると、
シグナルがキューイングされ、リアルタイムシグナル以外のシグナルで発生していた、イベントのオーバーライトを避けることができます。
加えて IOCTL_EXTMEM_SET_SIGNAL_INFO で 0 以外の値に設定すると、
以下の論理積が 0 以外であると、

- IOCTL_EXTMEM_RESON_POSITION の結果と IOCTL_EXTMEM_RESON_MASK
- IOCTL_EXTMEM_RESON_POSITION_AUX の結果と IOCTL_EXTMEM_RESON_MASK_AUX
- IOCTL_EXTMEM_RESON_POSITION2 の結果と IOCTL_EXTMEM_RESON_MASK2
- IOCTL_EXTMEM_RESON_POSITIONL_AUX の NOT 結果と IOCTL_EXTMEM_RESON_MASKL_AUX

ユーザ割り込みハンドラに

- IOCTL_EXTMEM_RESON_POSITION,
- IOCTL_EXTMEM_RESON_POSITIONL,
- IOCTL_EXTMEM_RESON_POSITION_AUX,
- IOCTL_EXTMEM_RESON_POSITION2_AUX

の結果が info->si_errno(RedHawk7 では si_pid)に info->si_ptr に割り込み原因がキャストされて配信されます。
の結果が info->si_errno(RedHawk8 では si_errno)に info->si_ptr に割り込み原因がキャストされて配信されます。
ただし SIGIO では、複数プロセスへのシグナル配信が可能でしたが、それ以外のシグナルでは最後に登録したプロセスのみシグナルイベントを受信できます。
IOCTL_EXTMEM_RESON_DATAH, IOCTL_EXTMEM_RESON_DATAH_AUX, IOCTL_EXTMEM_RESON_DATAL, IOCTL_EXTMEM_RESON_DATA2_AUX は、ドライババッファの初期値をセットします。

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 parameter¥n");
      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 bytes¥n",
                  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 bytes¥n",
                  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 %s¥n",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。

特殊なケースとして、 base_address = 0x1000 * DMA32 の場合には、
コンフィグ時に EXTMEM_DMAn=65536 で確保した 32bit address 領域のマッピングを行います。
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 - %s¥n",strerror(errno));
                return(-1);
        }
        sigdelset(&set,SIGIO);
        if (sigprocmask(SIG_SETMASK,&set,&oset)==(-1))
        {
                fprintf(stderr, "Cannot set sigprocmask - %s¥n",strerror(errno));
                return(-1);
        }

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

```
# memexact -x -MS=128M
memmap=exactmap memmap=0x9c400@0x1000 memmap=0xe7cf9c00@0x100000 memmap=0x2000#0xe7e4bc00
memmap=0x210000000@0x100000000 memmap=0x8000000$0x310000000
```

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$0x310000000"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=0x310000000 EXTMEM_RESLEN0=0x01000000 のように記述することで、予約メモリを設定できます。

