どすおぶいだにょ☆

創刊号

This Page is Blank.

どすぶいだにょ☆（創刊号） Internet配布版 Revision 0

もくじ

表紙：るりあ046
裏表紙：ChaN

■ 汎用ディスプレイコントローラの製作 ■

CRTにつなぎTai！

または、Xilinxで遊びTai！

本文・挿絵：ChaN

はじめに

何年か前、たった1個の PIC(16C84)だけでテレビに画像を出力するという製作記事がありました。その記事を目にしたときは、目からうろこが落ちるような印象を受けた覚えがあります。これをさくらちゃん風に言えば「ほぇーっ！」でしょうか(笑)。チープな 1 チップマイコンでビデオ信号を合成しようというのは、とても興味深い試みといえます。

しかし、それを実際のシステムに応用するとなると、とたんに壁に当たってしまいます。ビデオ信号合成に殆どの時間を取られるので、必要な処理を円滑に行うことができなくなってしまうからです。しかも、任意のグラフィックを表示することさえ無理で、単純なパターンの表示に限られます。そのせいか、このテクニックの応用例は凝った物でもオシロスコープ程度に留まっています。やはりまともなグラフィック表示をするにはディスプレイコントローラが必要になってくるのです。

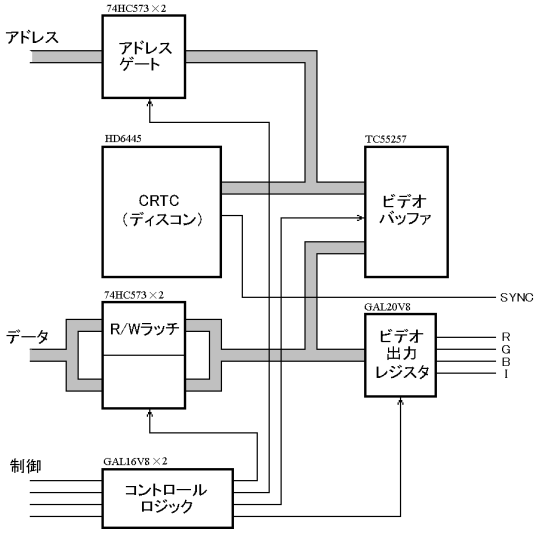
最近は LCD モニタや小型 CRT モニタのジャンク品が多く出まわるようになってきています。自作のマイコンボードにそれらを接続して表示してみたいと思うことが結構ありますよね。そんなとき手軽に使えるディスプレイコントローラがあれば便利です。そこで、1チップマイコンでも手軽に使えるディスプレイコントローラを製作してみることにしました。

ハードウェア

実はこれ以前にも汎用ディスプレイコントローラを製作したことがあります(**図 1**)。しかし、これに使用した第一世代 CRTC(HD46505 の類)は現在は殆どディスコンとなっていて入手が困難になっています。したがって、再現性を保つためには CRTC の代替品を探さなければなりません。

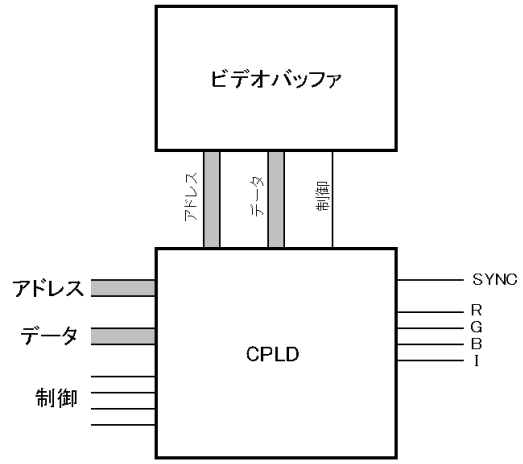
まず考えたのが、**図1**の CRTC を CPLD で置き換えてしまう方法です。実際にロジックを記述してみると、数百円のデバイスにフィットできました。単体の CRTC が消えた理由が何となく分かるような(^^;。この方法は単純ですが、全体から見ると単なる部品の置換ということになり、あまりにも芸が無さすぎて面白くないといえます。したがって、これはリサーチの段階で却下。

〈図 1〉 古い設計のディスプレイコントローラ



つぎに、集積化を進めてちょっと大きめの CPLD を使って周辺ロジックも全て1個の CPLD に突っ込んでしまおうと考えてみました。CPLD＋メモリの2チップ構成です(**図 2**)。これなら結構イケてる設計であろうということで、この案に絞って設計を進めます。

〈図 2〉 今回設計したディスプレイコントローラ



● 基本仕様

開発の目的から汎用性を第一に考えた仕様とします。

まず、なるべく多くのマイコンに容易に接続できるようにするため、制御インターフェースは最も手軽な SRAM 互換とします。このようにしておけば殆ど全てのマイコンに容易に接続することができるはずです。古いマイコンボードの SRAM ソケットへということさえ可能でしょう。外部バスの無いマイコンの場合は、I/O ポートを使って制御することができます。

次に、ビデオ出力は RGB 出力としてジャンクの LCD モニタや RGB 入力のモニタに接続できるようにします。しかし、RGB モニタはどこにでも転がっているというものではないので、ほかに少なくとも普通のテレビに接続できる必要があります。

数インチ以下の LCD モニタに表示するにはテレビゲーム程度の 320×240 ドットもあれば十分でしょう。テレビに映す場合も同様です。これらはパソコンモニタと違い解像度が低いので、640×480 などの高密度表示ではドットが潰れてしまって見難いです。また、それだけデータ量が増えることになり、チープなマイコンには重荷となるなどデメリットばかりとなってしまいます。同様に色数についても自然画像の表示が目的ではないので、16 色程度で十分と言えます。

ディスプレイコントローラの使用目的とは関係ないですが、Xilinx の CPLD を使用することも条件にします。このプロジェクトが浮上したのも、最近流行の Xilinx の XC9500 シリーズを試食しTai！ということからで、このプロジェクトのもう一つの目的になっています。

以上のような条件からディスプレイコントローラの基本仕様を次のように設定しました。

・特殊部品はなるべく使わず、単純化して再現性を確保。
・多くのマイコンに容易に接続できるように SRAM 互換インターフェースとする。
・表示の柔軟性のため、フルグラフィック表示とする。
・解像度や色数は手ごろな 320×240、16 色。
・RGB モニタのほかテレビにも接続できる。
・CPLD には XC9500 シリーズを使う。

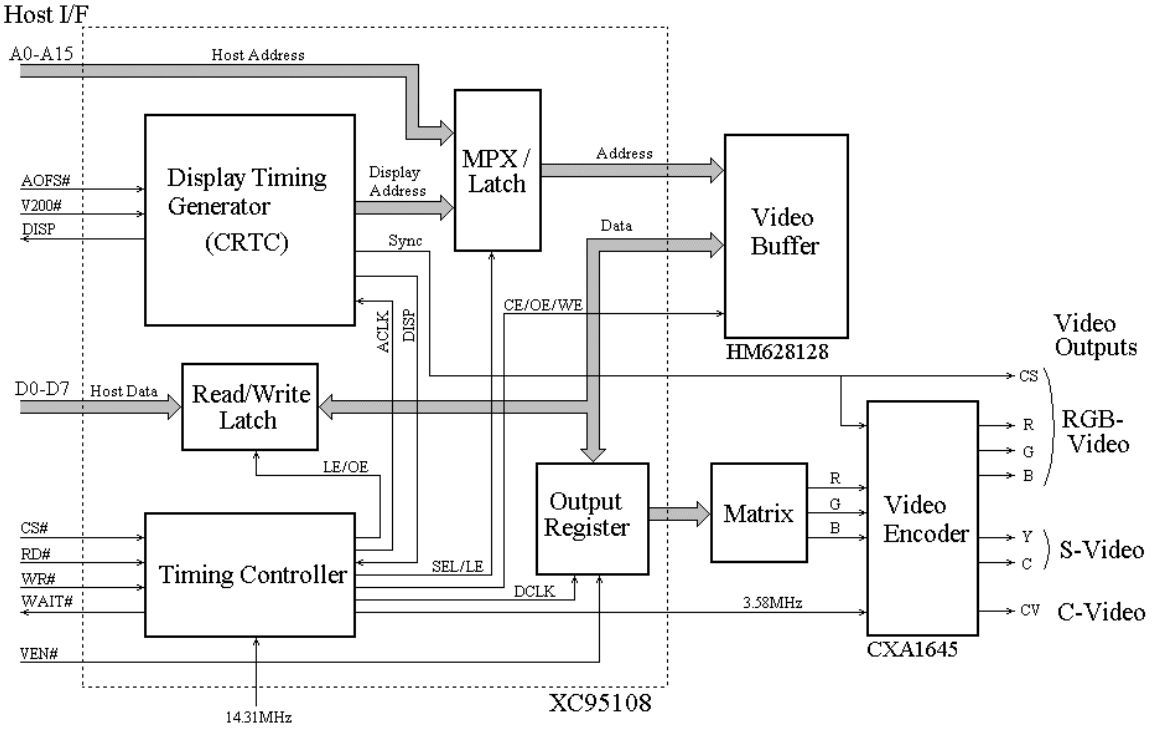
● ブロック図

図 3 に本器のブロック図を示します。CPLD には最近流行りの Xilinx XC9500 シリーズのうち中規模の XC95108(PLCC)を使用します。このチップは 108 個のマクロセルと 69 本の I/O ピン(うち数本はグローバル機能ピン)を持つもので、今回の製作にはちょうど良い規模です(**写真 1**)。

CPLD には制御機能を全て詰め込んでしまいます。これにより回路を大幅に単純化することができました。CRTC 部はディスプレイ出力のタイミングを生成するカウンタブロックです。タイミングコントローラ部はホストからのアクセスと表示アクセスの調停と実際のメモリアクセス等を制御するメインロジックです。

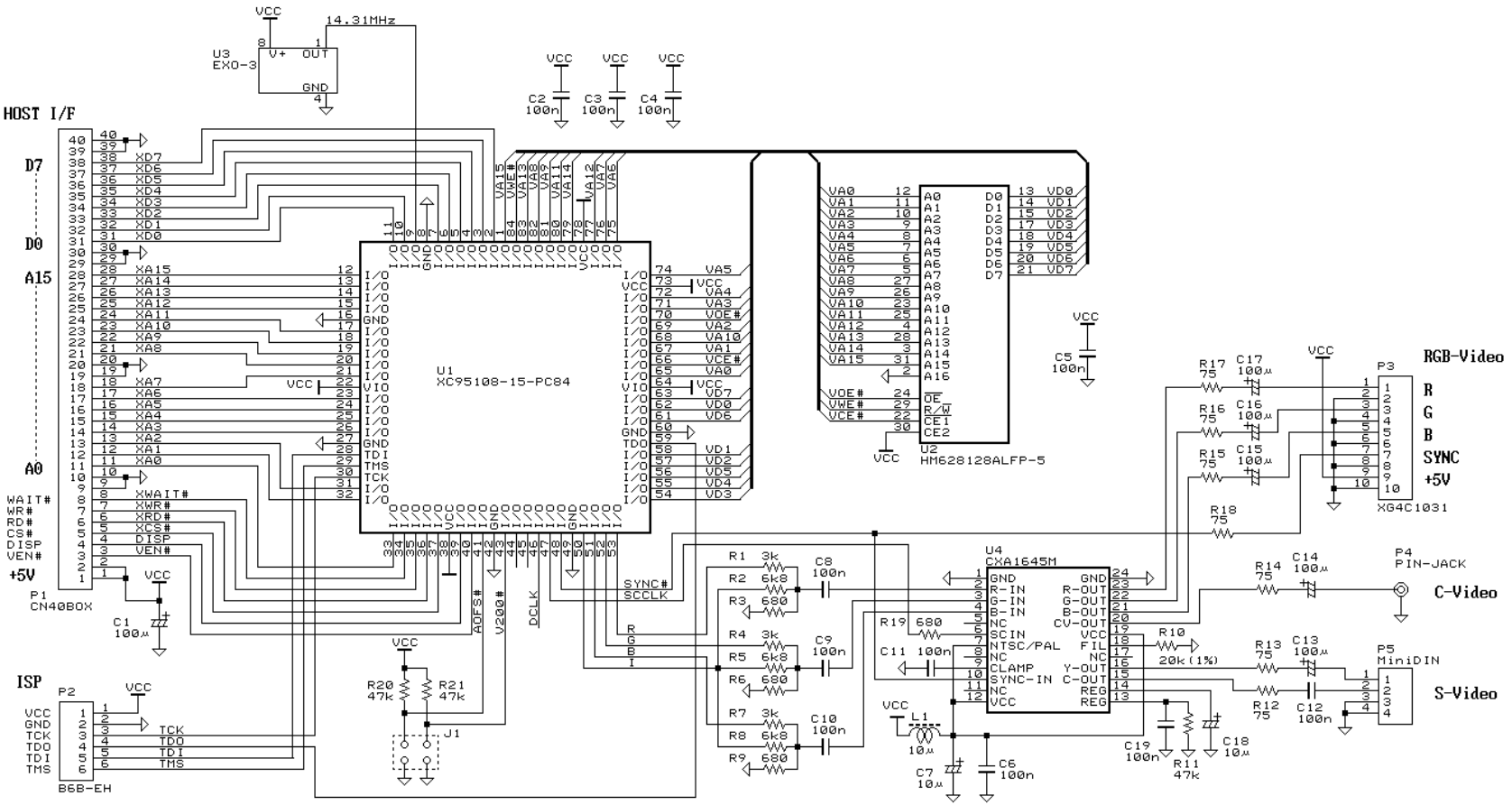
CPLD 以外で必要な半導体部品は SRAM とクロックオシレータくらいです。ビデオバッファには入手しやすい一般的な 128K ビット SRAM を使用しています。今回は RGB 出力のほかにさらに NTSC 出力が必要なため、RGB-NTSC エンコーダ(CXA1645)も追加しています。

〈図 3〉 ディスプレイコントローラのブロック図



〈写真 1〉 XC95108(PLCC)

〈図 4〉ディスプレイコントローラの回路図



● 回路図

図 4 に本器の全回路図を示します。とても簡単ですね。IC も 3 個しか使っていませんし、配線数も前作と比べて大幅に減らしています。これくらいなら誰でも簡単に作れるといえるレベルでしょう(最低でも CPLD 開発環境は必要ですが)。CPLD は回路に組み込んだまま書き換える必要があるので、ISP 用のコネクタも設けてあります。

CPLD からのビデオ出力データ(4 ビット)は抵抗マトリクスで RGB に変換されます。RGB 信号は CXA1645M で NTSC 信号にエンコードされ、同時に RGB・SV・CV に 3 分配されます。エンコードに必要なカラーサブキャリアクロックは CPLD で生成して CXA1645M に供給されます。この辺りはアナログ信号処理なので、電源のデカップリングをしっかりしないと画質が悪くなります L1 と R19 はデジタル部からの影響を低減するために必要な部品なので、無くても動作はします。

● 部品表

表 1 に本器の部品表を示します。特殊なのは無いので、パーツ屋を一巡すれば揃うでしょう。コミケ帰りはアキハバラに寄って同人誌を片手に部品集めというのも良いのではないかと思います(^^;。 **表 2** に販売店の例を示します。

