PHY & LINK チップのレジスタダンッププログラム説明書

PHY チップと LINK チップのレジスタを表示するプログラムの説明を行います．本ボードは，IEEE1394 の LINK LSI を ISA/PCI バスの，0xF080 からマップしてあります．本ボードは IEEE 1394a 準拠の TI TSB12LV32A を用いています．本ボード自体のレジスタは 0xF000 からマップされています．TSB12LV32 のレジスタは 0xF080 から順次マップされます．ただし，Windows 上ではドライバや DLL に I/O アドレスの絶対値は隠蔽されるため，相対的なアドレスを意識するだけで構いません．以降に，プログラムによって，各レジスタの内容をダンプした様子を示します．表示のフォーマットは；

(index,[offset]) = 16 進表示 2 進数表示

となっています．先頭のレジスタ表示をみると，TSB12LV32 のバージョン，0x711538A0 と表示されているのが分かります．

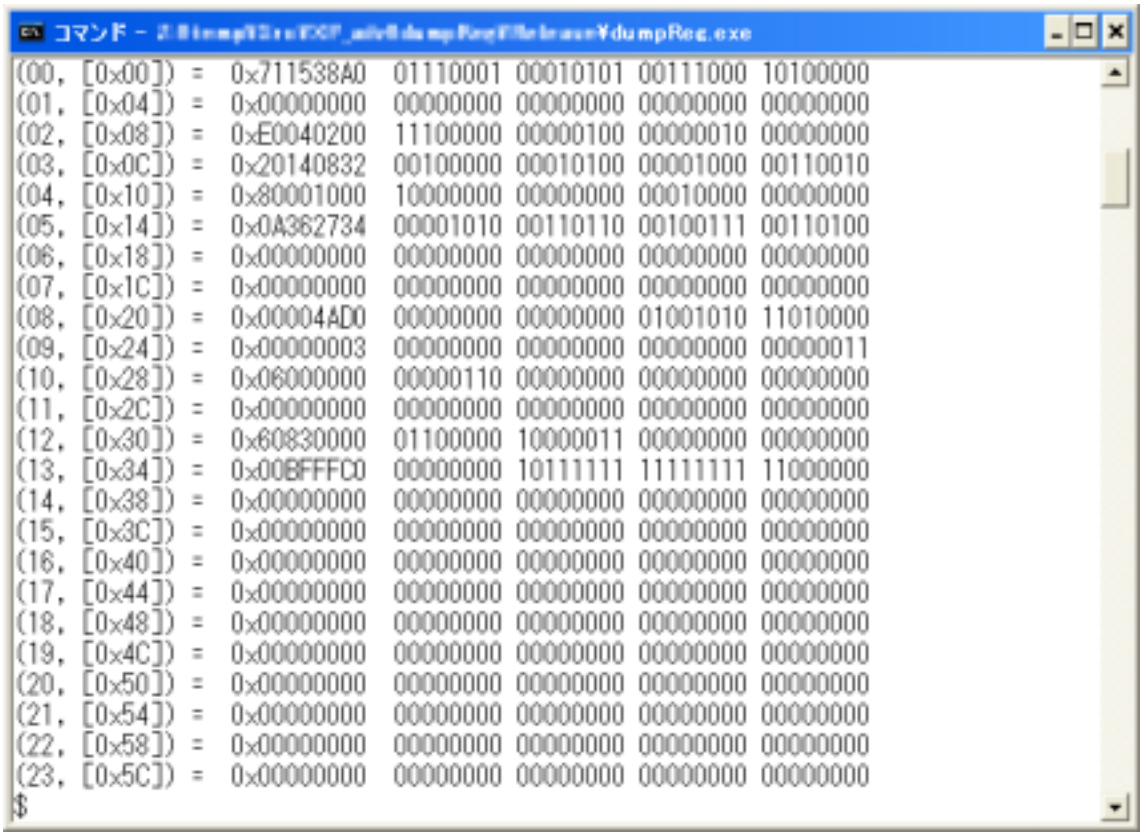


図 LINK レジスタの様子

PHY レジスタは，直接バスに接続されていません．PHY は LINK レジスタ経由でアクセスしなければなりません．LINK レジスタのオフセット 0x24 に Phy Access レジスタがありますので，ここを使ってアクセスします．以降に，PHY レジスタをダンプした様子を示します．