予約されたメモリは、/proc/extmem を見る事で確認できます。

```
# cat /proc/extmem
version: 8.2F                                                    <====6.8E 以降はこの部分にバージョンとリリース番号を表示
built  : Nov  4 2015, 16:33:44
boards : 2
license: Development
extmem0: VID=0x114A DID=0x5565 SHARED=0 BUSSLOT=02:00.0  <====8.2F 以降はこの部分にベンダーID とデバイス ID、共有フラグ,バススロットを表示
extmem1: VID=0x0000 DID=0x0000 SHARED=0 BUSSLOT=24:0c.0
extmem2: VID=0x0000 DID=0x0000 SHARED=0 BUSSLOT=00:00.0
extmem3: VID=0x0000 DID=0x0000 SHARED=0 BUSSLOT=00:00.0
extmem4: VID=0x0000 DID=0x0000 SHARED=0 BUSSLOT=00:00.0
extmem5: VID=0x0000 DID=0x0000 SHARED=0 BUSSLOT=00:00.0
extmem6: VID=0x0000 DID=0x0000 SHARED=0 BUSSLOT=00:00.0
extmem7: VID=0x0000 DID=0x0000 SHARED=0 BUSSLOT=00:00.0
extmem8: VID=0x0000 DID=0x0000 SHARED=0 BUSSLOT=00:00.0
extmem9: VID=0x0000 DID=0x0000 SHARED=0 BUSSLOT=00:00.0
extmem10: VID=0x0000 DID=0x0000 SHARED=0 BUSSLOT=00:00.0
extmem11: VID=0x0000 DID=0x0000 SHARED=0 BUSSLOT=00:00.0
extmem12: VID=0x0000 DID=0x0000 SHARED=0 BUSSLOT=00:00.0
extmem13: VID=0x0000 DID=0x0000 SHARED=0 BUSSLOT=00:00.0
extmem14: VID=0x0000 DID=0x0000 SHARED=0 BUSSLOT=00:00.0
extmem15: VID=0x0000 DID=0x0000 SHARED=0 BUSSLOT=00:00.0
extmem16: VID=0x0000 DID=0x0000 SHARED=0 BUSSLOT=00:00.0
extmem17: VID=0x0000 DID=0x0000 SHARED=0 BUSSLOT=00:00.0
extmem18: VID=0x0000 DID=0x0000 SHARED=0 BUSSLOT=00:00.0
extmem19: VID=0x0000 DID=0x0000 SHARED=0 BUSSLOT=00:00.0
extmem20: VID=0x0000 DID=0x0000 SHARED=0 BUSSLOT=00:00.0
extmem21: VID=0x0000 DID=0x0000 SHARED=0 BUSSLOT=00:00.0
extmem22: VID=0x0000 DID=0x0000 SHARED=0 BUSSLOT=00:00.0
extmem23: VID=0x0000 DID=0x0000 SHARED=0 BUSSLOT=00:00.0
extmem24: VID=0x0000 DID=0x0000 SHARED=0 BUSSLOT=00:00.0
extmem25: VID=0x0000 DID=0x0000 SHARED=0 BUSSLOT=00:00.0
resadr0: 0x0000000310000000
reslen0: 0x0000000001000000
resadr1: 0x0000000000000000
reslen1: 0x0000000000000000
resadr2: 0x0000000000000000
reslen2: 0x0000000000000000
resadr3: 0x0000000000000000
reslen3: 0x0000000000000000
resadr4: 0x0000000000000000
reslen4: 0x0000000000000000
resadr5: 0x0000000000000000
reslen5: 0x0000000000000000
resirq0: 0x0000000000000000
resirq1: 0x0000000000000000
resirq2: 0x0000000000000000
resirq3: 0x0000000000000000
resirq4: 0x0000000000000000
resirq5: 0x0000000000000000
```

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 %s¥n",strerror(errno));
            exit(0);
      }
      if(ioctl(fd,IOCTL_EXTMEM_GET_DRIVER_INFO,&resinfo)<0)
      {
                  fprintf(stderr,"extmem infomation fail %s¥n",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
```

インストール

extmem は、通常 /usr/local/CNC/drivers/extmem_driver ディレクトリ下に 提供されます。
使用に当たっては、下記の手順でシステムに併せてコンフィグレーションする必要があります。

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ID
EXTMEM_DIDn=デバイス ID
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
3 以上:init 時に強制的に IRQ をこの番号に割り当てる
EXTMEM_DMA_VIDn=DMA コントローラベンダーID(デフォルト:0x114a)
EXTMEM_DMA_DIDn=DMA コントローラデバイス ID(デフォルト:0x5565)
EXTMEM_DMA_SIDn=DMA コントローラサブシステム ID(デフォルト:0x965610B5)
例： insmod extmem.o EXTMEM_VID0=0x1221 EXTMEM_DID0=0x9152 EXTMEM_INT0=1

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、10 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。

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。