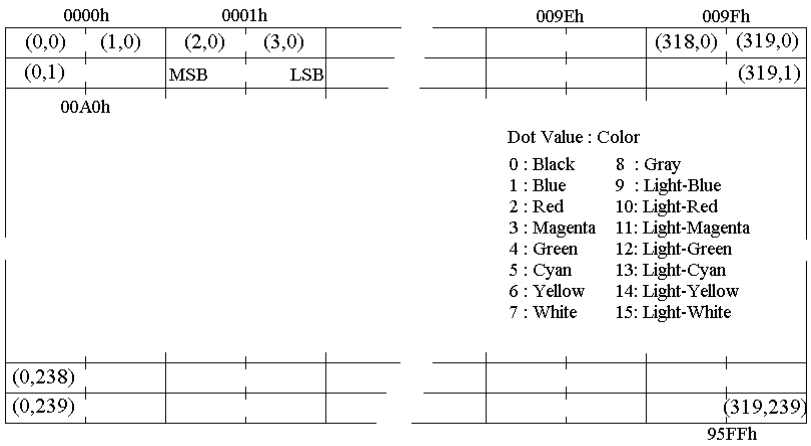
<表 1> ディスプレイコントローラの部品表

項目	部品番号	部品名	型番	数量	メーカ
1	U1	CPLD	XC95108-15PC84C	1	Xilinx
2	U2	SRAM	HM628128BLFP-5	1	日立
3	U3	オシレータ	EXO-3　14.31MHz	1	キンセキ
4	U4	ビデオエンコーダ	CXA1645M	1	ソニー
5	R1,R4,R7	抵抗	3kΩ	3	
6	R2,R5,R8	抵抗	6.8kΩ	3	
7	R3,R6,R9,R19	抵抗	680Ω	4	
8	R12,R13,R14,R15,R16,R17,R18	抵抗	75Ω	8	
9	R11,R20,R21	抵抗	47kΩ	3	
10	R10	抵抗	20kΩ±1%	1	
11	C2,C3,C4,C5,C6,C7,C8,C9,C10,C11,C12,C19	チップコン	100nF	12	
12	C7,C18	ケミコン	10μF	2	
13	C1,C13,C14,C15,C16,C17	ケミコン	100μF	6	
14	L1	マイクロインダクタ	10μH	1	
15	P1	コネクタ	40ピン(20x2)ピンヘッダ	1	
16	P2	コネクタ	6ピン(3x2)ピンヘッダ	1	
17	P3	コネクタ	10ピン(5x2)ピンヘッダ	1	
18	P4	コネクタ	ピンジャック	1	テイシン電機
19	P5	コネクタ	MJ373/4	1	マル信無線電機
20	J1	ジャンパSW		1	
21		基板		1	

<表 2> アキハバラでの販売店の例

部品名	販売店	参考価格
CPLD	若松通商、その他	¥2,000
SRAM	田中無線電機、若松通商、その他	¥800
ツリガネ	秋月電子	¥500
ビデオエンコーダ	若松通商、秋月電子	¥850
うさだ	ゲーマーズ	¥1,050
オシレータ(缶入りも可)	サンエレクトロ、その他	¥400
チップ抵抗	シーアール、千石電商(B1)	¥4
ぷちこ	ゲーマーズ	¥300
チップコン	シーアール、千石電商(B1)	¥10
MINI-DINコネクタ	千石電商(1F)	¥200
でじこ	ゲーマーズ	¥5

〈図 6〉 画面上のドットとバッファメモリの対応



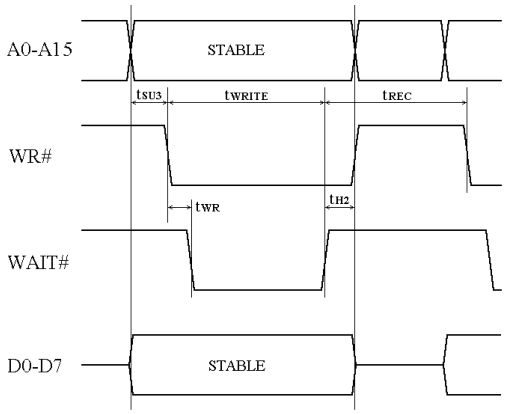
Note1: 2 ピクセルで 1 バイトのパック化ピクセルになっています。

Note2: AOFS ジャンパが ON のときはアドレスに 4000h が加算されます。

Note3: V200 ジャンパが ON のときは表示ライン数が 200 になります。

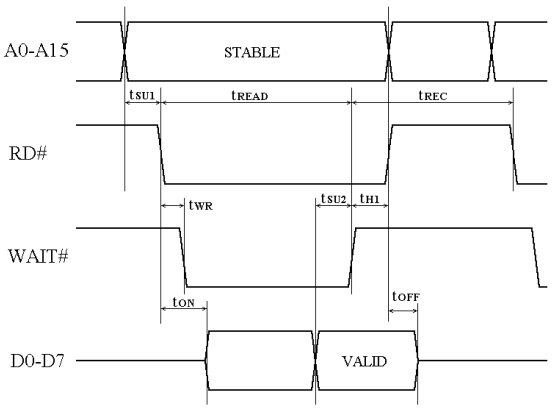
〈図 7〉 外部インターフェースアクセスタイミング

書き込みサイクル



※CS# = L

読み出しサイクル



※CS# = L

タイミングパラメータ

項目	意味	最小	最大	単位
tWR	WAIT# 応答時間		15	ns
tSU1	読み出しアドレスセットアップ時間(対コマンド)	-40		ns
tREAD	読み出しコマンド実行時間	140	485	ns
tH1	読み出しアドレス・コマンドホールド時間	0		ns
tSU3	書き込みアドレス・データセットアップ時間(対コマンド)	-40		ns
tWRITE	書き込みコマンド実行時間	70	415	ns
tSU2	読み出しデータセットアップ時間(対WAIT#↑)	15		ns
tH2	アドレス・コマンド・データホールド時間	0	∞	ns
tON	バスドライバON時間		9.5	ns
tOFF	バスドライバOFF時間		9.5	ns
tREC	アクセスリカバリー時間	70		ns

製作のポイント

● 組み立て

製作にあたって特に注意する点はありません。もちろん、電源ラインをしっかりするとかパスコンを入れるとか部品実装する上で一般的なことはちゃんやっておく必要はありますが。今回はデジタル回路とアナログ回路

が混在なので、いい加減に作ると画質が悪くなるのでそのへんも注意して組み立てる必要があります。製作したプロトタイプを写真 2 に示します。

● CPLD 開発環境の準備

このドキュメントを読んでいる人で知らない方はいないかと思いますが、PLD は買ってきたままではなんの機能も持たないただの石です。ユーザで用意したロジックを書き込んで初めて意味を持った動作をするようになるのです(PROM と同じですね)。

XC9500 シリーズの場合、書き込みには ISP ケーブルと CPLD 開発ツールが必要です。まだ CPLD 開発環境を持っていないならこれを機会に一つ作ってしまいましょう。ISP ケーブルは簡単に自作できます(**図 5**)。ツール類は Xilinx の web ページ(www.xilinx.com)から無料でダウンロードできます。巨大なファイルがたくさんあるのでどれをダウンロードしたら良いのか迷いますが、とりあえずコンパイラ(ソースコードから論理合成する)、フィッタ(論理をデバイスに組み込む)、プログラマ(ヒューズデータをデバイスに書き込む)をインストールすれば通常の開発には十分です。シミュレータの類は開発の補助的役割を果たします。それらはコンパイラを使いこなせるようになってからでも良いでしょう。

● 動作確認

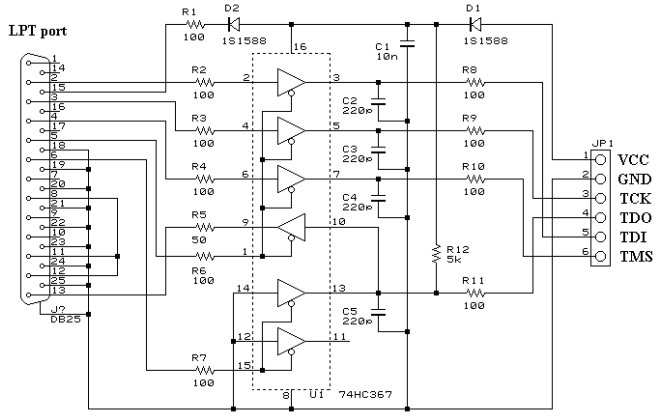
ほとんどがデジタル回路なので、調整個所はありません。回路に間違いが無ければ一発で動作するはずです。配線に間違いが無いかチェックしたら、電源を入れて JEDEC ファイル(crtif2a.jed)を XC95108 に書き込みます。

インターフェースバスにはプルアップを入れていないので、電源投入の瞬間に数 100mA の比較的大きな電源電流が流れるかもしれませんが問題ありません。200mA 程度になっていれば正常です。

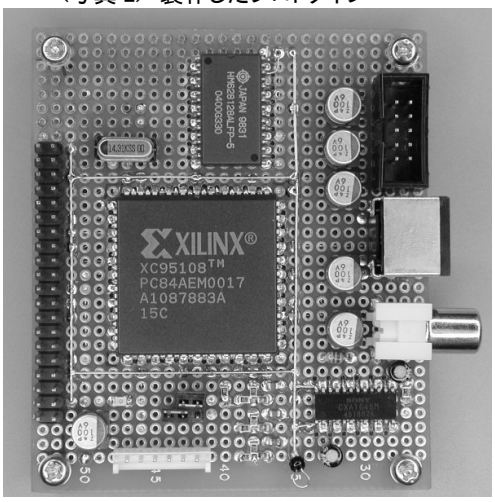
ディスプレイコントローラはホストコントローラと組み合わせてはじめて意味を持つものなので、単体では完全な動作確認ができません。

でも、ホストインターフェース部以外の動作は単体でも確認できます。ビデオ出力をテレビに接続して VEN#をグランドに落とします。画面いっぱいにランダムパターンが表示されて同期が取れていれば OK です。どんなパターンになるかは個々の SRAM チップによって異なります(真っ黒または真っ白かもしれない)。

〈図 5〉 Xilinx 用ダウンロードケーブル



〈写真 2〉 製作したプロトタイプ



使い方

● メモリマップ

水平 320 ドット、垂直 240(200)ラインの表示となります。ディスプレイメモリ上のデータと画面に表示されるドットとの関係を**図 6** に示します。1 バイト当たり 2 ドットのパック化ピクセルとなるので、1 水平ライン当たり 160 バイトということになります。

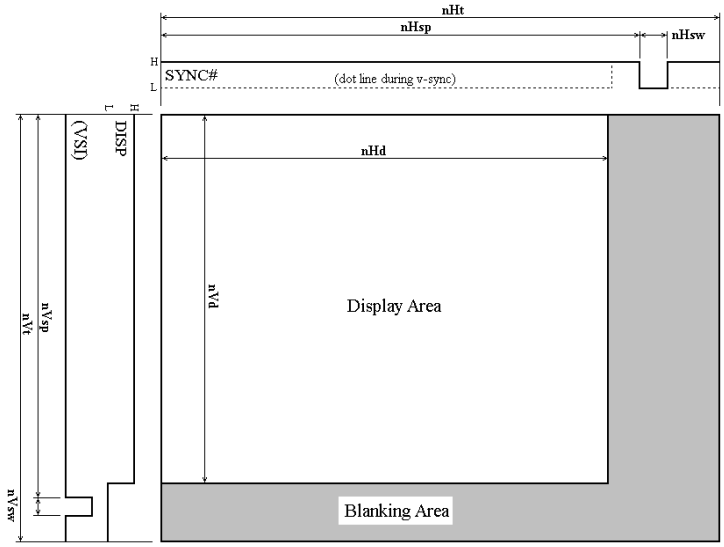
メモリアドレスは 16 ビットなので、64K バイトのメモリ空間となりますが表示エリアは 0000h～95FFh(240 ライン時)までとなります。この表示エリアは AOFS ジャンパで 4000h～D5FFh へずらすこともできます。非表示エリアのメモリは任意の用途に使えます。

240 ライン表示では、普通のテレビに表示したとき上下が画面から若干はみ出します。この場合は、V200 ジャンパで表示を 200 ラインに変更できます。200 ラインモードでは表示メモリエリアが 0000h～7CFFh となり 32K バイトに収まるので、物理メモリ空間の狭いマイコンでも楽にマッピングできると思います。

● インターフェースタイミング

ホストインターフェースを通してディスプレイメモリを直接アクセスして描画します。SRAM 互換インターフェース

〈図8〉 ビデオ出力タイミング



タイミングパラメータ

項目	意味	値	単位
nHt	水平トータル期間	227.5	ACLK
nHd	水平表示期間	160	ACLK
nHsp	水平同期位置	181	ACLK
nHsw	水平同期幅	17	ACLK
nVt	垂直トータルライン数	263	Line
nVd	垂直表示ライン数	240	Line
nVsp	垂直同期位置	224	Line
nVsw	垂直同期幅	3	Line

〈表 3〉 ビデオ出力の電気的仕様

・RGB ビデオ出力

RGB：0.7V p-p （75Ω終端時）

同期：H-V 混合（TTL レベル負極性）

・C ビデオ出力

1.0V p-p（75Ω終端時）

・S ビデオ出力

Y：1.0V p-p （75Ω終端時）

C：286mV p-p （75Ω終端時）

なので、基本的に SRAM を読み書きするのと同じ手順です。ただし、内部動作との調停のため、バスに直接接続する場合はタイミングによっては WAIT#を使ったハンドシェークが必要になります。実際のタイミングチャートを**図 7** に示します。図では RD#,WR#ストローブとなっていますが、CS#ストローブでも使用可能です。

● ディスプレイ出力

ビデオ出力の電気的仕様を**表 3** に、ディスプレイタイミングを**図 8** に示します。基本的にテレビゲームなどと同じノンインターレース出力です。

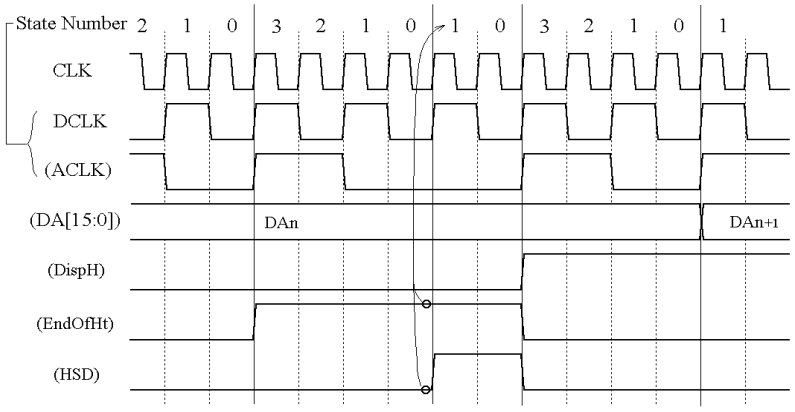
● 内部動作タイミングチャート

図 9、**図 10** に内部動作のタイミングチャートを示します。ACLK を1サイクルとして外部アクセスと表示アクセスがオルタネート動作となっているのが**図 10** を見ると分かるかと思います。外部アクセスの最大データ転送レートは、ACLK が 3.58MHzであることから 3.58MB/秒ということになります。

