Special

Linuxで PCI汎用入出力カードを制御する

芳之内弘

本誌2001年3月号の「RTLinuxによるFAシステム構築」では、PCIバス対応汎用入出力カードのコンフィグレーションレジスタの内容を取得する方法を説明しました。これは、コンフィグレーションレジスタに直接アクセスして情報を読みとる方法でした。この方法はユーザーモードでもカーネルモードでも使うことはできますが、RTLinuxを使う場合は、通常、I/O空間へのアクセスはカーネルモードで実行します。また、最近のカーネルのバージョン(2.2以降)では、このコンフィグレーションレジスタからPCIカードの情報を直接取得することはしないように勧めています(lunux/Documentation/pci.txtによる)。これはカーネルがリマップしているかもしれないからです。

そこで今回は直接コンフィグレーションレジスタをアクセスせずにPCIカードの情報を取得する方法を説明します。また、PCIの情報を取得するツールについても説明します。そして最後にこの情報に基づいた実際の割り込みルーチンを紹介します。

PCIカードを制御するには、PCIバスの基礎を知っていた方が何かと便利ですので、その基礎を簡単に説明します。詳しくは記事末の参考文献を読んでください。

PCIバス

PCIバスの簡単な構造については、図1を参考にしてください。CPUからPCIブリッジを経てPCIバスがあります。通常はこれがPCIバス0になります。このPCIバスをさらに拡張するために、PCI−PCIブリッジが存在することがあります。このバスはPCIバス1になります。PCIバス0にISAバスを拡張するときは、PCI−ISAブリッジを用います。PCIカードはPCIバス0およびPCIバス1に接続されます。

PCI−PCIブリッジを含むすべてのPCIデバイスにはコンフィグレーションレジスタと呼ばれるレジスタ群が実装されています。このレジスタには、PCIデバイスをアクセスするために必要な情報が書き込まれます。このレジスタにアクセスするための空間はコンフィグレーション空間と呼ばれ、メモリアドレス、I/Oアドレス空間とは区別されます。このレジスタの内容は、PCの立ち上げ時に、システムのプラグアンドプレイ機能により設定されます。

それでは、PCIカードに適用されるコンフィグレーションレジスタについて説明します(表1)。

PCIカードの実装をサーチするときは、このコンフィグレーションレジスタのベンダIDとデバイスIDをもとにしてサーチします。そして、この実装されているデバイスのベースアドレスと割り込みラインの値を読み出します。PCIカードの制御

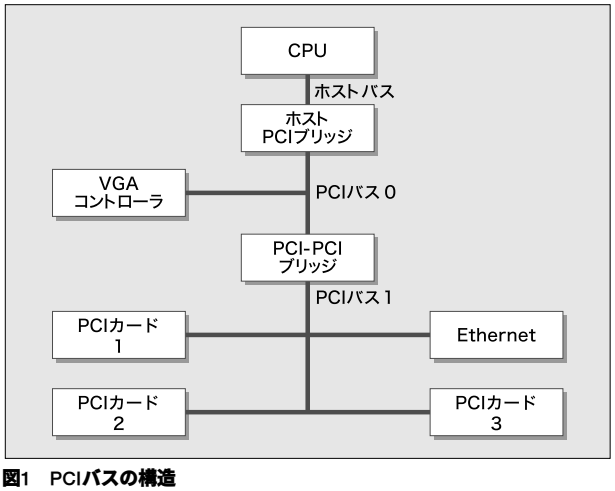

図1　PCIバスの構造

表1　PCIカードのコンフィグレーションレジスタ概要

アドレス	機能
0x00-0x01	ベンダID(R)
0x02-0x03	デバイスID(R)
0x04-0x05	デバイス制御(R/W)
0x06-0x07	デバイスステータス(R/W)
0x08	バージョンID(R)
0x09-0x0b	クラスコード(R)
0x10-0x13	ベースアドレスレジスタ0(R/W)
0x21-0x23	ベースアドレスレジスタ5(R/W)
0x3c	インタラプトライン(R/W)
0x3d	ＰＣＩインタラプトピン(R)