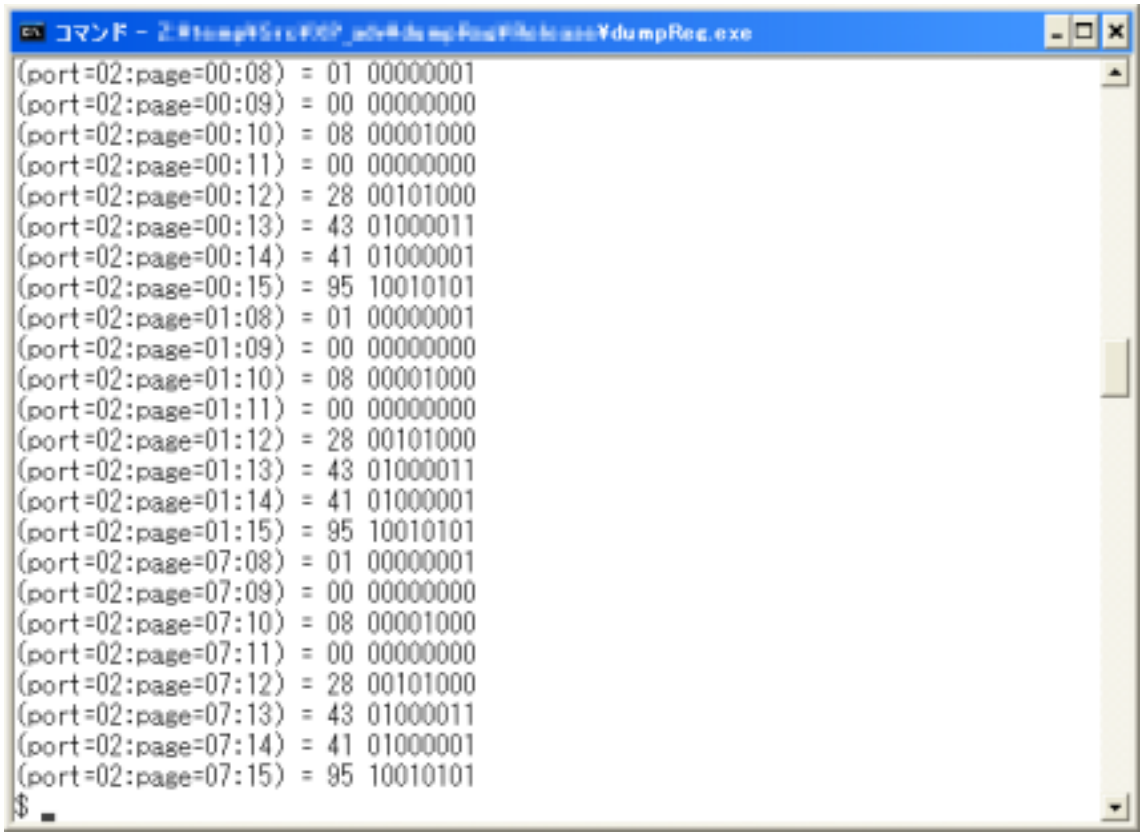


図　PHY レジスタの様子

本ボードでは，IEEE 1394a に対応したためレジスタの数も多くなり，画面ダンプだけでは表示しきれません．以降に，レジスタ内容の一覧を示します．

```
MN1394P-01 register dump Program version 0.9
        (C)Spacesoft corp. 2000-2007, All right reserved.
 ******* Command Summary *******
 DLINK ................. dump LINK registers
 DPHY ................. dump PHY  registers

 Q ..................... Quit
$ dlink
(00, [0x00]) =  0x711538A0  01110001 00010101 00111000 10100000
(01, [0x04]) =  0x00000000  00000000 00000000 00000000 00000000
(02, [0x08]) =  0xE0040200  11100000 00000100 00000010 00000000
(03, [0x0C]) =  0x20140832  00100000 00010100 00001000 00110010
(04, [0x10]) =  0x80001000  10000000 00000000 00010000 00000000
(05, [0x14]) =  0x2976C167  00101001 01110110 11000001 01100111
(06, [0x18]) =  0x00000000  00000000 00000000 00000000 00000000
(07, [0x1C]) =  0x00000000  00000000 00000000 00000000 00000000
(08, [0x20]) =  0x00004AD0  00000000 00000000 01001010 11010000
```

```
(09, [0x24]) =  0x00000003  00000000 00000000 00000000 00000011
(10, [0x28]) =  0x06000000  00000110 00000000 00000000 00000000
(11, [0x2C]) =  0x00000000  00000000 00000000 00000000 00000000
(12, [0x30]) =  0x60830000  01100000 10000011 00000000 00000000
(13, [0x34]) =  0x00BFFFC0  00000000 10111111 11111111 11000000
(14, [0x38]) =  0x00000000  00000000 00000000 00000000 00000000
(15, [0x3C]) =  0x00000000  00000000 00000000 00000000 00000000
(16, [0x40]) =  0x00000000  00000000 00000000 00000000 00000000
(17, [0x44]) =  0x00000000  00000000 00000000 00000000 00000000
(18, [0x48]) =  0x00000000  00000000 00000000 00000000 00000000
(19, [0x4C]) =  0x00000000  00000000 00000000 00000000 00000000
(20, [0x50]) =  0x00000000  00000000 00000000 00000000 00000000
(21, [0x54]) =  0x00000000  00000000 00000000 00000000 00000000
(22, [0x58]) =  0x00000000  00000000 00000000 00000000 00000000
(23, [0x5C]) =  0x00000000  00000000 00000000 00000000 00000000
$ dphy
(00) = 03 00000011
(01) = 3F 00111111
(02) = E3 11100011
(03) = 40 01000000
(04) = 84 10000100
(05) = 10 00010000
(06) = 00 00000000
(07) = 00 00000000
(port=00:page=00:08) = F8 11111000
(port=00:page=00:09) = 00 00000000
(port=00:page=00:10) = 00 00000000
(port=00:page=00:11) = 00 00000000
(port=00:page=00:12) = 00 00000000
(port=00:page=00:13) = 00 00000000