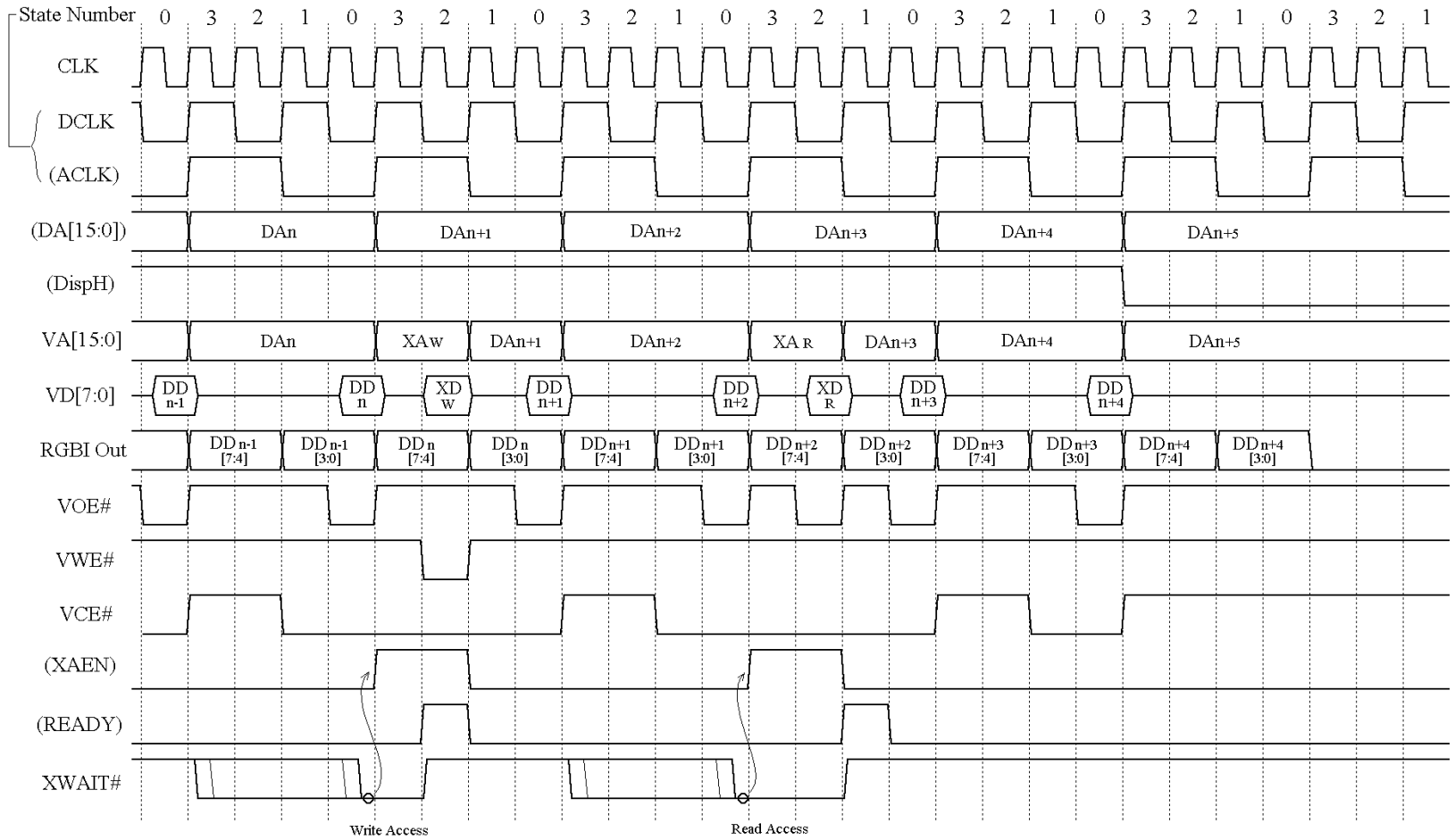
〈図 9〉 内部動作タイミング
（nHt 付近）

NTSC エンコードの際、輝度信号と色信号のスペクトラム・インターリーブのため、水平期間がカラーサブキャリアの xxx.5 サイクル分となるように調整する。

※括弧付きは内部信号（ノード）



〈図 10〉 内部動作タイミング （nHd 付近）



※括弧付きは内部信号(ノード)

最後に

● CPLD の使い心地

仕事では Lattice の CPLD を使うことが多く、Xilinx を使ったのは今回が初めてだったりします。わざわざ Xilinx の CPLD を使ってみたのは、みんなが使っているから(笑)という他愛も無い理由からで、どんなものかという評価の意味もありますが。

まず、WebPack の Fitter がタコで最初から思いっきりハマりました(笑)。いくら只ツールといってもこれはちょっと酷いと思います。どうも Fitter が CPLD の機能に追いついていないという感じがしました。

デバイスの論理的アーキテクチャは悪くはないと思いますが、プロセスが悪いのか規模の割にかなり消費電流が多いです。XC95108 の場合、1W 近く消費します。このシリーズはバッテリ動作機器には無理でしょう。

マクロセル数とI/Oピン数のバランスはとても良いです(用途にもよりますが)。今回のプロジェクトにはジャストフィット(マクロセル 100%使用)でした。

で、結論としては XC9500 シリーズはかなり使えそうだというのが率直なところ。開発ツールについてもバグを避けつつ使えば十分使えます(バグを避けつつ....開発ツールってのは大体そんなものですが)。個人で評価用に使う分には十二分といったところでしょうか。何よりもデバイスが安価で入手しやすいのがポイントです。アマチュアにこれだけ普及している理由も自ずと知れてきますね。

● 何に使うか

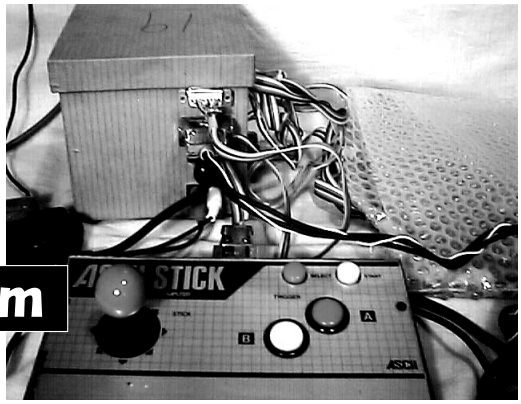
さて、何かに使いたいと思って製作した汎用ディスプレイコントローラですが、これは最終目的ではなくほかのことを実現するための手段に過ぎません。これを実際に何に使ったらよいのでしょうね。面白い考えがなかなか浮かんできません。まぁ、半分は CPLD の評価もあって、特に必要に迫られて設計したものではないですからね(^^;。

でも、ジャンクのLCDモニタが大量に出まわるなど、その手の需要は結構ありそうです。ソースをちょっといじってマスタークロックを 25.175MHz にすれば VGA モニタ(640x480、単色)への出力も簡単にできます。あとはアイディア次第 :-)

※使用した PLD ソースファイルは、
ftp://elm-chan.org/pub/ で公開しています。

激チープなゲームコントローラの製作

Kim



☆チープに 簡単に 快適に！

ここで紹介するのは、筆者が今までに製作・改造してきたゲーム機関連アイテムです。筆者は電子回路の知識が全くないので、泥臭いローテクを駆使して作ってます。本人は別に変な事をしているつもりはないのですが、外箱にボール紙を使ったりしているので、周りからは奇異な目で見られる事もしばしばです。しかし、これこそ、世界にひとつしかない「自作」の醍醐味とも言えるのではないでしょうか。

☆ジョイスティック・パッド

【アスキースティック改】

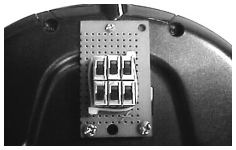
〈写真１〉ファミコン用とＰＣエンジン用の２本のケーブルが出ている。

〈写真２〉トグルスイッチ２つは連射切り替え用。１つはBボタンとセレクトを入れ替えるための物。D-Sub9 ピンコネクタは拡張用で、各種アタッチメントを接続する。

写真１は元祖ファミコン向けにアスキーから発売されていた「アスキースティック」というジョイスティックを汎用に改造したものです。連射機能も付けました。背面に２本出ているケーブルは、それぞれ元祖ファミコンとＰＣエンジン用です。構造は単純で、中にそれぞれの機種のコントロール基板を埋め込んで配線してあるだけです。これぞまさにハイテクならぬローテク！

【ファイティングパッド6B改】

〈写真３〉中央上部に増設されたトリガーがある。セレクトボタンやコイン投入に使う。



〈写真４〉裏面にはトリガー６つ分の連射切り替えスイッチが並ぶ。連射回路そのものは内蔵しておらず、外部の回路を使う。

写真３はセガ・メガドライブ用の純正ジョイパッドを汎用に改造した物です。初期の３トリガーのパッドは大きくてこの世の物とは思えないほど使いづらかったのですが、この６トリガーの後期型は小さく、パッド部分も扱いやすくなり、数あるゲーム用パッドの中でもトップクラスの使いやすさです。どうもセガという会社は、やる事にムラが多いような気がします。

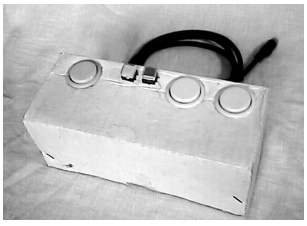
【汎用コントローラ接続用アタッチメント】

〈写真５〉メガドライブ用アタッチメント。もう一本のケーブルは汎用コントローラを接続する。

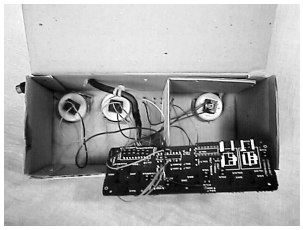
〈写真６〉ＰＣエンジン用アタッチメント。連射回路はこの中のものを使う。

製作した汎用コントローラを接続する為の物です。メガドライブ用とＰＣエンジン用があります。中の配線にコネクタをつなげただけです。

【パックランド専用コントローラ】



〈写真６〉ＰＣエンジン版専用。中央２つはセレクトボタンとスタートボタン。外箱の材質はボール紙。とても軽い。



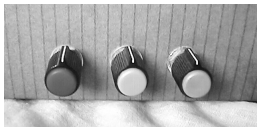
〈写真7〉外箱を開けたところ。パッドの基板をそのまま入れて配線しただけ。

ＰＣエンジン版ナムコ「パックランド」の専用コントローラです。もともとアーケード版は３つのボタンだけでプレイする変則的なコンパネだったので、それを再現しています。段ボールの外箱にパッドの回路をそのまま埋め込んで配線してあります。ほとんど勢いだけで作ってしまいました。製作費は 1000 円もかかってないと思います。

☆ゲーム基板コネクタボックス

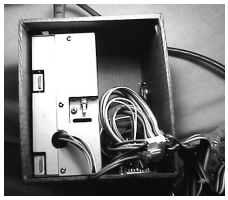
〈写真８〉コネクタは上から１Ｐ、２Ｐ、画面出力、音声出力になっている。 Macや CobaltQube を先取りした精悍なデザイン。（これもボール紙）しかし、NeXT には負ける。

ゲームファンが自宅でゲーム基板で遊ぶときは、筐体の代わりに「コントロールボックス」という電源、ジョイスティック、スピーカー等が内蔵された装置を使うのが普通です。コントロールボックスは市販品もありますが、２万円以上と値が張り、また場所も取るので保管も面倒です。ジョイスティックは既に製作済みなので、後は電源とコネクタボックスさえあれば、ゲーム基板で遊べる！という訳で製作しました。



〈写真９〉側面に取り付けられた色合い調整用の可変抵抗。それぞれ RGB に挟んである。

〈写真 10〉背面には電源コネクタと排気孔がある。



〈写真 11〉中はほとんど電源が占めている。

中は電源ユニット（千石電商で 1500 円）とコネクタと色合い調整用の可変抵抗が入っているだけです。必要な信号はゲーム基板からだいたい出ているので、コネクタさえ用意できればゲームは出来ます。費用はあわせて 4000 円くらいでした。同期分離回路は手持ちの物を使いましたが、昔のゲームラボ誌（95 年２月号とか）を見ると回路図が載っているので、自作も出来るでしょう。漏電対策などしていない危険極まりないシロモノですが、売り物ではないので、そこも味の内ということで勘弁してもらいましょう。

☆自作の道は情熱あるのみ

以上の物は全て電子回路を組まずに作ったものです。なので、あまり格好いいものではありませんが、箱に穴を空けたりハンダゴテを握って物を作る感覚は、ゲームでは味わえないものです。電子工作は敷居の高いものばかりではないですよ。

〈写真 12〉実際にゲーム基板を接続しているところ。同期分離回路としてメガドライブ用 RGB ユニットを流用している。同じ物が LM1881N というチップで比較的簡単に作成出来る。サンプルに使ったのはコナミの横スクロールシューティング「ゼクセクス」。このような、今後絶対に移植が出ないと思われるゲームを堪能できるのも、基板マニアの悦びなのだ。

近年RGBモニタ事情

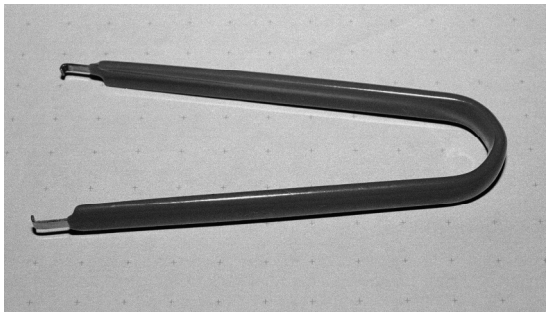
画質にこだわるゲームファンは、モニタは絶対にRGB接続でしょう。しかし、適当な 15kHz のRGBモニタが無いことは長年の悩みのタネでした。ＮＴＴ（当時はまだ電電公社でしたか）のキャプテンや MSX がまだ注目を集めていたころは、家電メーカーのAV対応型テレビには21ピンRGBコネクタがついてましたが、今ではソニーのプロフィールの１機種のみです。他は 15 インチモニタの中古市場くらいでしょう。そこに現れた救世主がソニーの AV マルチ端子。プレイステーション用とはなっていますが、手ごろな価格で最新の 15kHzRGBモニタが手に入るとは、まさに夢のようでした。しかし、AV マルチ端子が付いている機種は、悪名高い４倍DRC機か、ギラギラと強調された画質の機種しかありません。これではせっかくの RGB もだいなしです…（涙）。 さらに輪をかけて、コピーガード信号問題での RGB への逆風もありました。RGB モニタをめぐる環境が改善する日は来るのでしょうか。ちなみに、X68k 用の 15 インチモニタ(うちのは CZ-602D)にPS2を RGB 接続して DVD-Video を見ると、ソースのエンコードさえ良ければ、恐ろしいくらい綺麗に映ります。

AT互換機のBIOS ROMのバックアップを作ろう

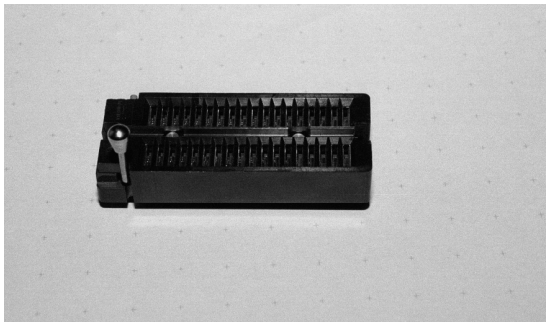
kiyonari@cds.ne.jp　きよりん

AT 互換機自作派にとってさけて通れないのが BIOS のアップデート作業ですが、この作業はベテランでも緊張するものです。この作業に失敗してマザーボードを起動不能にしてしまう事故が後をたちません。
そこで、万が一、起動不能にしてしまった場合に備えて BIOS　ROM のバックアップを作ってみましょう。

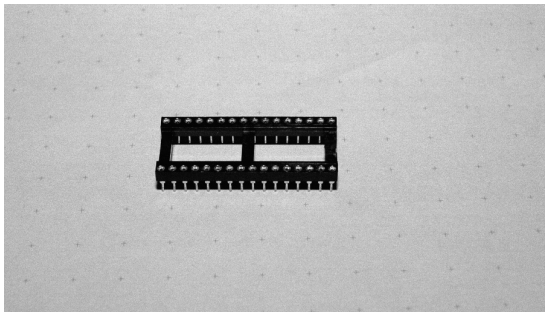
用意する物



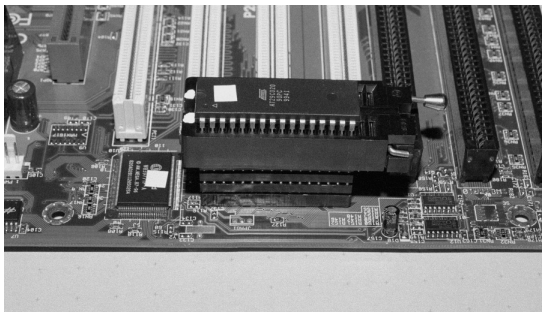
IC 引き抜き工具
BIOS　ROM をマザーボードから安全に引き抜くために IC 引き抜き工具を使いましょう。
価格は３００円くらいからで、秋葉原の工具屋さんで販売しています。