```
mknod -mrw /dev/extmem0 c      major      0
mknod -mrw /dev/extmem1 c      major      1
mknod -mrw /dev/extmem2 c      major      2
mknod -mrw /dev/extmem3 c      major      3
mknod -mrw /dev/extmem4 c      major      4
mknod -mrw /dev/extmem5 c      major      5
mknod -mrw /dev/extmem6 c      major      6
mknod -mrw /dev/extmem7 c      major      7
mknod -mrw /dev/extmem8 c      major      8
mknod -mrw /dev/extmem9 c      major      9
```

3) コンフィグレーションシェルスクリプト

extmem に合わせて、ボードサポートパッケージが提供される場合には、"config_パッケージ名.sh"と、"config_パッケージ名.fix.sh"が提供されています。

"config_パッケージ名.fix.sh"使用方法

下記に示すように"cat /proc/extmem"を行うと、BUSSLOT=に続けて、バススロットが表示されます。（この表記は、"lspci"に合わせていますので、小文字の 16 進数表記です）

```
# cat /proc/extmem
version: 8.2F
built  : Nov  6 2015, 09:10:47
boards : 2
license: Development
extmem0: VID=0x114A DID=0x5565 SHARED=0 BUSSLOT=02:00.0
extmem1: VID=0x114A DID=0x5565 SHARED=0 BUSSLOT=24:0c.0
extmem2: VID=0x0000 DID=0x0000 SHARED=0 BUSSLOT=00:00.0
:
:
```

以下のように、"02:00.0" や"24:0c.0"がバススロット番号です。

```
# lspci -d 114A:5565
02:00.0 Network controller: VMIC GE-IP PCI5565,PMC5565 Reflective Memory Node (rev 01)
24:0c.0 Network controller: VMIC GE-IP PCI5565,PMC5565 Reflective Memory Node
```

ここで、従来のコンフィグレーションシェルを動作させ、

```
# ./config_pci5565.sh
Loading module extmem ...
mknod -m=rw /dev/extmem0 c 243 0
mknod -m=rw /dev/extmem1 c 243 1
```

続けて、新規フィックスシェルを呼び出します。

(注意：このシェルには、extmem_start が組み込まれていません。)

```
# ./config_pci5565.fix.sh
ln -s /dev/extmem0 /dev/pci5565a
ln -s /dev/extmem1 /dev/pci5565b
# ll /dev/pci5565?
```

```
lrwxrwxrwx 1 root root 12 11 月  6 09:11 2015 /dev/pci5565a -> /dev/extmem0
lrwxrwxrwx 1 root root 12 11 月  6 09:11 2015 /dev/pci5565b -> /dev/extmem1
```

この様に、ディフォルトでは、該当ボードの(従来と同じ)検出順にデバイス名称を命名します。スロットを固定するためには、フィックスシェルのバススロット部分を変更します。

```
変更前
EXTMEM0=( "/dev/extmem0" "/dev/pci5565a" "" );
EXTMEM1=( "/dev/extmem1" "/dev/pci5565b" "" );
EXTMEM2=( "/dev/extmem2" "/dev/pci5565c" "" );
:

変更後(ボードの順序を入れ替える設定)
EXTMEM0=( "/dev/extmem0" "/dev/pci5565a" "24:0c.0" );
EXTMEM1=( "/dev/extmem1" "/dev/pci5565b" "02:00.0" );
EXTMEM2=( "/dev/extmem2" "/dev/pci5565c" "" );
:
```

再度、フィックスシェルを動作させるとボード名が入れ替わります。

```
# ./config_pci5565.fix.sh
ln -s /dev/extmem0 /dev/pci5565b
ln -s /dev/extmem1 /dev/pci5565a
# ll /dev/pci5565?
lrwxrwxrwx 1 root root 12 11 月  6 09:14 2015 /dev/pci5565a -> /dev/extmem1
lrwxrwxrwx 1 root root 12 11 月  6 09:14 2015 /dev/pci5565b -> /dev/extmem0
```

このシェルでは、どの EXTMEM フィールドでも記述できますので、異なる種類のボード構成の場合では、従来の /etc/rc.local の記述方法に加えて、フィックスシェルを連続に呼び出すことで、バススロ>ットを固定できます。

(フィックスシェルは個別に記述してください。)

例題

以下の例題では、Makefile の define の部分で、３種のモデルをプログラムしています。

linux 標準非同期通知(SIGIO signal)モデル