(port=00:page=00:14) = 00 00000000
(port=00:page=00:15) = 00 00000000
(port=00:page=01:08) = F8 11111000
(port=00:page=01:09) = 00 00000000
(port=00:page=01:10) = 00 00000000
(port=00:page=01:11) = 00 00000000
(port=00:page=01:12) = 00 00000000
(port=00:page=01:13) = 00 00000000
(port=00:page=01:14) = 00 00000000
(port=00:page=01:15) = 00 00000000
(port=00:page=07:08) = 00 00000000
(port=00:page=07:09) = 00 00000000
(port=00:page=07:10) = 00 00000000
(port=00:page=07:11) = 00 00000000
(port=00:page=07:12) = 00 00000000
(port=00:page=07:13) = 00 00000000
(port=00:page=07:14) = 00 00000000
(port=00:page=07:15) = 00 00000000
(port=01:page=00:08) = F8 11111000
(port=01:page=00:09) = 00 00000000
(port=01:page=00:10) = 00 00000000
(port=01:page=00:11) = 00 00000000
(port=01:page=00:12) = 00 00000000
(port=01:page=00:13) = 00 00000000
(port=01:page=00:14) = 00 00000000
(port=01:page=00:15) = 00 00000000
```

```
(port=01:page=01:08) = F8 11111000
(port=01:page=01:09) = 00 00000000
(port=01:page=01:10) = 00 00000000
(port=01:page=01:11) = 00 00000000
(port=01:page=01:12) = 00 00000000
(port=01:page=01:13) = 00 00000000
(port=01:page=01:14) = 00 00000000
(port=01:page=01:15) = 00 00000000
(port=01:page=07:08) = 00 00000000
(port=01:page=07:09) = 00 00000000
(port=01:page=07:10) = 00 00000000
(port=01:page=07:11) = 00 00000000
(port=01:page=07:12) = 00 00000000
(port=01:page=07:13) = 00 00000000
(port=01:page=07:14) = 00 00000000
(port=01:page=07:15) = 00 00000000
(port=02:page=00:08) = 01 00000001
(port=02:page=00:09) = 00 00000000
(port=02:page=00:10) = 08 00001000
(port=02:page=00:11) = 00 00000000
(port=02:page=00:12) = 28 00101000
(port=02:page=00:13) = 43 01000011
(port=02:page=00:14) = 41 01000001
(port=02:page=00:15) = 95 10010101
(port=02:page=01:08) = 01 00000001
(port=02:page=01:09) = 00 00000000
(port=02:page=01:10) = 08 00001000
(port=02:page=01:11) = 00 00000000
(port=02:page=01:12) = 28 00101000
(port=02:page=01:13) = 43 01000011
(port=02:page=01:14) = 41 01000001
(port=02:page=01:15) = 95 10010101
(port=02:page=07:08) = 01 00000001
(port=02:page=07:09) = 00 00000000
(port=02:page=07:10) = 08 00001000
(port=02:page=07:11) = 00 00000000
(port=02:page=07:12) = 28 00101000
(port=02:page=07:13) = 43 01000011
(port=02:page=07:14) = 41 01000001
(port=02:page=07:15) = 95 10010101
$ q
```