ゼロプレッシャー IC ソケット　１個
IC を安全に抜き挿しできるように作られた IC ソケットです。レバーを動かして IC の足をくわえたり離したりします。３２ピンのものが必要です。写真では手持ちのものを使ったため、３６ピンの物になっています。



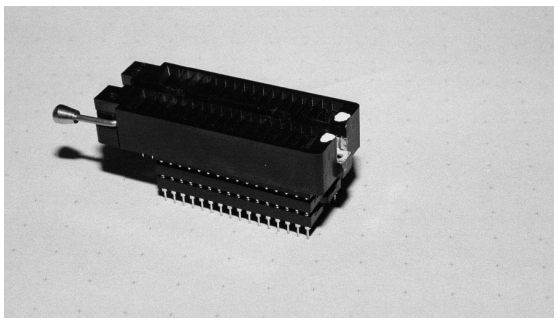
IC ソケット　３～４個
通常の IC ソケットです。ちょっと値段が高くなりますが、ピンが丸ピンのものが強度がありますのでお勧めです。これも３２ピンのもの。ソケットを３～４個重ねて高さのかさ上げをするのは、ゼロプレッシャーソケットを直接マザーボードの IC ソケットに差し込むことができない場合があるからです。



バックアップ用の新品のフラッシュ ROM
これはマザーボードに使われているものと同じか同等のものを用意します。
フラッシュ ROM の種類を調べるには BIOS 書き換えユーティリティを起動して確認します。

部品の購入について

各部品は秋葉原の電子パーツを扱っているお店で購入します。フラッシュ ROM ですが、秋葉原でも取扱店は少なく、Laox ザ・コンピューター館隣の若松通商かラジオデパート２Ｆの光南電気で入手できます。（注）
ゼロプレッシャー IC ソケットですが、３Ｍ社のテキスツールというものは入手がたやすく、扱っている店は多いのですが、値段が少々高く３２ピンのものは約２５００円ほどです。ジャンク電子パーツ販売の秋月電子通商で扱っている ARIES 社のものはこれの半額くらいですが、入荷が不安定で品切れの時があります。
（注）１１月２３日現在、若松通商では 29F020 などの２メガビットフラッシュ ROM の在庫がありません。入荷時期未定だそうです。



フラッシュ ROM はくれぐれも挿し込む向きをまちがえないでください。逆に刺すと ROM が破壊されます。ベテランでもやってしまうことがあります。IC には向きを示すために切り欠きやマークがあるのですが、これでもうっかりまちがえてしまうことがあるため念のためにマーカーペンやシールなどで IC とソケットにマーキングをしておきます。また、わたしのようにソケットにピンが多めのものを代用する場合も挿し間違いが起きないようにテープなどで余分なピンの部分をマスキングしておきます。

起動ディスクの作成

BIOS　ROM 書き換えユーティリティは Windos の MS － DOS プロンプトでは動作させることができませんので、純粋な DOS の英語モードでフロッピーディスクから起動させるための起動ディスクを作成します。

Windows の「コントロールパネル」－「アプリケーションの追加と削除」－「起動ディスク」から起動ディスクを作成してください。そしてフロッピーディスクの内容を次のようにします。

Windows ９５、９８及び９８ SE（起動ディスクの１）の場合
以下のファイルを残して他のファイルを削除します。
IO.SYS と MSDOS.SYS は不可視属性ですからエクスプローラーの設定をすべてのファイルが見えるように設定します。

COMMAND.COM
IO.SYS
MSDOS.SYS
JKEYB. SYS
CONFIG.SYS

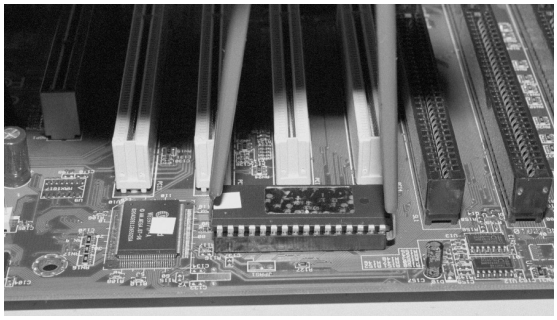
そして CONFIG.SYS の内容を以下のように編集します。

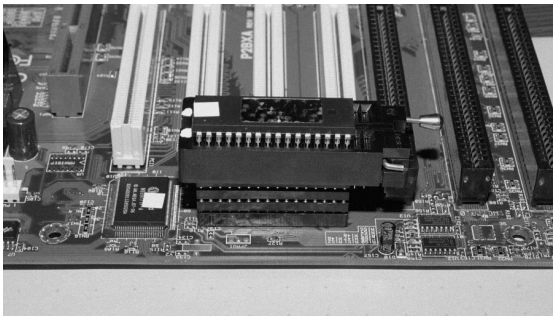
CONFIG.SYS の内容

files=10
buffers=10
DEVICE=JKEYB.SYS

WindowsMe の場合ですが、Me は純粋な DOS というものが排除されてしまった OS で、起動ディスクを作って DOS を起動しても必ずメモリーマネージャーが組み込まれてしまうようです。そのために BIOS 書き換えユーティリティが正常に動作しないことが考えられます。現在はまだ Me に関しての情報が少ないので詳しいことがわかりません。ただ、当方での実験では前記した同じ内容に起動ディスクを設定したもので「AWDFLASH」という書き換えユーティリティは正常に BIOS　ROM の書き込みがおこなえたことを報告しておきます。

設定を終えたら、BIOS 書き換えユーティリティと書きこみたい BIOS のファイルをフロッピーディスクに入れます。

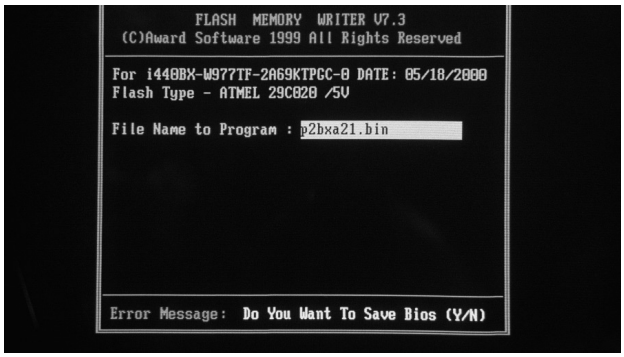




BIOS　ROM の抜き取り

IC 引き抜き工具を使ってマザーボードの BIOS　ROM を抜き取ります。そして、IC ソケットを数段かさねた上にゼロプレッシャーソケットを差し込んだものをマザーボードにセットします。マーキングに注意して向きをまちがえないようマザーに付いていた BIOS ROM を取り付け、起動ディスクを入れてコンピューターを起動します。なお、BIOS の設定はフロッピードライブから起動できるようにします。また、シャドウ RAM の設定項目がある機種はシャドウ RAM の設定を必ずおこないます。

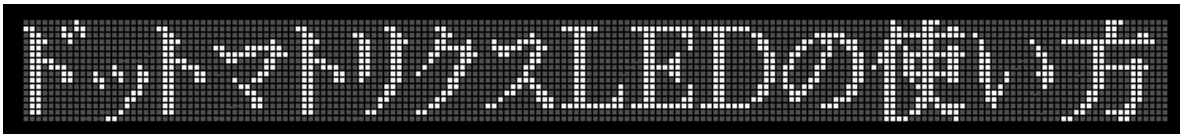
AWDFLASH の起動



コンピューターのコマンドプロンプトが表示されたら BIOS 書き換えユーティリティを起動します。フラッシュ ROM をまだ購入していない場合は、ユーティリティの指示に従って書き換えたい BIOS のファイル名を入力すると現在使われているフラッシュ ROM の種類が表示されます。（AWARD 製のユーティリティの場合）作例のマザーボードの場合ですと、SST　29EE020 というフラッシュ ROM が使われていました。これをメモして半導体を扱っているお店に行きます。例で購入したのは ATMEL 社の AT 29C020 という同等品です。

新品のフラッシュ ROM が用意できたのなら、BIOS 書き換えユーティリティを起動した時点で BIOS　ROM を新しいものに挿し替え、書きこみたいファイル名を入力して BIOS に書きこみます。書き込みが終了したのなら一旦電源を切り再起動し、正常に書きこまれているかどうかを確認します。

バックアップ ROM ができたのならばソケットを外して ROM をセットし直して終了です。でき上がった ROM は静電気による破壊を防ぐためにアルミホイルに包んで保管します。



本文:ChaN

はじめに

最近、ドットマトリクスLED 表示器が流行っている。駅に街角にと至るところで見られるようになってきた。電子系エンジニアでこれに興味を持たれた方も多いのではないだろうか。

実は私もこれに惹かれたクチである(表示器に見入って電車に乗り損ねたり(^^;。そしていつかは仕事で設計してみたいと日ごろから思っていた。そしてついにその手の話が舞い込んできたのである。まだ受注が確定したわけではなかったのだが、私は喜び勇んで先行して全力リサーチを開始して技術をどんどん蓄えていった。だが.... 結局その仕事はお流れ(;_;)。やり場のない悔しさとはこのことか(笑)。しばらく後になってから別件でLED コントローラの設計をすることになり、とりあえず希望はかなったのだが。まぁ、その話はこれくらいにしておこう。

さて、これだけ身近になった分野にもかかわらず、ドットマトリクス LED 応用機器の開発に必要となる技術を解説をした文献がほとんど無いようである。そこで、せっかく蓄えたノウハウをしまっておくのももったいないので、ドキュメントとして簡単にまとめてみることにした。

LED パネルの設計方法

● ドットマトリクス LED の構造

ドットマトリクス LED の内部は**図 1** のように LED がマトリックス状に並んで結線されている(だからドットマトリクスというのだろうけど)。16×16 ドットの単色モジュールならローラインとカラムラインがそれぞれ 16 本の計 32 ピンとなる。2 色モジュールならカラムラインの数が倍になって 48 ピン。

当然のことながらドットマトリクス LED を点灯させるには、点灯するローラインを順次切り換えながらのダイナミックスキャン方式ということになる。

● ドライブ回路の基本構成

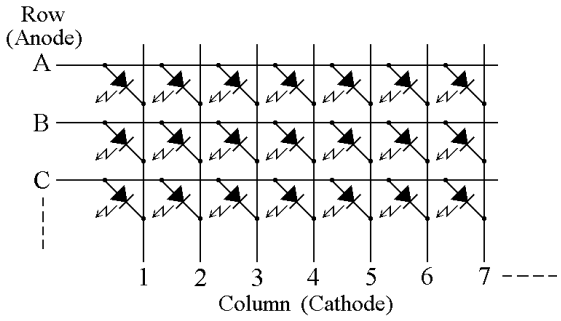
◆基本構成

ドットマトリクスLED 駆動回路の基本構成を**図 2** に示す。カラム駆動回路はシフトレジスタ、ラッチ、LED ドライバ(シンク駆動)の 3 段構成となっている。ロー駆動回路はローライン(行)選択入力をデコードして特定の行をソース駆動するドライバ回路から成る。

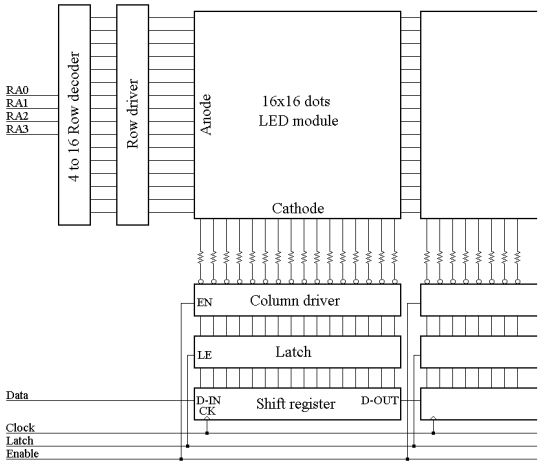
◆基本動作

ドットデータは一番下のシフトレジスタに順次送り込まれラッチに記憶され、そのデータのに対応した出力がON になる。RA[3:0]入力で特定の行を選択するとその行

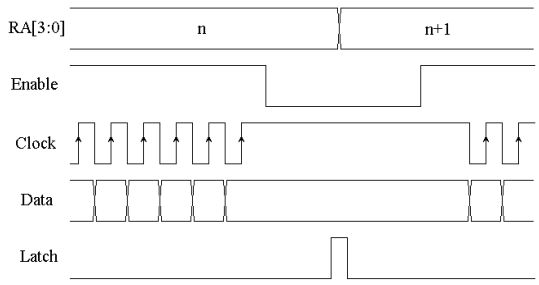
〈図 1〉 ドットマトリクス LED の内部構造



〈図 2〉 ドットマトリクス LED の駆動方法



〈図 3〉 コントロールタイミング



がラッチされたパターンで点灯する。そして、その行を点灯している間に次の行のデータを送り、それをラッチすると同時に RA[3:0]を次の行へ切りかえるという繰り返しによりそれぞれの行をスキャンしていくのである。

◆Enable 信号の必要性

Enable 信号によりカラムドライバの出力を一斉に OFF にすることができる。動作原理からは特に必要無いと思われがちだが、保安上結構重要な信号である。

コントローラが動作していないときは、同じ行が長時間点灯しつづける(多くの場合 LED の焼損につながる)ことがないよう、異常時はこの信号により全て OFF にして LED モジュールを保護するのである。また、ケーブル外れの場合も OFF に転ぶようにしなければならないのは言うまでもない。

行選択を切り換えたとき、行ドライバのスイッチングの遅れのため、ドライバの切り換えが完了するまで隣り合う行へ表示が漏れる。この過渡状態は行スキャンの周期からするとほんの一瞬で、低速スキャンなら特に問題になることはない。しかし、数 100fps 以上の高速スキャンではドットの漏れが目立ってくる。これを防ぐには、**図 3** に示すようにカラムドライバを一旦 OFF にしたあと、データラッチ・行切り換えを行い、ロードライバのスイッチングが完了してからカラムドライバを ON にするという動作が必要になる。

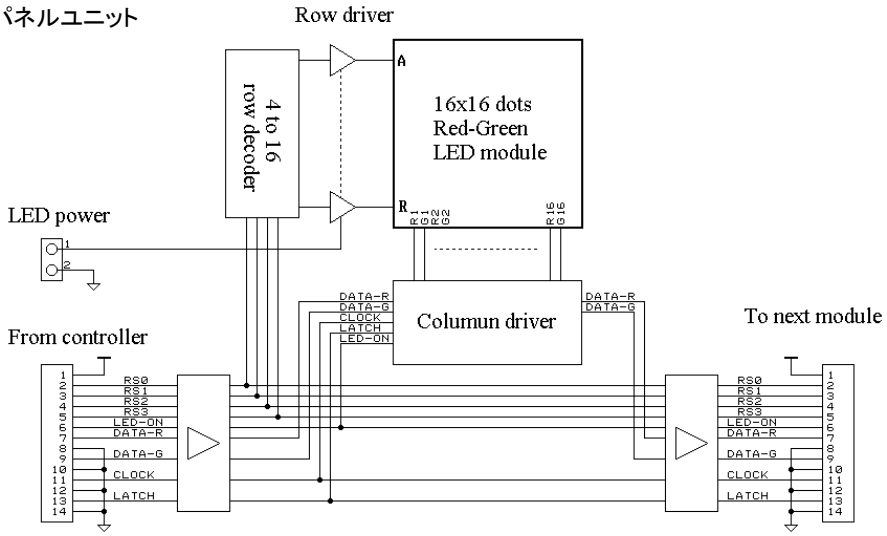
さらに Enable 信号を積極的に使い、調光機能を実現している例もある。これは行表示期間中の Enable 信号のアクティブ期間の比率を変えることにより行われる。