```
USRDEF = -DIOSIGNAL
```

single process AST model

```
USRDEF = -DAST
```

multi thread AST model

```
USRDEF = -DASTHANDLER
```

-Makefile-

```
LNKLIB = -lextmem -lpthread -lccur_rt -lm
USRDEF = -DIOSIGNAL
#USRDEF = -DAST
#USRDEF = -DASTHANDLER
CMPCMD = cc -O -c $(CCOPT) $(USRDEF) $(SETSMP) -I .
LIBCMD = ar ruv
LNKCMD = cc $(CCOPT)

#
MAIN        = sample
EXTLIB          =
LNKOBJ          = $(MAIN).o
TEMPFILES = core core.*
CFLAGS          = -O $(USRDEF) -D__USE_POSIX -D__USE_POSIX199309
LDFLAGS         =

$(MAIN):    $(LNKOBJ) $(XOXLIB)
            $(LNKCMD) $(LDFLAGS) -o $(MAIN)      $(LNKOBJ) $(LNKLIB) $(EXTLIB)

            $(CMPCMD) $(CFLAGS) $<

$(LNKOBJ):     $(SYSINC) Makefile

clean:
            -rm -f $(TEMPFILES) $(LNKOBJ)
```

-sample.c-

```
#include <stdio.h>
```

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

#include <time.h>
#include <sys/mman.h>
#include <unistd.h>
#include <sys/io.h>
#include <signal.h>
#include <fcntl.h>
//#include <asm/system.h>
#include <pthread.h>

#define PCIBAR0        0

int
do_input(int fd,int reg[])
{
      static extmem_busw_command_t      prog[2];
      static extmem_buswindow_t         wcmd;
      int pc=0,ret;

      prog[pc].window=PCIBAR0;
      prog[pc].operation=MODE_LONG |MODE_R;
      prog[pc].offset=0x00;
      prog[pc].data_ptr=&reg[0];
      pc++;

      prog[pc].window=PCIBAR0;
      prog[pc].operation=MODE_LONG |MODE_R;
      prog[pc].offset=0x1c;
      prog[pc].data_ptr=&reg[1];
      pc++;

      wcmd.count = pc;
      wcmd.busw_command = prog;
      ret = ioctl(fd,IOCTL_EXTMEM_BUSWINDOW, &wcmd );
      return(ret);
}

int
do_output(int fd,int reg[])
{
      static extmem_busw_command_t      prog[1];
      static extmem_buswindow_t              wcmd;
      int pc=0,ret;

      prog[pc].window=PCIBAR0;
      prog[pc].operation=MODE_LONG |MODE_W;
      prog[pc].offset=0x00;
      prog[pc].data_ptr=&reg[0];
      pc++;

      wcmd.count = pc;
      wcmd.busw_command = prog;
      ret = ioctl(fd,IOCTL_EXTMEM_BUSWINDOW, &wcmd );
      return(ret);
}

extmem_driver_info_t     info;
int fd,user;
extern int errno;

#if defined(IOSIGNAL)
static void
```

```
sigio_handler (int sig, siginfo_t *info , void *ptr)
{
      unsigned int status;
      static extmem_busw_command_t      prog[1];
      static extmem_buswindow_t         wcmd;
      int pc=0,ret;
      extmem_int_info_t intr={0,0};

      if(user)
      {
            prog[pc].window=PCIBAR0;
            prog[pc].operation=MODE_LONG |MODE_R;
            prog[pc].offset=0x10;
            prog[pc].data_ptr=&status;
            pc++;
            wcmd.count = pc;
            wcmd.busw_command = prog;
            ioctl(fd,IOCTL_EXTMEM_BUSWINDOW, &wcmd );
      }
      else
      {
            mb();
            status = inl(info.region[PCIBAR0].address+0x10);
      }
      ioctl(fd,IOCTL_EXTMEM_REQ_INT_STATUS, &intr );
      printf("¥nSIGNAL %08x:number of interrupt %d¥n",status,intr.irq);
      return;
}
#endif

#ifdef ASTHANDLER
static pthread_cond_t Condition;
static pthread_mutex_t Mutex;
static sig_atomic_t Event;

static void *
astio_handler (void *arg)
{
      unsigned int status,st;
      static extmem_busw_command_t      prog[1];
      static extmem_buswindow_t              wcmd;
      int pc=0,priv;
      extmem_ast_t ast={0,0};

      pthread_mutex_lock(&Mutex); /* lock */
      priv = iopl(3);
      for(;;)
      {
            ast.timeout = 1000000;      /* 1sec */
            pthread_mutex_unlock(&Mutex); /* lock */
            do
            {
                 st = ioctl(fd,IOCTL_EXTMEM_PAUAST, &ast );
            } while (st!=0);
            pthread_mutex_lock(&Mutex); /* lock */
            if(priv)
            {
                 prog[pc].window=PCIBAR0;
                 prog[pc].operation=MODE_LONG |MODE_R;
                 prog[pc].offset=0x10;
                 prog[pc].data_ptr=&status;
                 pc++;
                 wcmd.count = pc;
                 wcmd.busw_command = prog;
                 ioctl(fd,IOCTL_EXTMEM_BUSWINDOW, &wcmd );
            }
            else
```

```
            {
                //mb();