R：読み出し可能
W：書き込み可能
R/W：読みだし、書き込み可能
レジスタの内容がハード(ROMを含む)で固定されているデータは読みだしのみ可能です。

はこの値をもとにして実行します。

LinuxのPCI関数

このPCIデバイスをサーチするには、pci_find_device関数を使います。この関数はlinux/devices/pci/pci.cの中で定義されています。実際にはリスト1のようにして使います。devはVENDOR_IDとDEVICE_IDが一致したpci_dev構造体のポインタです。従って、この関数を実行してもコンフィグレーションレジスタはアクセスされません。カーネルが読み込んでセットしたpci_dev構造体のポインタをリターンするだけです。このpci_dev構造体はlinux/include/linux/pci.hの中で定義されています。その主なメンバは次の通りです(表2)。必要な情報はこの構造体にセットされていますので、このdevポインタを使ってデータを読み出します。この構造体は実装されているPCIデバイスの数だけ作られ、連続する構造体は*nextでリンクされています。このリンクをたどることで目的のデバイスを探すことができます。

PCIツール

LinuxにはPCIをサポートするツールとして「pciutils」があります。これはlspciとsetpciのコマンドツールです。ディストリビューションに用意されていないときは記事末のRESOURCE[1]よりダウンロードしてインストールしてください。

lspciを実行すると、PCIデバイスの情報が得られます。setpciを使う場合は注意が必要ですので安易に使わないようにしましょう。次にlspciの実行例を示します(実行例1)。ここでは、出力される内容のうち、先月の記事で使用したコンテックのPCIカード2

表2　pci_dev構造体

メンバ名		説明
struct pci_dev	*next	次の構造体へのポインタ
unsigned short	pci_dev->vendor	ベンダID
unsinged short	pci_dev->device	デバイスID
unsigned int	pci_dev->irq	割り込みライン
unsigned long	pci_dev->base_address[0]	ベースアドレス0

枚のみについて示しています。1行目の最初の「00」はバス番号、次の「09」はデバイス番号です。最後の「8122」がデバイスIDです。デバイスが「unkonwn device」で表示されるのは、デバイスIDの「8122」が登録されていないからです。ベンダIDの情報を登録しているファイルを修正すれば、この数字の代わりにデバイス名が正確に表示されます。

この場合は、/usr/share/pci.idsのContecに関する部分を次のように修正します(リスト2)。段落はタブです。Contecの項がない場合は追加します。追加する場合は、ベンダIDの昇順に登録します。pci.idsを修正した後でもう1度lspciを実行します(実行例2)。これでデバイス名が表示されました。

また、「lspci -v」を実行した場合には、実行例3のように

リスト1　pci_find_device関数

```
struct pci_dev *dev = NULL;
dev = pci_find_device(VENDOR_ID, DEVICE_ID, dev))
```

リスト2　pci.idsの修正

```
1221  Contec Co., Ltd
      8122  PIO-32/32L
            1221 8122 Input Output Board
      8105  CNT(4)
            1221 8105 24bit four Counters
```

実行例1　lspciの実行結果

```
# lspci
00:09.0 Multimedia controller: Contec Co., Ltd: Unknown device 8122
00:0e.0 Multimedia controller: Contec Co., Ltd: Unknown device 8105
```

実行例2　pci.ids修正後のlspci実行結果

```
# lspci
00:09.0 Multimedia controller: Contec Co., Ltd PIO-32/32L
00:0e.0 Multimedia controller: Contec Co., Ltd CNT(4)
```

実行例3　「lspci -v」の実行結果

```
00:09.0 Multimedia controller: Contec Co., Ltd PIO-32/32L
      Subsystem: Contec Co., Ltd Input Output Board
      Flags: medium devsel, IRQ 5
      I/O ports at d000

00:0e.0 Multimedia controller: Contec Co., Ltd CNT(4)
      Subsystem: Contec Co., Ltd 24bit four Counters
      Flags: medium devsel, IRQ 10
      I/O ports at b000
```

なります(これもContecの情報だけを示します)。-vオプションではベースアドレスとIRQ番号が得られます。「lspci -vv」を実行すればさらに詳しい情報が得られます(実行例4)。「cat /proc/pci」でも情報は得られます。その内容を次に示します(実行例5)。