● 一般的なドライバ付き LED モジュール

ドライバ付き LED モジュールとして市販されている製品は大体**図 4** のような構成になっている。行選択がコード入力でなくパルスインクリメントのものも存在するが、特定の行を順次選択するという機能としてはどちらも同じである。もちろん、これ以外にもバッファメモリを内蔵するなど集積化を進めたものもある。

制御信号はモジュール内を抜けるようになっているので、複数のモジュールを簡単に数珠繋ぎすることができる。大電流が流れる LED 駆動電源は別コネクタでそれぞれのモジュールへ個別に配線される。

〈図 4〉 一般的な LED パネルユニットの構成



● 実際の設計例

図 5 に LED パネルの設計例を示す。これは私ならこうするといった程度のものであり、動作を保証するものではないことをお断りしておく。なお、行セレクタにはカラム駆動回路のドライバ IC を流用して、パルスインクリメント式としている。

◆カラム駆動回路

カラム駆動回路には LED メーカ各社から供給される専用のドライバ IC を使うのが常識となっている。この IC は、シフトレジスタ、ラッチ、LED ドライバの 3 つの機能が集積されているので、1 チップでカラム駆動回路を構成できる。さらにドライバ出力は定電流となっていて、1 本の外付け抵抗で出力電流を設定できるので、電流制限抵抗さえ不要である。

◆出力電流の設定

一般的に数 10mA 程度に設定される。1 個の LED に流す電流の DC 最大定格は一般的に 25～30mA であるが、ダイナミック点灯のため実際にはこれを超えて使われることが多い。そのため、LED モジュールにはパルス定格(パルス幅と繰返し周期により導かれる)というものも設定されている。DC 定格を越す場合は、同じ行が長時間点灯することがないように適切な保護機能を付けなければならない。

また、定電流ドライバの場合、LED 駆動電圧から LED の順方向電圧を引いた分をドライバで負担することになるので、ドライバの消費電力にも注意する必要がある。

◆ロードライバ

出力電流×カラム数の電流に耐えなければならない。損失は電流×ドライバの電圧降下となり、行毎の個別素子の場合は行数で分割される。

◆LED 駆動電圧

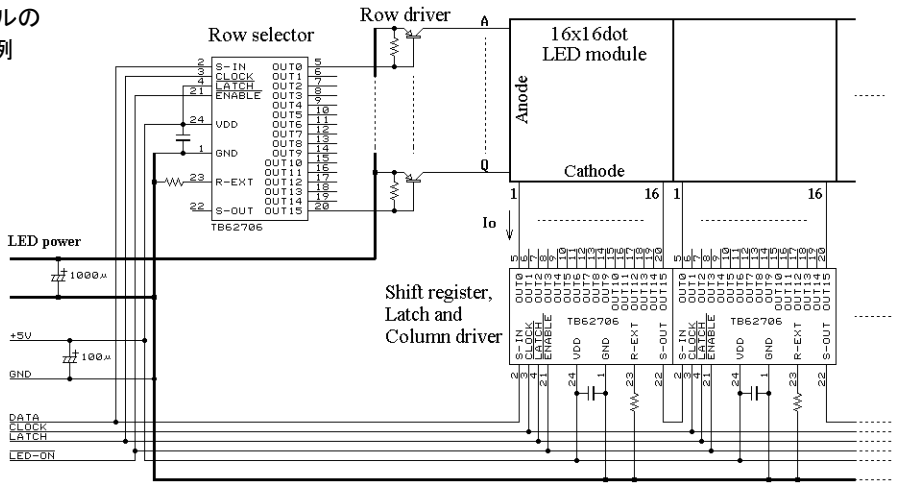
供給される LED 駆動電圧は、まず行ドライバと LED の電圧降下で消費され、そしてそれらを差し引いた残りをカラムドライバで負担することになる。カラムドライバの電

圧はそのままドライバでの損失に消えるため低いほど良いが、この電圧をあまり下げるとドライバの定電流回路が飽和してしまうので、1V 程度は必要である。このことから、必要な LED 駆動電圧は 3.5V 程度ということになる。

しかし、普通のドライバ付き LED モジュールでは入手が容易な 5V の電源に対応する例が目立つ。駆動電圧を5V に設定して、発熱の激しくなるカラムドライバにはヒートシンクを背負わせてしまうという乱暴な設計である。これはエネルギー効率の観点からあまり良くないし、装置全体の発熱も増えることになる。今後は省エネ意識の浸透にしたがい何らかの対策を考えたものが増えてくるだろう。

また、大型の LED パネルでは 1 ドットに 2～4 チップの LED 素子が直列になっているものがある。これについては直列数に応じて駆動電圧を高く設定する必要がある。

〈図 5〉 LED パネルの駆動回路例

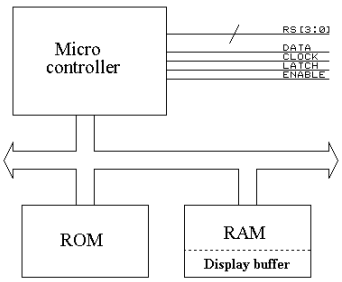


コントローラの構成

● I/O ポートで直接制御

図 6 のように汎用マイコンの I/O ポートに直結して制御する方式。回路が単純で安価である。ディスプレイバッファのデータを割り込み等で定期的に LED パネルに送って表示する。しかし、プログラムで表示データを転送するためマイコンの負荷が大きく、多数の LED パネルを制御するのは難しくなる。主に数枚以下の LED パネルの制御に用いられる。

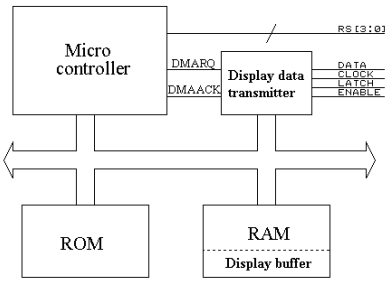
〈図 6〉 I/O ポートに直結



● DMA を使う

表示データの転送に DMA を使う方式(**図 7**)。巷で良く見かける LED パネルのコントローラは、ほとんどがこの方式だろう。外部に簡単なインターフェースを設けてマイコン内蔵の DMA コントローラによって表示データを LED パネルに送るので、表示動作の負荷もプログラム転送に比べてずっと軽くなる。

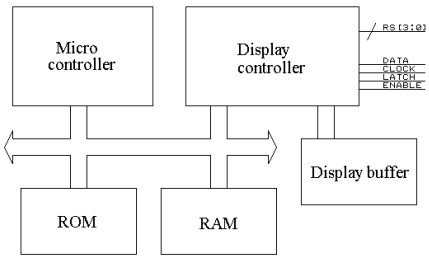
〈図 7〉 DMA 転送



● CRT コントローラを使う

さらに多連・並列接続して、数百ドット四方にもなる大画面の LED パネルを制御しようとなると、DMA でもデータ転送が追いつかなくなってくる。そのような場合は、**図 8** のように CRT インターフェースに良く見られるような独立したディスプレイバッファと制御回路を設けることになる。

〈図 8〉 独立したディスプレイコントローラ



スクロール表示のコツ

情報表示板などの LED パネルでは、必ずといってよいほど文字のスクロール表示が行われる。しかし、文字のスクロールを歪なくくっきりとした表示で行うにはそれなりの工夫も必要になってくる。

● 従来の垂直スキャン方式の問題

◆文字の歪み

CRT ディスプレイの延長として、または LED 駆動回路の原理にしたがって設計すると、**図 9** のように行を横一直線に配置して垂直方向にスキャンするのが自然である。実際、多くの LED 表示器がこの方式で設計されている。

静止画像だけ表示しているのならこれで問題ない。しかし、例えばこれを1ドット/フレームのスピードで文字を左へ向かって水平スクロールさせたとする。すると、**図 10** のように文字が歪んで見えてしまう。

これは、行スキャンの移動と視点の移動が直角に交わることにより起きる。視点の位置は1フレームのスキャンが終了する毎に隣のドットへ移るといった離散的ものではなく、一定速度の連続したものであるためだ。たとえば、あるカラムの最上部のドットが光ってから順次下へスキャンしていく間にも視点はどんどん左へ移動していく。スキャンがそのカラムの最下部に達するまでに約1フレーム分の時間差があり、その時は既に視点が隣のカラムへ移ってしまっているのである。このため、下へスキャンして行くに従ってそのカラムのドットは右のほうへずれていき、最下部では約カラム分ずれてみえることになる。これが文字の歪みの起こる原理である。

◆文字のブレ

歪みを軽減するためには、フレーム周波数を上げてスクロール速度に対してスキャン速度を十分に速くしてやればよい。1ドットスクロールする間に複数回フレームスキャンするのである。1ドット/4フレーム程度にすれば歪みはほとんど目立たなくなる。しかし、1ドット分スクロールする間に同じドットが複数回点灯することになるため、**図 11** のように文字がブレて見える。こんどは文字のシャープさが失われて、ボケた文字に見えるという弊害が発生してしまうのである。

このブレを消すため、スキャン速度は高速のままにしてスクロールした直後のフレームだけ点灯してあとのフレームは点灯しないという方法がある。しかし、このようにするとこんどは輝度が極端に落ちてしまい、実用的ではない。

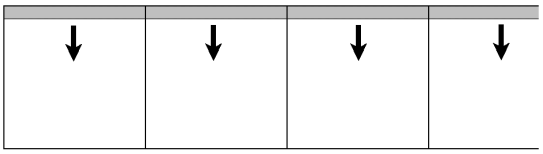
この歪みの問題は垂直スキャン方式の宿命であり、クリアする方法は無い。実際のシステムでは、文字のボケに目をつぶって高速スキャンで歪を誤魔化しているものが多い。また、ある地下鉄車内の表示器では、1ドット/フレームのまま派手に歪んだ状態で使われていたものもあった。

● 優れた水平スキャン方式

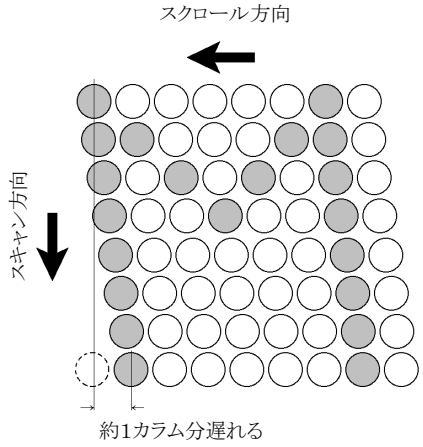
垂直スキャン方式の諸問題をクリアするため、LED パネルの配置に工夫をした水平スキャン方式が採用される例が多くなってきた（**図 12**）。これは垂直スキャン方式の LED モジュールを単にそれぞれ左に 90° 回転させたものである。これによりスキャン方向と始点の移動方向が交わることはなくなり、晴れて文字の歪は解消する....というほど話は単純でもない。

水平スキャン方式においてもドット点灯の時間差の影響は避けられず、スクロールした場合はそれぞれの LED モジュールの境界において隣接する行がが重なって見えることになる。そのまま普通にスクロール処理すれば、それぞれの LED モジュールの境界で文字がつぶれたように見えてしまう(**図 13**)。スクロールする文字を追っていくとちょうど芋虫が這うように後ろにずれていくといった感じだ。

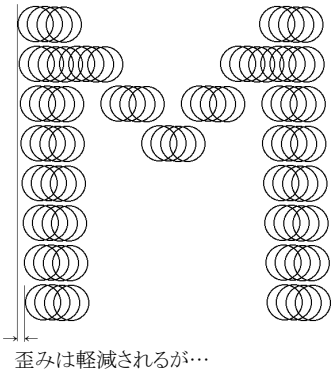
<図 9>垂直スキャン方式



<図 10> 文字の歪み(1 ドット/フレーム)



<図 11> 文字のブレ(1 ドット/4 フレーム)



これを回避するには、ソフトウェアに少し工夫して**図 14** に示すようにモジュール間で隣り合う(重なって見える)行は一つであるとしてスクロール処理すればよい。これによりはじめて歪なくかつシャープな表示を実現できることになる。

◆分散スクロール

1ドット/フレームのスクロールは鮮明な表示が可能であるが、それではスクロールが速すぎる場合がある。フレーム周波数を下げるにしても 60Hz 程度が限界なので、それより下げる場合は1ドット/nフレームとしてスクロール速度を落とすしかない。

このときスクロール処理はnフレーム毎に一斉といった方法ではダメだ。こんどは逆にLEDモジュール間で伸びて見えてしまう。このため、**図 15a** のスクロール処理を**図 15b～c** に示すように1フレーム毎に各LEDモジュールに分散して適用する必要がある。

◆描画処理でのメリット

実は水平スキャン方式のメリットはこれだけではない。ソフトウェアから見て表示バッファの構成がスクロールや描画処理に大変適したものであるということだ。水平スキャン方式ならスクロール処理はバイト単位のコピーだけで済むし、同様に部分スクロール機能や可変ピッチ文字の実装も容易である。

従来の垂直スキャン方式ではスクロールする際にビット単位のシフト処理が必要になり、さらに部分スクロールや部分描画では境界のビットマスク処理が必要になってくるなど描画処理でいろいろ面倒である。

◆複雑なスクロールでの問題

情報表示板で逆スクロール(左から右へ)することなどまずないと思うが、その場合は時間差の影響が逆に現われ、モジュール間で文字が伸びて見えてしまうので、表示バッファ上でモジュール間に見えないカラムを挟むなどの処理をするしかない。また、スキャン方向を逆にするという手もある(ハードウェアで決め打ちの場合はお手上げだが)。

さらに部分スクロールや途中停止・開始など複雑な表示になるとどうにも対処のしようがない。これらについては高速スキャンで誤魔化すしかなくなってくる。

実際にやってみる

さて、ドットマトリクス LED の基本的なところが理解できたところで実際に表示器を自作ってみよう。

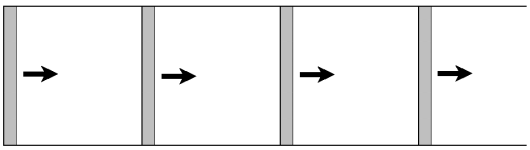
ちょっと実験してみる程度ならジャンク屋で売られているドライバ付きの LED モジュールが適当だろう。単独のLEDモジュールの手持ちが余っているとか駆動回路から自作ってみたいという場合は、最初から設計ということになる。スキルアップのためには最初からやってみるのもいいかもしれない。

● カードサイズ LED 表示器

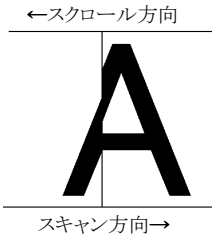
図 16 に製作した LED 表示器の回路図、**写真 1** に外観を示す。この例では小型化のため単独の LED モジュールを使っている。今回も小型化に挑戦ということで、厚さ 10mm を目指したが及ばず、記録は 10.5mm に終わった。