                status = inl(info.region[PCIBAR0].address+0x10);
            }
            printf("¥nAST HANDLER %8d(%8d): status %08x¥n",ast.pending,ast.timeout,status);
      }
      pthread_mutex_unlock(&Mutex); /* lock */
}

static pthread_t
setup_ast_handler(void *(*thread_main)(void *arg))
{
         static int      thread_status,thread_arg;
      static pthread_t           tid;
      static pthread_attr_t   attributes;
      void *status_p;

      if (pthread_cond_init(&Condition,0)<0)
      {
            fprintf(stderr, "Cannot create condition - %s¥n",strerror(errno));
            return((pthread_t)-1);
      }
      if (pthread_mutex_init(&Mutex,0)<0)
      {
            fprintf(stderr, "Cannot create condition - %s¥n",strerror(errno));
            return((pthread_t)-2);
      }

      if (pthread_attr_init(&attributes))
      {
            fprintf(stderr, "Cannot set attributes - %s¥n", strerror(errno));
            return((pthread_t)-3);
      }
      if (pthread_attr_setscope(&attributes,PTHREAD_SCOPE_SYSTEM))
      {
            fprintf(stderr, "Cannot set attributes - %s¥n", strerror(errno));
            return((pthread_t)-4);
      }
      if (pthread_attr_setdetachstate(&attributes,PTHREAD_CREATE_JOINABLE))
      {
            fprintf(stderr, "Cannot set attributes - %s¥n", strerror(errno));
            return((pthread_t)-5);
      }
      if (pthread_attr_setinheritsched(&attributes,PTHREAD_INHERIT_SCHED))
      {
            fprintf(stderr, "Cannot set attributes - %s¥n", strerror(errno));
            return((pthread_t)-6);
      }
      pthread_mutex_lock(&Mutex); /* lock */
      Event=0;
      if(pthread_create(&tid,&attributes,thread_main,(void *)&thread_arg))
      {
            fprintf(stderr, "Cannot create thread - %s¥n", strerror(errno));
            return((pthread_t)-7);
      }
      pthread_mutex_unlock(&Mutex); /* unlock */
      return(tid);
}
#endif

int
main()
{
      int i,flags;
      void *ptr;
      extmem *dev;
      int bar,user=0;
```

```
	unsigned int data[4],status[2];
	struct sigaction io_act;
#ifdef IOSIGNAL
	extmem_int_info_t int_enable={0,0};
#endif
#if defined(ASTHANDLER)||defined(AST)
	extmem_ast_t ast_enable={0,0};
	static pthread_t           tid;
#endif
#if defined(AST)
	extmem_ast_t ast={0,0};
#endif
	struct timespec w100={0,100000000};/* 100ms */

	fd = open("/dev/extmem0",O_RDWR);
	if (fd<0)
	{
		fprintf(stderr,"open fail¥n");
		exit(0);
	}
	if(ioctl(fd,IOCTL_EXTMEM_GET_DRIVER_INFO,&info)<0)
	{
		fprintf(stderr,"ioctl fail¥n");
		exit(0);
	}
	for(bar=0;bar<PCI_MAX_NUM_REGS;bar++)
	{
		switch(info.region[bar].type)
		{
			case 0:
			break;
			case 1:
				printf("PCIBAR %d I/O Region addr 0x%08x offset 0x%08x %10d bytes¥n",
					bar,
					info.region[bar].address,
					info.region[bar].offset,
					info.region[bar].size);
			break;
			case 2:
				printf("PCIBAR %d MEM Region addr 0x%08x offset 0x%08x %10d bytes¥n",
					bar,
					info.region[bar].address,
					info.region[bar].offset,
					info.region[bar].size);
			break;
		}
	}
#if ASTHANDLER
	printf("This is AST SAMPLE PROGRAM(call back model)¥n");
#endif
#ifdef AST
	printf("This is AST SAMPLE PROGRAM(single process model)¥n");
#endif
#ifdef IOSIGNAL
	printf("This is SIGNAL SAMPLE PROGRAM¥n");
#endif

#ifdef IOSIGNAL
	if (fcntl (fd, F_SETOWN, getpid()) != 0)
	{
		fprintf (stderr,"Error : Process ID = %d ¥n", getpid());
	}
	/* Register signal */
	flags = fcntl (fd, F_GETFL);
	if ( flags == -1)
	{
		fprintf (stderr,"Error : flags¥n");
```

```
      }
      if ( fcntl(fd, F_SETFL, flags | FASYNC) == -1 )
      {
            fprintf (stderr,"Error : flags .....¥n");
      }

      sigemptyset (&io_act.sa_mask);
      io_act.sa_sigaction = sigio_handler;
      io_act.sa_flags = SA_SIGINFO|SA_RESTART;
         if ( sigaction ( SIGIO, &io_act, NULL ) != 0 )
      {
            fprintf (stderr,"Error : sigaction failed¥n");
      }
      int_enable.init = 1;
      ioctl(fd,IOCTL_EXTMEM_REQ_INT_NOTIFY,&int_enable);
#endif
#if defined(ASTHANDLER)||defined(AST)
      ast_enable.timeout = 1;
      ioctl(fd,IOCTL_EXTMEM_DCLAST,&ast_enable);
#endif
#if defined(ASTHANDLER)