プログラム開発

プログラムの開発手法を述べる必要があるとは思えません，単純なコンソールプログラムです．コンパイラは Win32 に対応したコンパイラなら何でも構いませんが，確認したのは，ポピュラーであろうと思われる，Visual C++です．まず，その前に環境を示します．

Hardware Platform	PC AT 互換機，ボードが PCI スロットに装着できること．
Software Platform	Windows XP(Windows98, Windows 2000 は動作するが未保障)
コンパイラ	Win32 に対応した C++コンパイラ たとえば，Visual C++

Visual C++を起動します.「ファイル」メニューから「新規作成」を選択します.「新規作成」ダイアログで,「Win32 Console Application」を選択します. プロジェクト名の入力と, プラットフォームも指定してください.「Win32 Console Application – ステップ 1/1」が現れますので,「空のプロジェクト」を選びます.「新規プロジェクト情報」が, 表示されますので, 間違いがなければ,「OK」を押します. これで, プロジェクトが作られます. しかし, まだ中身はなにも入っていません. ソースファイルを, Visual C++環境の中で入力することもできます. あるいは, 使い慣れたエディタで作成し, プロジェクトへ追加します. これで, 完了です. ディバッグなどが終わったら, Release の形に再コンパイルして, 完了です. Release の形にするには,「ビルド」→「アクティブな構成の設定」を選びます.「Win32 Release」を選びます. この状態でビルドを行えば, ディバッグ情報が取り除かれた, 実行形式ファイルが作成されます.

プログラム説明

このプログラムは, 3 つのコマンドを受け付けます.

コマンド	説明
DLINK	LINK レジスタを表示します.
DPHY	PHY レジスタを表示します.
Q	Quit します.