カラム駆動回路には TB62705 を使用している。

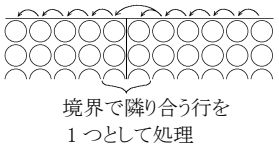
〈図 12〉 水平スキャン方式



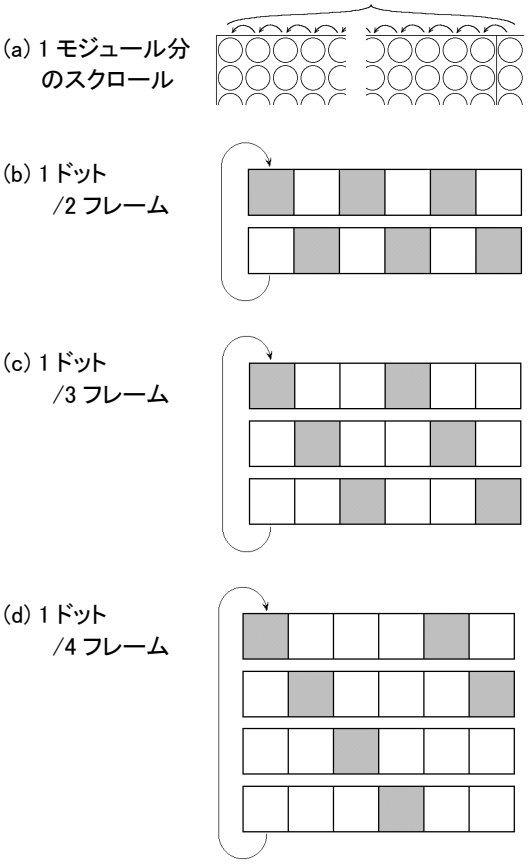
〈図 13〉 文字のつぶれ



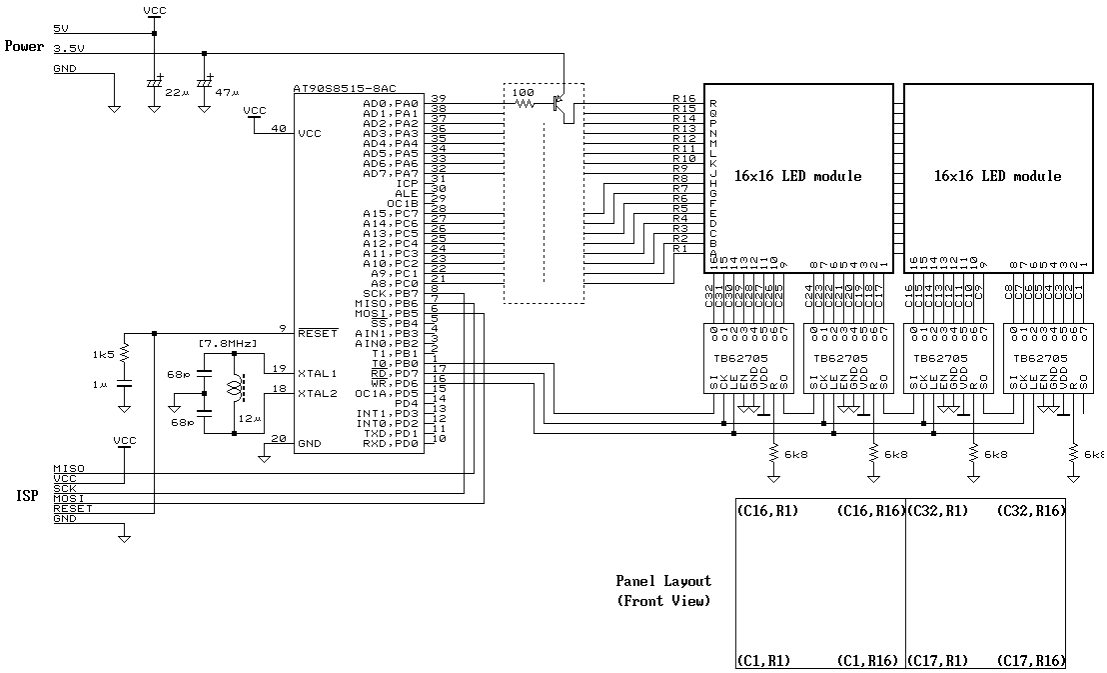
〈図 14〉 スクロール処理



〈図 15〉 分散スクロール(6 個連結の場合)



〈図 16〉 カードサイズ LED 表示器の回路



TB62705 は 8 ビットシフトレジスタ＋ラッチ＋定電流ドライバが 1 チップに集積されたドットマトリクス LED ドライバである。このような専用ドライバの入手が難いようなら 74HC595＋トランジスタアレーという構成が手軽で良いだろう。

行駆動回路はデコーダを使わずにそれぞれの行をポートで直接駆動している。このため、カラムドライバの ENABLE 機能は使う必要が無い。

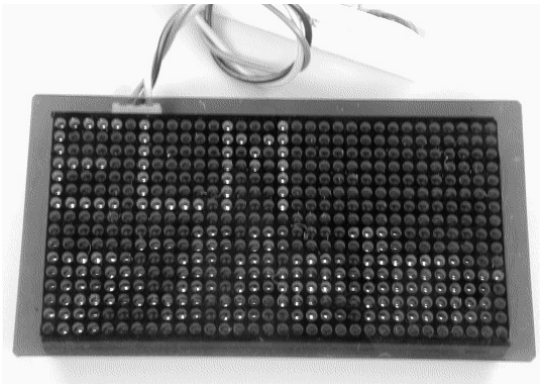
モジュールのレイアウトは回路図の右下に示すように水平スキャンとなるようにしてある。

最後に

とりあえず基礎的なポイントだけまとめてはみたが、実際のシステムを開発するにはこれだけではまだ不十分で、もっと多くのノウハウが必要になってくるだろう。

でも、この記事によって身近で活躍する電子機器のテクノロジーに興味を持ってもらえれば、また、研究にあたってわずかでも参考になれば幸いである。

〈写真 1〉 製作したカードサイズ LED 表示器



【追記】

現在、(株)若松通商よりドットマトリクス LED の実験キットが販売されている(私のプロデュースだけど(^^;)。

これは、96×16 ドットの単色 LED パネル(完成品)を AVR マイコンを使ったコントローラ(要半田付け)で制御するというもので、実用性もあるキットである。手軽に実験できるので、ドットマトリクス LED に興味を持たれたのなら試してみてはどうだろうか。

MAS3507D MP3オーディオデコーダを使用した

MPROMの製作

もろぼし☆らむ **elfin@mth.biglobe.ne.jp**
http://www2s.biglobe.ne.jp/~elfin/

はじめに

MPROM とは、MPEG1-Layer3(MP3)形式で記録されたファイルの格納された CD-ROM/CD-R/RW ディスクから、直接そのファイルを読み出して再生するオーディオ機器の通称です。

昨今、高圧縮率、高音質のオーディオデータ圧縮フォーマットであり、規格化された圧縮方式である MP3 形式は、半導体メモリに記録するデータ形式として最適なものとして多くの携帯オーディオ機器で採用されています。この MP3 ファイルを大容量の CD-ROM ディスクに大量に記録し、長時間再生可能な音楽メディアとしたのが、この MPROM となります。

とはいえ、この MP3 データのデコードは計算量が膨大になるため、単純なマイコンでこれを行わせるにはかなりの高速な計算能力と周辺メモリが必要となります。しかし、デコードを専用に行う LSI が開発されたことにより、これを使うことで低能力のシステムでも MP3 オーディオシステムを作ることができるようになりました。

今回は、このようなデコード LSI で初期に登場し、特に秋葉原で簡易に入手できる Micronas 社の MAS3507D を使って、MPROM システムの試作を行ってみます。取り敢えず、解説は今回のシステム自体の話よりも、一般的な MP3 オーディオの実現のノウハウという観点で進めていこうと思います。

基本仕様

●CD-ROMディスクの読み出し

CD-ROM ディスクは、SONY と Philips 社が RedBook として規格化し、国際規格の IEC908 となったオーディオ CD をデータが記録できるようにセクタの定義を行い、ECC を付加することでその信頼性を上げたものです。さらにデータを格納する場合、実際のデータがどこにあるかということを認識するために、論理フォーマットが必要となります。普通、FD や HDD などでは FAT や NTFS というものが採用されていますが、CD-ROM ディスクでは多くの OS で互換性があることに重点を置き、読み出しのみのメディアという特性を生かした ISO9660 という専用のフォーマットが規定されています。

しかし ISO9660 では、OS 間の互換性を取ることに重点が置かれているため、厳しいファイル名に使用できる文字種の制限とファイル・ディレクトリ名の長さの制限が存在します。これにより、普通の ISO9660 ディスクでは漢字のファイル名などを使用することができません。そこで、Microsoft 社の Windows ではこの問題を解消するため、ISO9660 と互換性を保ちつつ、このような制限を取り除いた Joliet というファイルシステムを開発し、採用しています。

今回は ISO9660 形式と、さらに Joliet 形式をサポートすることで、CD-R ライタで作ることのできる一般的な CD-ROM ディスクに書かれた MP3 ファイルを読めるようにします。

●MP3オーディオ再生

前述の CD-ROM 読み出しの機能とともに、重要なのが MP3 オーディオの再生機能です。この再生は比較的簡単で、単純に MP3 オーディオデータを MP3 デコーダ LSI に対して同期シリアル方式で転送を行うものです。ただし、このデータ転送速度は一般に MP3 オーディオでは最高 320KBPS に達するので、CD-ROM 読み出しの時間も含めて、この転送速度に十分に追い付く程度の能力が送信側に求められます。

オーディオ再生では、この他にどのようなデータを転送するかというデコードデバイスに対する初期設定、またはオーディオボリュームの設定などの基本的な設定を行います。

●その他の機能

上記の基本機能の他に次のような MP3 オーディオとしての機能を実装します。

大画面の表示装置

MP3 オーディオファイルにはそのファイルのフォーマット情報の他に、ID3 タグという曲名情報を付加することができますので、これを表示する必要があります。また、CD-ROM ディスクにあるファイルのリストを表示し、それらの中から演奏するデータを選択したりするというようなファイラーの表示を行うため、大画面の液晶表示を付加します。

プログラム再生

ファイラーから再生したい MP3 ファイルを複数選択し、その選択したファイルを連続的に演奏するような簡単なプログラム演奏(プレイリスト再生)を行えるようにします。

CD-DA再生

MP3 オーディオファイルの他に、一般のオーディオトラックの再生をサポートします。

設定の記録と取得

MP3 オーディオファイル再生順序、または各種設定(オーディオボリューム)の保存を行います。

基本設計

●ハードウエアの概要

ハードウェアブロック図を図 1 に示します。ハードウェアは大きく分けて３つのブロックにわかれます。一つが CPU、メモリ、CPLD を中心とする基本制御ブロック、さらに MP3 データをデコードし、D/A 変換をして増幅を行うアナログブロック、もう一つが ATAPI-CDROM デバイスのインターフェイスブロックです。各ブロックに相当する回路図１～４を示します。

基本制御ブロック(回路図1,2)

この部分は、プログラムの実行を行うための基本的なブロックで、フラッシュ ROM、RAM でプログラムの実行を、さらに全体を制御する CPLD により、デバイスのマッピングと LCD や MP3 データストリーム転送に関する DMA タイミング制御を行います。
この他に、システムを操作するためのキーや、そのキー入力のためのパラレル入力部分、実際に画面を表示する LCD パネルのインターフェイスもあります。

アナログブロック(回路図3)

この部分で、MP3 データストリームをデコードして、デジタルオーディオデータに変換します。さらに、変換されたオーディオデータをアナログ信号に変換し、増幅を行います。また、CD-DA 再生時にドライブから送られるアナログオーディオ信号とのミキシングもここで行われます。

ATAPIインターフェイスブロック(回路図4)

この部分で、ATAPI-CDROM の IDE インターフェイスと本システムとの接続をするためのバス変換を行います。実際には IDE インターフェイスは単純な TTL レベルのバスですので、基本的にはレベル変換とバスの整合を行っているだけです。

●ソフトウエアの概要

MP3 データ再生やステータス表示などは、すべて割り込みを使って制御され、プログラムの構造は、大まかに以下の 3 つの部分で構成されます。

MP3データ送信

MP3 データ転送での割り込みは DMA がデータバッファにあるデータを全部送信し終わった時点で入ります。このときの割り込みで、次に送信するバッファを設定します。これを全ての演奏データを送り終えるまで、何度も繰り返します。
このように MP3 データ転送部分は割り込みルーチンの中で完結しており、全体の動作と分離して動作しますので、メインプログラム側はデータストリームの転送を気にする必要はほとんどありません。勿論、MP3 データの CD-ROM からの読み出しも、前述の転送完了割り込みルーチンの中で行われます。

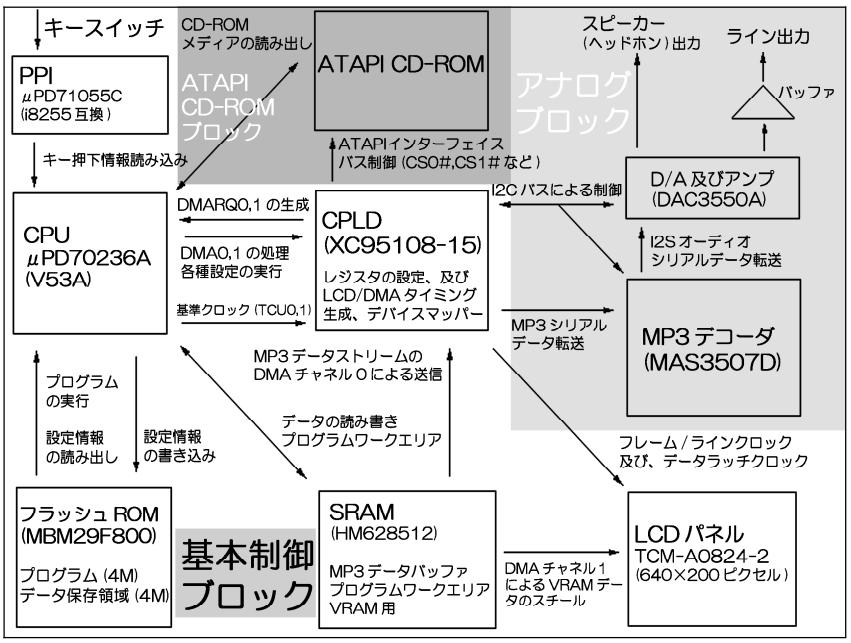
ファイラー

ファイラーのシステムは、MS-DOS で一般的なファイラープログラムとほぼ同様で、１ディレクトリを基準にしてそのディレクトリ内のファイルを表示し、カーソルキーで選択カーソル移動、実行キーでディレクトリならその階層に移動、データならばそれを実行というような感じになっています。

MP3プレイヤー

MP3 プレイヤーのメインコードは、基本的にはプレイリストの内容をみて、演奏するファイルを決めた後、そのファイルを解析して種類を特定し、適切に読み出し開始位置と演奏データサイズを決定して演奏を開始します。それをプレイリストで登録されているファイルがなくなるまで繰り返すことになります。

図1
MPROM/ハードウエアブロック図



主要部品

CPU
V53A（NEC）12.288MHz
SRAM
HM628512（Hitachi）
512KByte（4MBit/8Bitbus）
FLashROM
MBM29F800BA
1MByte（8MBit/16bit bus）
Gate
XC95108-15
DSP
Mas3507D（Micronas）
D/A,AMP
Dac3550A（Micronas）

基本的なデバイスの設計

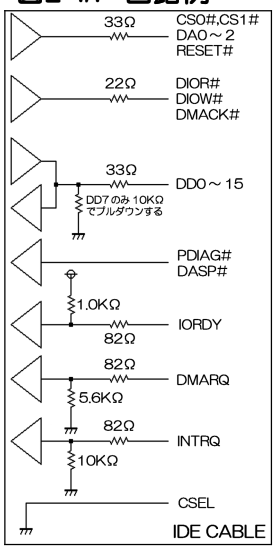
●CD-ROMドライブの制御

今回使用した CDROM ドライブはインターフェイスが単純な IDE インターフェイスで接続される、ATAPI-CDROM デバイスです。ATAPI デバイスは、ATA デバイスの延長上にあるもので、デバイスに対して SCSI のコマンドパケットに似た動作指定パケットを送ることで動作するものです。また、AT 互換機では標準インターフェイスとして HDD に IDE を使用していることから、CD-ROM を接続する上での標準的なインターフェイスとなっています。今回は、インターフェイスが単純で制御が簡単であることから、これを使用しています。