      tid = setup_ast_handler(astio_handler);
#endif
      if (info.region[bar].type==2)
      {
            ptr = mmap(0,info.region[PCIBAR0].size,PROT_READ|PROT_WRITE,MAP_SHARED,fd,0x1000 * PCIBAR0);
            if (ptr==MAP_FAILED)
            {
                  fprintf(stderr,"mmap fail¥n");
                  exit(0);
            }
            dev=(extmem *)ptr;
            munmap(ptr,info.region[PCIBAR0].size);
      }
      else
      {
            user = iopl(3);
            if(user)
            { /* not root */
                  do_input(fd,data);
                  do_output(fd,data);
            }
            else
            { /* super user */
                  status[0] = inl(info.region[PCIBAR0].address+0x10);
                  status[1] = inl(info.region[PCIBAR0].address+0x14);
                  outb(0xff,info.region[PCIBAR0].address+0x9);
#if defined(IOSIGNAL) || defined(AST) ||defined(ASTHANDLER)
                  outl(0xfffffffe,info.region[PCIBAR0].address+0x10);
                  outl(0xffffffff,info.region[PCIBAR0].address+0x14);
                  status[0] = inl(info.region[PCIBAR0].address+0x10);
                  status[1] = inl(info.region[PCIBAR0].address+0x14);
                  printf("%08x %08x¥n¥n",status[0],status[1]);
#else
                  outl(0xffffffff,info.region[PCIBAR0].address+0x10);
#endif
                  for(;;)
                  {
#if defined(AST)
                        int st,status;
                        static extmem_busw_command_t      prog[1];
                        static extmem_buswindow_t              wcmd;

                        ast.timeout = 100000;       /* 100 msec */
                        st = ioctl(fd,IOCTL_EXTMEM_PAUAST, &ast );
                        if(st==0)
                        {
```

```
                                    if(user)
                                    {
                                     prog[0].window=PCIBAR0;
                                     prog[0].operation=MODE_LONG |MODE_R;
                                     prog[0].offset=0x10;
                                     prog[0].data_ptr=&status;
                                     wcmd.count = 1;
                                     wcmd.busw_command = prog;
                                     ioctl(fd,IOCTL_EXTMEM_BUSWINDOW, &wcmd );
                                    }
                                    else
                                    {
                                     //mb();
                                     status = inl(info.region[PCIBAR0].address+0x10);
                                    }
                                    printf("¥nAST %8d(%8d): status %08x¥n",ast.pending,ast.timeout,status);
                                }
        #else
                                nanosleep(&w100,NULL);
        #endif
        #ifdef ASTHANDLER
                                pthread_mutex_lock(&Mutex); /* lock */
        #endif
                                //mb();
                                data[0] = inl(info.region[PCIBAR0].address+0x00);
                                data[1] = inl(info.region[PCIBAR0].address+0x04);
                                data[2] = inl(info.region[PCIBAR0].address+0x1c);
                                outl(data[0],info.region[PCIBAR0].address+0x04);
                                printf(" %08x %08x %08x¥r",data[0],data[1],data[2]);fflush(stdout);
        #ifdef ASTHANDLER
                                pthread_mutex_unlock(&Mutex); /* lock */
        #endif
                        }
                    }
                    outl(0xffffffff,info.region[PCIBAR0].address+0x10);
                    iopl(0);
              }
        #ifdef IOSIGNAL
              int_enable.init = 0;
              ioctl(fd,IOCTL_EXTMEM_REQ_INT_NOTIFY,&int_enable);
        #endif
        #if defined(ASTHANDLER)||defined(AST)
              ast_enable.timeout = 0;
              ioctl(fd,IOCTL_EXTMEM_DCLAST,&ast_enable);
        #endif
        #ifdef ASTHANDLER
              pthread_join(tid,NULL);
        #endif
              if (info.region[bar].type==2)
              close(fd);
        }
```