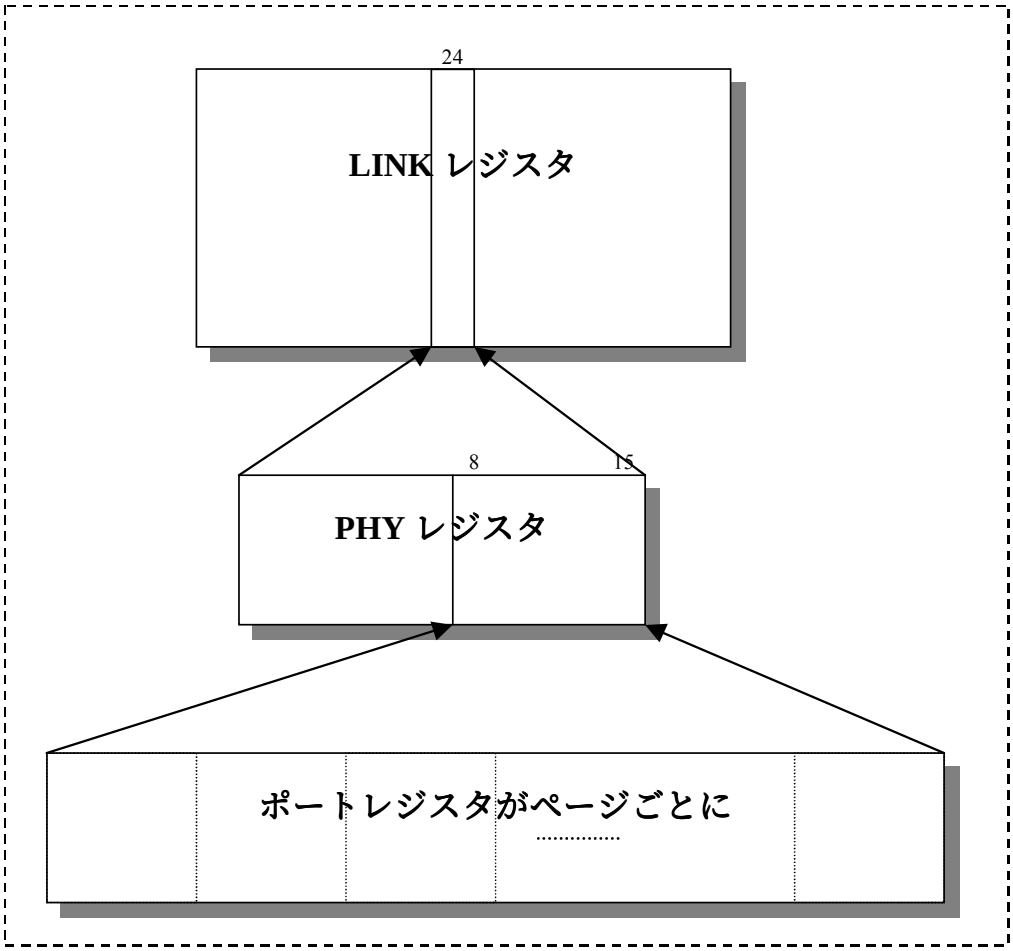
コマンド処理はテーブルドリブンにしてありますので, コマンドを追加するのはテーブルを追加するだけです.

起動直後に, initHardware ルーチンを呼び出しています. アクセスは DWORD(32Bit)で行います. また, iAPX86 アーキテクチャを使いますのでリトルエンディアンです.

LINK レジスタの読み込みを行うのは, readLinkReg ルーチンです. レジスタはオフセット(レジスタ番号ではない)で指定します. 読み出したデータを, QUADLET で呼び出し元に返します.

PHY レジスタは, LINK チップから PHY チップをインダイレクトして読み込みますが, IEEE 1394a では, 拡張されたレジスタを, さらにインダイレクトしなければなりません. PHY レジスタの読み込みを行うのは, readPhyReg ルーチンです. 指定された PHY レジスタを読み出します. レジスタはレジスタ番号(オフセットではない)で指定します.

PHY レジスタは, 直接バスに接続されていません. PHY は LINK レジスタ経由でアクセスしなければなりません. LINK レジスタのオフセット 0x24 に Phy Access レジスタがありますので, ここを使ってアクセスします. Phy Access レジスタのオフセットが 0x24 ですから, Phy Access レジスタは, LINK レジスタのベースアドレスに+0x24 した場所にマップされていることになります. PHY レジスタの 8-15 には, さらに拡張されたポートレジスタがページごとにマップされます. PHY レジスタを読み込むためには, LINK レジスタへの書き込みも必要になります.



LINK レジスタの, Phy Access レジスタは以降のように定義されています. エンディアンの違いがよく分かります. LINK チップのデータシートと良く見比べてください.

```
typedef struct {                        // Phy interface bits
    unsigned    phyrxdata:8,
                phyrxad:4,
                rsvd2:4,
                phyrgdata:8,
                phyrgad:4,
                rsvd1:2,
                wrphy:1,
                rdphy:1 ;
} link_phychip_bits ;
```

この程度が重要な部分です. 細かくはプログラムソースと, ハードウェア解説, そして, チップのデータシートを読めば理解できると思います.

より詳しく知りたい場合は, dumpReg サンプルのソースコードを参照してください.