PCIカードのアクセス

コンテックのPCIバス対応汎用入出力カード「PIO-32/32L」を使った、入出力と割り込みの実例です(リスト3、ソースコードは付録CD-ROMに収録してあります)。Linuxカーネル2.2.14にRTLinux 2.2のパッチを当てたもので動作を確認しています。以下を順に実行してください。

- 同じディレクトリにrtl.mk、Makefile、sample1.cを置いて、makeを実行してコンパイルします。このrtl.mkはRTLinuxのバージョンやインストールディレクトリでパスが変わりますので注意してくだい。
- rtlinux-2.2のディレクリでinsrtlを実行して次のモジュールをインストールします。

```
modules/rtl_time.o
modules/rtl_sched.o
modules/rtl_posixio.o
modules/rtl_fifo.o
```

- insmod sample1を実行してsample1.oモジュールをロードします。
- rmmod sample1でsample1.o はアンイストールされます。

sample1をロードすると、0.5ms(ミリ秒)ごとにtask1スレッドが起動されます。このタスクが起動されると、ポート4のビット1を交互にオン、オフします。またさらにポート4のビット0をオンします。ポート0のビット0がオンになると割り込みが発生するようにプログラムしてあるので、ポート4のビット0の出力端子をポート0のビット0の入力端子に接続しておくと、出力ポートのビット0がオンになると同時に割り込みが発生します。そして、割り込みハンドラでは ポート4のビット0をオフにします。

従って、0.5msごとに割り込みが周期的に発生します。オシロスコープを接続するとその様子が良く理解できます。オシロスコープの写真を掲載したかったのですが、うまく撮影できませんでしたので、図2を参照してください。ビット1のオンとオフの間隔が異なっていますが、これは実際にポートにオフを書き込んでから、ポート出力がオフになるまでにおよそ80μs(マイクロ秒)遅れているからです。

I/Oカードの遅れ(スペックでは1msの応答時間)がありますので、ビット0をオンにしてから95μsから105μs後に割り込みが発生しています。この遅れはその大部分がボードの遅れ時間です。

このボードの遅延時間を差し引くと、割込みハンドラの反応時間はおよそ5μsから15μsになります。正確に割り込みハンドラの反応遅延を測定するには、もっと高速のI/Oカードを用いる必要があります。

実行例4 「lspci -vv」の出力結果

```
00:09.0 Multimedia controller: Contec Co., Ltd PIO-32/32L
      Subsystem: Contec Co., Ltd Input Output Board
      Control: I/O+ Mem- BusMaster- SpecCycle- MemWINV-
               VGASnoop- ParErr- Stepping- SERR- FastB2B-
      Status: Cap- 66Mhz- UDF- FastB2B- ParErr-
               DEVSEL=medium >TAbort- <TAbort- <MAbort- >SERR- <PERR-
      Interrupt: pin A routed to IRQ 5
      Region 0: I/O ports at d000

00:0e.0 Multimedia controller: Contec Co., Ltd CNT(4)
      Subsystem: Contec Co., Ltd 24bit four Counters
      Control: I/O+ Mem- BusMaster- SpecCycle- MemWINV-
               VGASnoop- ParErr- Stepping- SERR- FastB2B-
      Status: Cap- 66Mhz- UDF- FastB2B- ParErr-
               DEVSEL=medium >TAbort- <TAbort- <MAbort- >SERR- <PERR-
      Interrupt: pin A routed to IRQ 10
      Region 0: I/O ports at b000
```

実行例5 「cat /proc/pci」の出力結果

```
  Bus  0, device   9, function  0:
  Multimedia controller: Unknown vendor Unknown device (rev 0).
    Vendor id=1221. Device id=8122.
    Medium devsel.  IRQ 5.
    I/O at 0xd000 [0xd001].

  Bus  0, device  14, function  0:
  Multimedia controller: Unknown vendor Unknown device (rev 0).
     Vendor id=1221. Device id=8105.
     Medium devsel.  IRQ 10.
     I/O at 0xb000 [0xb001].
```