インターフェイス設計

IDE インターフェイスの一般的な回路は、図 2 のようになっています。元々、IDE インターフェイスは、昔はハードディスクコントローラのインターフェイスと同様であったため、信号線自身にアドレス信号やチップセレクト信号などがあります。そして、ハードディスク内部にレジスタがあり、この中のレジスタにアドレス、チップセレクト信号を使ってアクセスするような形となっています。バスレベルは普通の 5.0V TTL レベルのバスです。システム内部は CMOS レベルで動作していますが、バスタイミングはほぼ 86 系バスと同じですので、単にバスは 74HCT245 を使って TTL-CMOS レベル変換を行って接続します。バス方向設定(DIR)は CPU の R/W 信号を、バス接続信号(G)は CPLD で生成したセレクト信号(ATAPICS)を使います。どちらの信号も CPU バスサイクルの T1 サイクルで確定しますので、実際のアクセスまでは十分に余裕のあるタイミングとなっています。[*1]

図2 I/F 回路例



しかし、ATA インターフェイスは、簡単に言えばそのまま 86 系のバスが 40 線のフラットケーブルでハードディスクコントローラに繋がっている形ですので、この接続には十分に注意を払う必要があります。特に信号のクロストークや反射が発生しやすいので、ケーブルの接続やインターフェイス回路の設計などには十分注意する必要があります。また、各バスの信号線の抵抗は回路保護と終端のためだけでなく、前述のクロストークなどを防止するダンピング抵抗としての役割もありますので規定値の抵抗を必ず入れます[*2]。

なお、アクセスタイミングは PIO モード 4 では図 3 のようになっています。なお、PIO モード 4 ではウェイト信号が必須となりますので、設計上必ず考慮する必要があります。

ATAデバイスの制御

単純な ATA デバイスでは、制御レジスタに対して必要なパラメータをセットして実行を行います。それらのレジスタはそのままハードディスクをアクセスするようなシリンダ・ヘッド・セクタパラメータとなっています。本システムでは、これらのレジスタはワードアドレスにマップされており、表 4 のようになっています。

これらのレジスタのパラメータの読み書きもワードアクセスで行います。一般に ATA インターフェイスのバスは 16 ビットですが、このようなレジスタへのアクセスでは、データレジスタ以外はデータバス下位 8 ビットのみが有効になりますので、デバイスのマッピングには注意する必要があります。

ATAPI パケットコマンドの実行は、コマンドレジスタにパラメータを書き込んだときに開始されます。その後に 400ns 待ったあと、ステータスをみて BUSY ビットや ERROR ビットを監視します。ERROR も BUSY もなくなったら、データの読み込み、または書き込みを開始します(図４)。

図3 PIOモード4アクセスタイミング

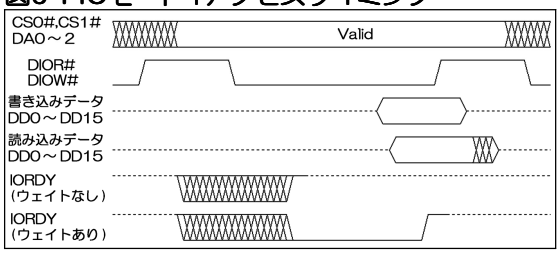
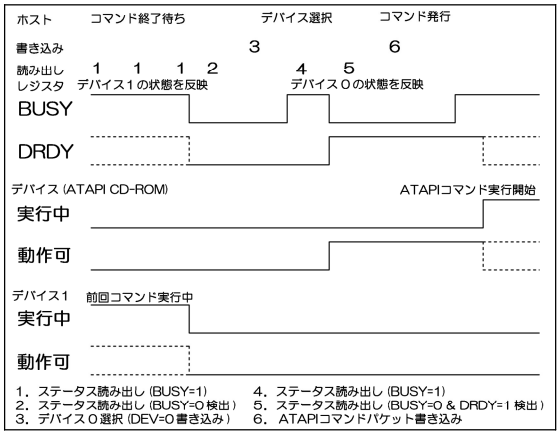


図4 ATAPIコマンド発行シーケンス



---

*1CPLD のタイミング解析では遅延が最大でも 30ns 程度なので、データをラッチする T3 サイクルまで 12.28MHz 動作時では 90ns ですから、ラッチ信号(IOWR/IORD)到達まで 60ns 程度余裕がありますので、HCT245 で十分対応可能な時間です。

*2 よく ATA 用ケーブルでスマートケーブルというのがありますが、このようなケーブルはクロストークを助長するので、使用するとシステムの安定性を損なう可能性があります。ATA66 用 80 ピンケーブルできちんと信号がツイストペアになっていれば、問題はないのですが、そこまで考えて作っているとは思えない・・(笑)。

ATAPIデバイスの制御

ATAPI デバイスは前述のように ATA デバイスの延長上にあるもので、最初の動作は基本的には ATA デバイスと同様です。異なるのは、コマンドレジスタに ATAPI パケットコマンドをセットし、最初のデータ転送(図 4 の 6)にパケットを送信することです。このパケットを認識して、ターゲットとなるデバイスはパケットで決められている所定の動作を行います。

この動作は、基本的に割り込み信号 INTRQ を使って制御します。一応、割り込みを使わなくてもある程度は操作できますが、データの転送であまり多くをやりとりできなかったり不安定になりますので、割り込みを使うのが一般的です。基本的な動作パターン(Read コマンド)を図 5 に示します。

●MP3デバイス制御

今回使用した MP3 デコード用のデバイスは、デコード用 DSP の MAS3507D と D/A コンバータとパワーアンプを内蔵した DAC3550A です。これを 2 つ組み合わせて使うことで、コンパクトな MP3 オーディオシステムを構築することができます。

I2Cバスによるデバイス制御

MAS3507D と DAC3550A のデバイスとしての制御は、マイコンで一般的な I2C バス(Inter Integrated Circuit)を使って行われます。I2C バスは同期シリアルバスの一種で、SCL（クロック）と SDA（データ）の２線で制御される 100kpbs ～ 3.4MBps 程度の転送速度をもつバスです。単純なバスプロトコルであるため簡単に制御可能で、かつそのバス上にデバイスを自由に追加・削除ができるようになっています。

I2Cバスのアーキテクチャ

I2C バスはハードウェア的には SCL、SDA 共にプルアップされた信号線です。その信号線に各デバイスの入力が接続され、さらに出力がオープンドレイン、またはオープンコレクタで接続されています。これにより、複数のデバイスが接続可能となっています。そして、各デバイスはマスターとスレーブがあり、スレーブには 7 ビットのアドレスが振られています。マスターはクロックを操作しつつ、アドレスを指定して目的のスレーブのデバイスを操作することになります。

I2Cバスの基本的な制御方法

I2C バスプロトコルのタイミングを図 6 に示します。このなかで重要なのが、スタートコンディションとストップコンディションという状態です。これは、それぞれマスターデバイスからのデータ通信開始信号、終了信号で、前者が SCL が H の状態のときに SDA を L に落とすことで、後者が逆に H に上げるものです。I2C バスプロトコルでは SCL が H 状態のときに SDA を動かしてはいけないという前提がありますので、このような操作でスレーブは通信の開始、終了を確実に知ることができます。データの送信・受信は SCL の立ち上がりエッジで取り込むと考えればよいでしょう。

データを送信する場合、スタートコンディションのあとにターゲットとなるスレーブデバイスのアドレス７ビットと R/W ビットの合計８ビットを送った後に ACK/NAK のリザルトをスレーブから受信します。このあと、データを送りリザルトに NAK が返るまで繰り返します。受信をする場合には逆に、アドレスを送ったあとに 8 ビットのデータを受信したあとに ACK を送り、受信し終わった時点で NAK を返します。以上の動作を本システムではすべてソフトウェアで制御しています。

図5
ATAPI デバイスアクセスタイミング

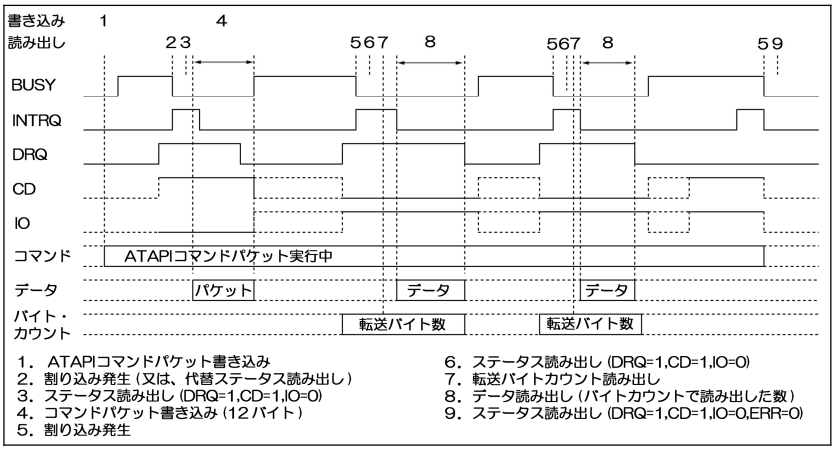
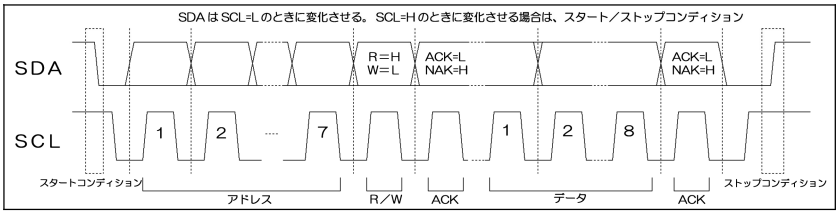


図6
I2Cバスアクセスタイミング



MAS3507DとDAC3550Aの制御

それぞれの I2C 制御アドレスと主なコマンドを表 1,2 に示します。なお、DAC3550A の場合、アドレスは MCS0,1 の端子で Bit6,7 の 2 ビットを自由に設定できますので、同一のバスの上に 4 つの同じデバイスを乗せることができます。今回、MAS3507D は 32 ビットの SDO モードでオーディオデータを転送させますので(図 7)、DAC3550A の SR_REG の LR_SEL ビットを設定する以外は特に必要な設定はありません。この他に DSP の初期設定を Layer2,3 再生許可、マルチメディアモード、基準クロック 14.725MHz、オーディオ出力 SDO 32bit、CLKO 分周としますので、PI0 ～ 4、8 を全てプルダウン(0)しておきます。

本回路上では、DSP と DAC は 3.3V 動作していますが、システムは 5.0V で動作しますので、レベル変換を行う必要があります。そのため、出力にはトランジスタを使ってオープンコレクタで接続し、入力は 74HC14 を使います。なお、トランジスタの ON/OFF は 5.0V レベルとなっていますので、ある程度大きなベース抵抗を入れておいてベースの電圧とコレクタの電圧が逆転しないようにしておきます。

MP3データストリーム転送

MP3 データを MAS3507D に送信する場合、SIC にデータクロック、及び SID にクロックに同期したデータを入力します(図 8)。MAS3507D を PIO ピンでマルチメディアモードに設定する場合、データのフロー制御は DEMAND ピンで行われます。このピンが H レベルのときにデータ入力可能となりますので、このときだけ MP3 データをデータクロックに乗せて送ります。また、MAS3705D にはブロードキャストモードというものがあります。これは MP3 データのフロー制御をコントローラができないとき[*1] に使用されますので、MP3 オーディオ機器ではこちらは使われません。

CPLDのロジック設計

CPLD は再プログラム可能なデバイスで、手軽に複雑な回路を回路記述言語を使用して集積することができます。今回は、この言語に ABEL を用いて設計しており、大まかには CPLD には以下の 3 つの機能を入れています。

デバイスのマッピング

デバイスのマッピングは、特定の I/O ポートやメモリのアドレスにアクセスがあったときに、適切なデバイスにチップ選択信号を出してアクセスを許可するものです。今回のシステムでの、I/O ポート、及びメモリのマッピングは表 3 のようになっています。

通常はアドレスとメモリ・I/O 選択信号によってデバイスを 1 つだけ選択しますが、DMA 転送時のみはメモリと I/O が同時に選択されますので、このときだけは指定された DMA 転送で使われるデバイス 2 つを同時に選択するという特別な動作をします。

各種制御レジスタ

上記のマッピングの他に、ATAPI インターフェイスや I2C バスの信号線の制御を行ったり、その他の CPLD のピンの状態の設定や取得を行う必要もあります。この値で必要なものは CPLD 内部のラッチに保存され、その特定の I/O ポートが選択されたときに、その値を書き換えたり、読み出せるようにします。

DMA転送制御

2 つの基本的な機能の他に、このシステムの核である LCD の VRAM スチール処理や各種クロックの生成、及び MP3 データの DMA 転送の際のタイミング生成、パラレルーシリアル変換転送がありますが、これらの詳細な説明は後の項目で解説します。

表1　MAS3507Dのレジスタ(抜粋)

アドレス	名前	詳細
$8E	DCCF	DC-DC コンバータモード設定
$AA	Mute/Bypass ToneControl	デジタル出力のミュート設定
$C8	PIO Data	PIO ピンの読み出し
$E6	Startup Config	スタートアップ設定のマスク
$E7	KPrescale	トーンフィルタのプリスケーラ設定
$6B	KBass	低周波数領域の増幅設定
$6F	KTreble	高周波数領域の増幅設定

※ Mas3507D には、他にも内部メモリ設定などがありますが詳細はデータシートを読んでください。

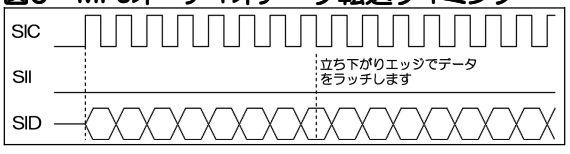
表2 DAC3550Aのレジスタ

アドレス	名前	詳細
$01	SR_REG	サンプリングレートコントロール、及び入力設定。
$02	AVOL	ボリューム、及びミュート制御
$03	GCFG	一般の設定。及び外部入力の許可設定。

※ MCS0=0,MCS1=0 のとき

図7 32bitオーディオデータ転送タイミング

図8　MP3オーディオデータ転送タイミング



*1 例えば、ネット配信されるデータを直接受けて再生する場合は、サーバーの都合でデータの送受信が開始されたり停止なされますので、フロー制御ができません。

表3 CPLDデバイス・レジスタマッピング

CPLDのデバイスのマッピング

I/O アドレス	入出力	デバイス
100 ～ 2FFh	I/O	ATAPI CD-ROM
300h	I/O	制御レジスタ 0(GENERAL)
400h	I	制御レジスタ 1(GENERAL)
500h	O	MP3 データストリーム転送用シフトレジスタ
600h	I/O	制御レジスタ 2(LCD)
700h	I/O	PPI(μ PD71055C)

CPLD制御レジスタ0

Bit	名称	機能
7	I2CCOUT	1 のとき I2C SCL を出力にする
6	I2CDOUT	1 のとき I2C SDA を出力にする
5	DSPEN	MAS3507D の DSPEN 端子設定
4	RESETDSP	DSP と D/A のリセット端子
3	RESETCD	ATAPI I/F の RESET#信号設定
2	SDMASK	1 のとき SDAT の出力許可
1	I2C_COM	I2C SCL への出力データ
0	I2C_DATA	I2C SDA への出力データ

CPLD制御レジスタ1(入力専用)

Bit	名称	機能
7	SREGEMP	MP3 シフトレジスタが空のときに 1
6	DEMAND	Mas3507D の DEMAND 端子状態
5	-	-
4	-	-
3	INTRQ	ATAPI I/F の INTRQ 信号状態
2	PDIAG	ATAPI I/F の PDIAG 信号状態
1	I2C_COM	I2C SCL の状態(入力)
0	I2C_DATA	I2C SDA の状態(入力)

CPLD制御レジスタ2

Bit	名称	機能
7	DOFF2	LCD パネルの DOFF2 端子設定
6	DOFF1	LCD パネルの DOFF1 端子設定
5	-	-
4	-	-
3	-	-
2	-	-
1	STEALON	LCD データスチール開始
0	LCDON	LCD タイミング信号 YD/LP/XSCL 出力開始