書式

```
#include <sys/libextmem.h>
```

機能解説

a) misc ライブラリ

リアルタイム用の汎用ライブラリです。

1. 時間計測用関数

リアルタイムカウンタの値を読み込む

```
void readtime(struct timespec *nowtime);
```

２つのリアルタイムカウンタの差をマイクロ秒に変換する

```
void adjust(struct timespec *start,
            struct timespec *end,
```

```
                float *realtime);
```

--- ２点間の時間を計測する ---

```
        struct timespec start,end;
        float realtime;
        readtime(&start);
        /*
        この部分の時間を計測する
        */
        readtime(&end);
        adjust(&start,&end,&realtime);
        printf("%f マイクロ秒¥n");
```

2. POSIX インターバルタイマ作成関数

```
        timer_t create_posix_timer
        (
             int signum,
             int sec,
             int nsec
        );
```

戻り値

インターバルタイマ ID

引数

signum　シグナル番号(通常は SIGRTMIN-SIGRTMAX)
sec　　インターバルタイマの秒
nsec　　インターバルタイマのナノ秒

--- 1 ミリ秒のタイマシグナルを作成する ---

```
        #define MAXCOUNT 100
        sigset_t set,oset;
        int signum,status,count;
        timer_t timer_id;
        siginfo_t info;
        if (sigprocmask(0,NULL,&set)==(-1))
        {
            fprintf(stderr, "Cannot get sigprocmask¥n");
            exit(1);
        }
        sigaddset(&set,SIGRTMIN);
        if (sigprocmask(SIG_SETMASK,&set,&oset)==(-1))
        {
             fprintf(stderr, "%s: Cannot set sigprocmask¥n");
             exit(1);
        }
        signum = SIGRTMIN;
        timer_id = create_posix_timer(signum,0,1000000);
        if (timer_id<0)
        {
              fprintf(stderr, "%s: Cannot create posix timer¥n");
              exit(1);
        }
        do
        {
                 status = sigwaitinfo(&set,&info);
                 count++;
        } while (count<MAXCOUNT);
        timer_delete(timer_id);
```