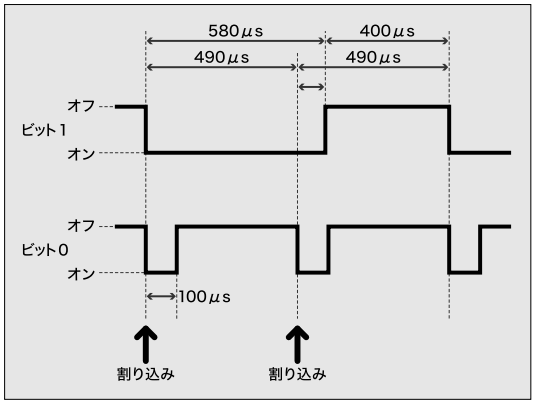

図2 出力ポートの波形

リスト3 sample1.c

```
/*********************************************************
 *  RT-Linux module for PCI-board                        *
 *     2001/01/14                                        *
 *********************************************************/

#include <rtl_core.h>
#include <rtl_time.h>
#include <rtl.h>
#include <rtl_sched.h>

#include  <linux/pci.h>
#include  <asm/io.h>


#define  NOPCI -1
#define  PCI_MAX     6

#define     VENDOR      0x1221         // Contec
#define     DEVICE_PIO   0x8122        // PIO-32/32L(PCI)
#define     DEVICE_PO64  0x8142        // PO-64L(PCI)
#define     DEVICE_CNT   0x8105        // CNT-24-4(PCI)

#define TIME_0_5MSEC    500000       //0.5msec

pthread_t  thread1;                    //スレッド

static struct pci_bord_data {
   unsigned int  addr0;
   unsigned char irq;
} pci_tbl[6] ;

static struct pci_dev  *dev = NULL;  //PCIの情報がセットされた構造体

unsigned int addr0,addr1;
unsigned char irq0,irq1;
int      int_count=0,count_task1=0;
int      cnt=0;
char     out_data=0;
/*-----------------------------------------------------
  PCIカードの情報を得る関数
-----------------------------------------------------*/
int get_pci_info( unsigned int vender_id, unsigned int dev_id ){
    int  j;

    printk("get_pci_info Start vender_id=%04x  dev_id=%x \n",
        vender_id,dev_id);

    dev = NULL;
    j = pci_present();
    if ( j != 0 ){
     printk("PCI presented \n");
     dev = pci_find_device(vender_id, dev_id, dev);
     if(dev != NULL){
         printk("Pcidev found \n");
         pci_tbl[cnt].addr0 = PCI_BASE_ADDRESS_IO_MASK &
                              dev->base_address[0];
         pci_tbl[cnt].irq = dev->irq;
         cnt++;
     }
     else
         printk("No device found.  \n");
     return cnt;
    }else{
     printk("NO PCI presented \n");
     return NOPCI;
    }
```

リスト3　続き

```
}

/*************************************************
   RialTime Task task1   0.5ms
   周期的に発生するスレッド
   0.5msごとにポート4のビット1をオン／オフする
              ポート4のビット0をオンする
   ビット0をオンにすると割り込みが発生する。
*************************************************/
void *task1(void *t) {

  while(1) {
      count_task1++;
      if(count_task1==1){
       out_data |= 2;                //ビット1をオン
      }
      if(count_task1==2){
       count_task1 = 0;
       out_data &= 0xfd;             //1111 1101 ビット1をオフ
      }
      out_data |= 1;                 //ビット0をオン
      outb_p(out_data, addr0 +4);

      pthread_wait_np();
  }
}
/*************************************************
  割り込みハンドラ
  ポート4のビット0をオンにするとこのハンドラが起動される。
  このハンドラ内でビット0をオフにする。
  PCIバスの割り込みは共有割り込みなので、割り込みハンドラ
  の最初で割り込みフラグをチェックする必要がある。
  また割り込みはレベル割り込みなので、割り込みの原因になる
  フラグはクリアする必要がある。
*************************************************/
unsigned int intr_handler(unsigned int irq, struct pt_regs *regs) {
    unsigned char cdata;

    cdata = inb(addr0 + 0x11);       //割り込みフラグの読み込み
    if(cdata){
        outb_p(cdata,addr0 + 0x11);  //割り込みフラグクリア

     out_data &= 0xfe;               //1111 1110  ビット0のオフ
     outb_p(out_data, addr0 +4);
    }
    rtl_hard_enable_irq (irq0);
1   return 0;
}
/*************************************************
  モジュールの初期化関数
      スレッドを起動
      PCIカードの情報を取得
      割り込みモードの設定
      割り込みハンドラの起動
   を実行する。
*************************************************/
int init_module( void  ){
    pthread_attr_t      attr;
    struct sched_param  sched_param;
    hrtime_t            now, period;

    int j;

    //-----0.5msごとに発生するスレッドをセットする------
    now = gethrtime();
    period = TIME_0_5MSEC;
```

リスト3　続き

```
    pthread_attr_init(&attr);
    sched_param.sched_priority = 4;
    pthread_attr_setschedparam(&attr, &sched_param);
    pthread_create(&thread1, &attr, task1, (void *)1);
    pthread_make_periodic_np(thread1, now, period);

    //-----PIO-32/32Lカードの情報を得る-------------------------
    j = get_pci_info( VENDOR, DEVICE_PIO );
    printk("pci_tbl PIO:  addr0=%x irq=%d \n",pci_tbl[0].addr0,pci_tbl[0].irq);
    addr0 = pci_tbl[0].addr0;
    irq0 = pci_tbl[0].irq;

    //-----CNT24カードの情報を得る ------------------------------
    dev = NULL;
    j = get_pci_info( VENDOR, DEVICE_CNT );
    printk("pci_tbl CNT:  addr0=%x irq=%d \n",pci_tbl[1].addr0,pci_tbl[1].irq);
    addr1 = pci_tbl[1].addr0;
    irq1 = pci_tbl[1].irq;

    //-----PIOカードの割り込みモードの設定---
    outb_p(0,addr0 + 0x0e);                 //コネクタピン入力で割り込み発生
                                            //ピン入力High -> Lowで割り込み発生
                                            //INT A端子へ出力 -> INT AでPC割り込み
    outb_p(0x0e,addr0 + 0x10);              //割り込みマスクを解除
    outb_p(0x0f,addr0 + 0x11);              //割り込みフラグをクリア

    outb_p(0,addr0+4);                      //ポートをクリアする

    rtl_request_irq (irq0, intr_handler);   //割り込みハンドラの登録
    rtl_hard_enable_irq (irq0);             //起動

    return( 0 );
}
/*************************************
  スレッドと割り込みハンドラの開放
*************************************/
void cleanup_module( void ){
    pthread_delete_np(thread1);
    rtl_free_irq(irq0);

}
/*****************************************************/
```

R E S O U R C E

[1] pciutilsのダウンロードサイト
ftp://sunsite.sut.ac.jp/pub/archives/linux/sunsite-unc/hardware/!INDEX.html

●PCIバスに関する参考文献

・「PCI/CompactPCIバスの概要と応用」
インターフェース1997年3月号／CQ出版社発行

・「PCIデバイス設計入門」
TECH I Vol.3／CQ出版社発行／ISBN 4-7898-3314-3／2200円

・「PCIバスの詳細と応用へのステップ」
OpenDesign No.7／CQ出版社発行／ISBN 4-7898-3530-8／1835円(税込み)

・「PCIバスの基礎と応用」
トランジスタ技術SPECIAL No.65／ISBN 4-7898-3257-0／1840円(税込み)

・「PCIバスによるI/O制御」
大川善邦著／オーム社発行／ISBN 4-274-94608-8／2200円

・JFドキュメント「The Linux Kernel(6. PCI)」
http://www.linux.or.jp/JF/JFdocs/The-Linux-Kernel.html

●PCI基板の取扱説明書

・Conterc社PIO-32/32L(PCI)取扱説明書
http://www2.contec.co.jp/products/download.cgi?HTML=DTL&KATA=PIO-32/32L(PCI)&KATASIKI=PIO-32/32L(PCI)&BUNRUI=0,0,0&SYUBETU=0#3