3. 優先度設定関数

呼出したプロセスあるいはスレッドの優先度を設定する

```
        int set_priority
        (
             int id,
```

```
        int class,
        int priority
  );
```

戻り値

エラーなら-1 成功なら 0

引数

id

プロセスなら MPA_PID, スレッドなら MPA_TID

class,priority

class 0 の場合には nice スケジューリング
priority は-20 から+20 を設定する
quantum は、以下の通り

```
-20 => 199ms
-19 => 194ms
-18 => 189ms
-17 => 184ms
-16 => 184ms
-15 => 174ms
-14 => 169ms
-13 => 164ms
-12 => 164ms
-11 => 155ms
-10 => 150ms
 -9 => 145ms
 -8 => 145ms
 -7 => 135ms
 -6 => 130ms
 -5 => 125ms
 -4 => 125ms
 -3 => 116ms
 -2 => 111ms
 -1 => 106ms
  0 => 106ms
  1 =>  96ms
  2 =>  91ms
  3 =>  86ms
  4 =>  86ms
  5 =>  77ms
  6 =>  72ms
  7 =>  67ms
  8 =>  67ms
  9 =>  57ms
 10 =>  52ms
 11 =>  47ms
 12 =>  47ms
 13 =>  38ms
 14 =>  33ms
 15 =>  28ms
 16 =>  28ms
 17 =>  18ms
 18 =>  13ms
 19 =>  13ms
```

class 1 の場合には SCHED_OTHER スケジューリング
priority は 0 のみ
quantum は、nice と同じである

class 2 の場合には SCHED_RR スケジューリング
priority は 1 から 99 を設定する
quantum は、すべて同じ 10.3 ミリ秒である

class 3 の場合には SCHED_FIFO スケジューリング
priority は 1 から 99 を設定する

4. 実行プロセッサ指定関数

呼出したプロセスあるいはスレッドの実行プロセッサを設定する

```
int set_processor
(
      int id,
      int cpu
);
```

戻り値

エラーなら-1 成功なら 0

引数

id

プロセスなら MPA_PID, スレッドなら MPA_TID

cpu

0 から始まる CPU の番号

名前

x86_memcpy - メモリ領域をコピーする。

書式

```
#include <extmem.h>

void *x86_memcpy(void *dest, const void *src, size_t n);
```

説明

x86_memcpy() はメモリ領域 src の先頭 n バイトを メモリ領域 dest にコピーする。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。 (only x86_64)

返り値

x86_memcpy()() は dest へのポインタを返す。 このコードは、以下のように x86 用にアセンブラで記述されている。

```
void *x86_memcpy(void *to, const void *from, size_t n)
{
         int d0, d1, d2;
         asm volatile("rep ; movsl¥n¥t"
         "movl $4,%%ecx¥n¥t"
         "andl $3,%%ecx¥n¥t"
         "jz 1f¥n¥t"
         "rep ; movsb¥n¥t"
         "1:"
         : "=&c" (d0), "=&D" (d1), "=&S" (d2)
         : "0" (n / 4), "g" (n), "1" ((long)to), "2" ((long)from)
         : "memory");
         return to;
}
```

名前

x86_64_memcpy - メモリ領域をコピーする。

書式

```
#include <extmem.h>

void *x86_64_memcpy(void *dest, const void *src, size_t n);
```

説明

x86_memcpy() はメモリ領域 src の先頭 n バイトを メモリ領域 dest にコピーする。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。

返り値

x86_64_memcpy()() は dest へのポインタを返す。

名前

x86_64_memset - メモリ領域に値をセットする。

書式

```
#include <extmem.h>

void *x86_64_memset(void *dest, int c"src, size_t n);
```

説明

x86_memset() はメモリ領域 src の先頭 n バイトに 値をセットする。

返り値

x86_64_memcpy()() は dest へのポインタを返す。