表4 ATAPIインターフェイスのレジスタのマッピング

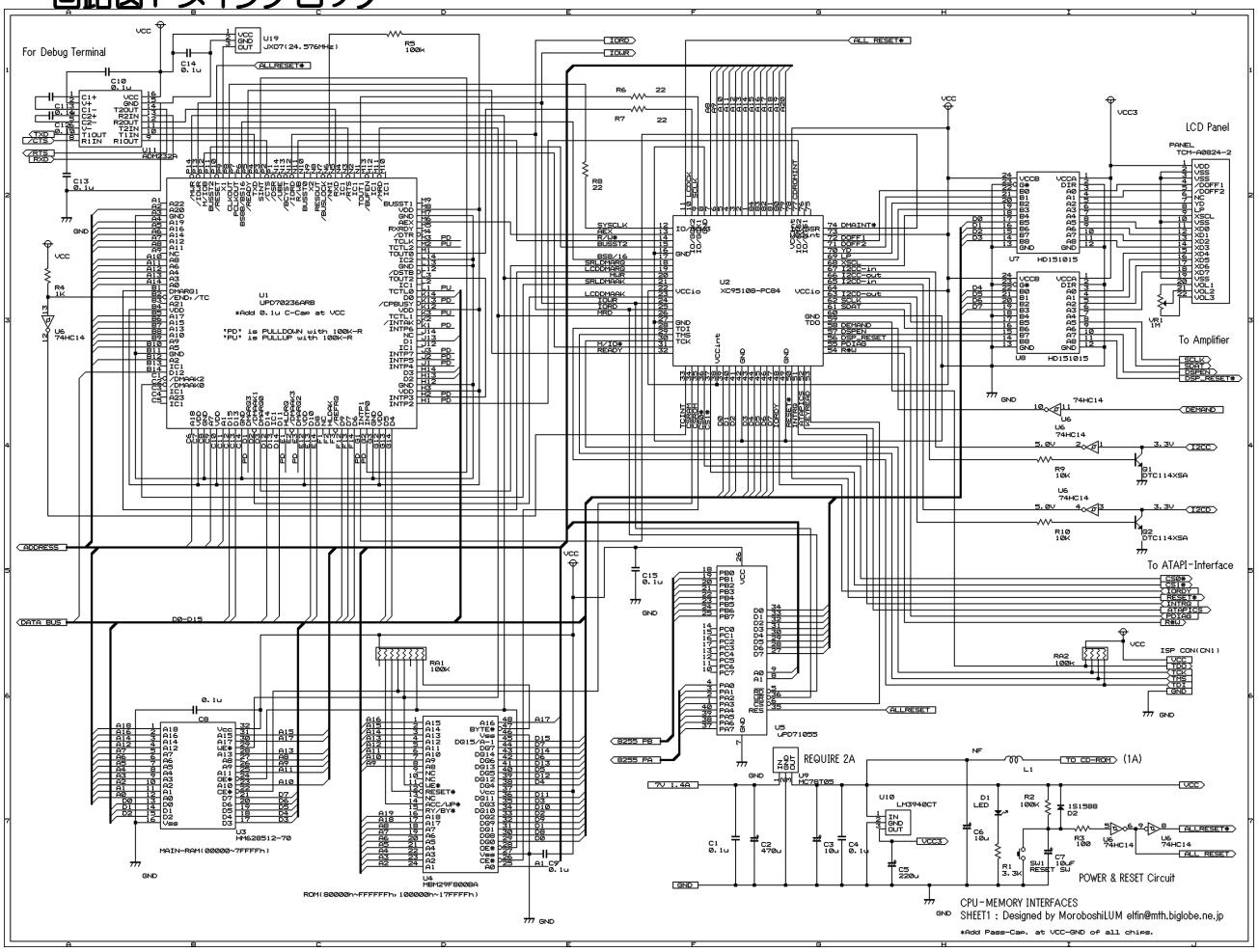
アドレス	ATAPI アドレス DA2 ～ DA0			ATAPI CSx CS1#	ATAPI CSx CS0#	レジスタ名(ソースコード内表記)	入出力	レジスタ名
100h	0	0	0	H	L	ATA_DATA	I/O	Data レジスタ(16bit)
102h	0	0	1	H	L	ATA_ERR	I	Error レジスタ
102h	0	0	1	H	L	ATA_FERT	O	Features レジスタ
104h	0	1	0	H	L	ATA_INTR	I	Interrupt Reason レジスタ(ATAPI)
104h	0	1	0	H	L	ATA_SECC	I/O	Sector Count レジスタ(ATA)
106h	0	1	1	H	L	ATA_SECN	I/O	Sector Number レジスタ
108h	1	0	0	H	L	ATA_CYCL	I/O	Cylinder Low レジスタ
108h	1	0	0	H	L	ATA_BYCL	I	Byte Count LSB レジスタ
10Ah	1	0	1	H	L	ATA_CYCH	I/O	Cylinder High レジスタ
10Ah	1	0	1	H	L	ATA_BYCH	I	Byte Count High レジスタ
10Ch	1	1	0	H	L	ATA_DVHD	I/O	Device/Head レジスタ
10Eh	1	1	1	H	L	ATA_CMD	O	Command レジスタ
10Eh	1	1	1	H	L	ATA_STAT	I	Status レジスタ
20Ch	1	1	0	L	H	ATA_DVCL	O	Device Control レジスタ
20Ch	1	1	0	L	H	ATA_ASTAT	I	Alternate Status レジスタ

<デバイスマッピング部分の CPLD Source(ABEL 7.0)>

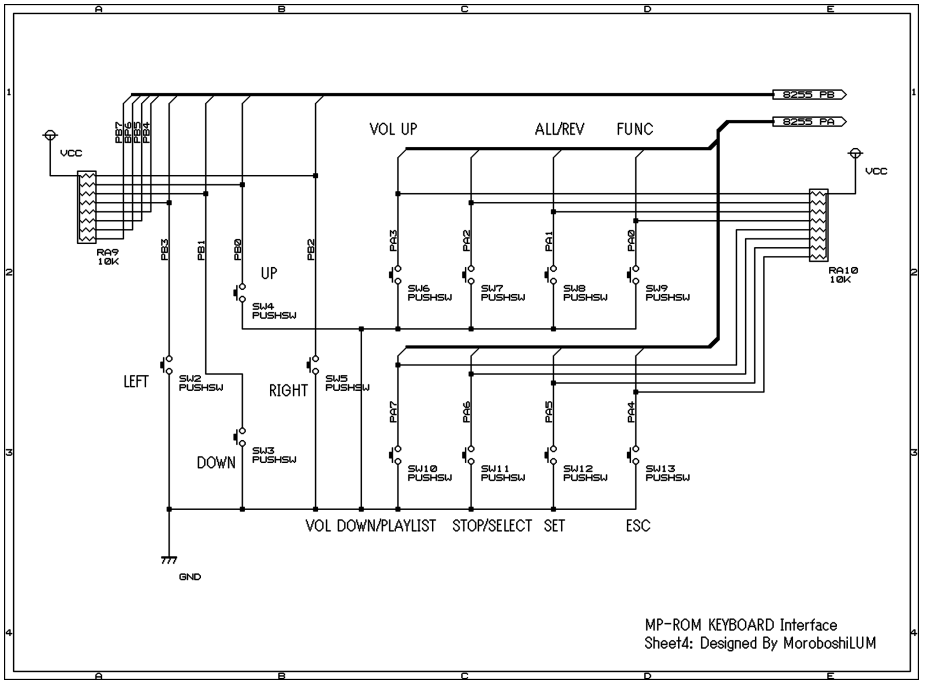
```
" DEVICE MAPPER
!CSRAM = ((HiAddress < ^h8) & !(A20 & AEX)) & ((MIO & !BUSST)  "SRAM(一般アクセス)
         # ((!I2SDMAAK # !LCDDMAAK) & BUSST));                 "SRAM(DMA アクセス)
!CSROM = ((HiAddress >= ^h8) # (A20 & AEX)) & MIO & !BUSST;    "Flash ROM
!KEYREAD= (Address == ^h7) & !MIO & !BUSST;                    "uPD70115C(PPI)
ATAPICSS = ((Address == ^h1) # (Address == ^h2)) & !MIO & !BUSST;        "ATAPI CD-ROM I/F
!I2SREGCK = (Address == ^h3) & !MIO & !BUSST;                  "制御レジスタ 0
!I2SREGCKI = (Address == ^h4) & !MIO & !BUSST;                 "制御レジスタ 1
!SREGCK  = ((Address == ^h5) & !MIO & !BUSST) # (!I2SDMAAK & BUSST);   "MP3 データストリームレジスタ
!LCDREGCK = (Address == ^h6) & !MIO & !BUSST;                  "制御レジスタ 2

" ATAPI SIGNAL CONTROL
BUS8 = !(CSROM & ATAPICS);          "バスサイズ設定(V53A ダイナミックバスサイジング用)
!CS0 = ATAPICSS &  A8;              "ATAPI CS0#
!CS1 = ATAPICSS & !A8;              "ATAPI CS1#
!ATAPICS = ATAPICSS;                "ATAPI CD-ROM インターフェイスバストランシーバ(74HCT245) G#
```

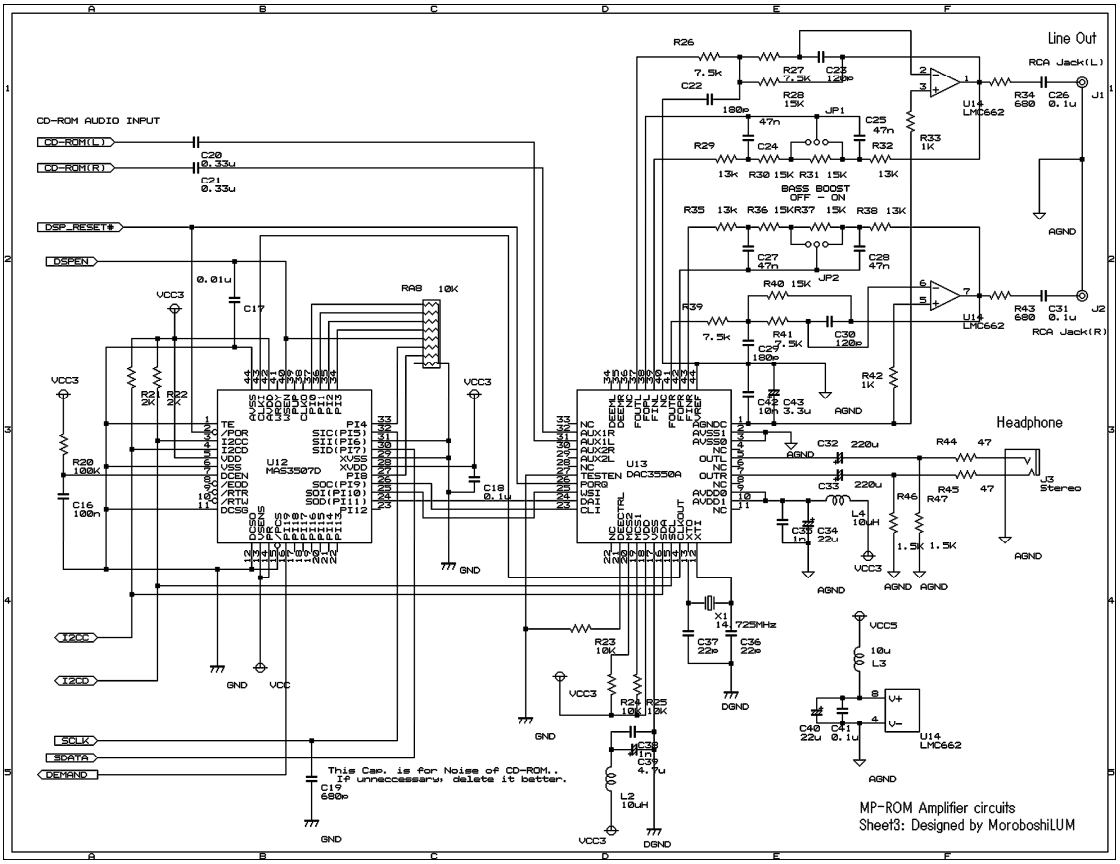
回路図1 メインブロック



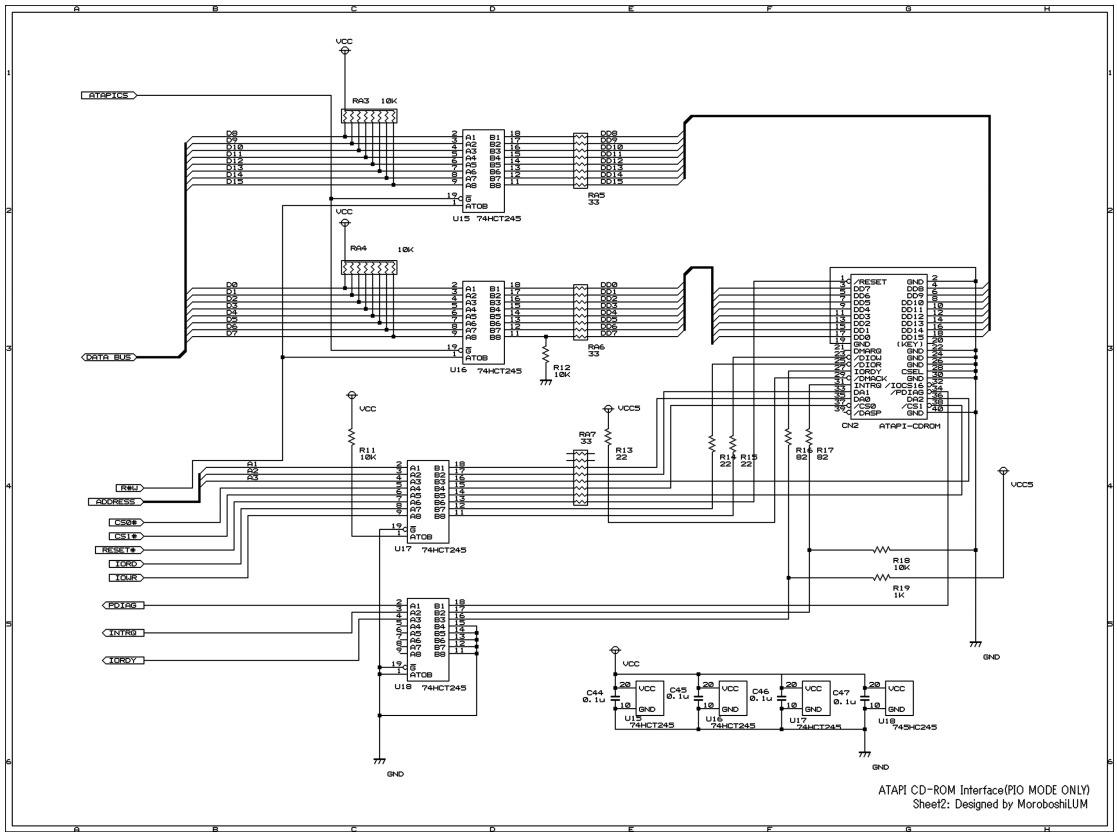
回路図2
キー入力部



回路図3
MP3デコーダ・アナログ系回路



回路図4
ATAPI CD-ROMインターフェイスブロック



LCD制御部分の設計

ラスタ表示の LCD や CRT などは一般的に 1 行、1 行を走査しながら表示します。この際、非力なシステムを対象としたキャラクタ表示専用 LCD コントローラや、ノートパソコン用のグラフィックスチップのように、コントローラ内部に VRAM を持っているようなものを別として、通常は VRAM とするメモリから随時表示するデータの読み出しを行います。このような処理をスチール処理と言います。

スチール処理は CPU 自身が行ってもかまいませんが、この処理は大変重く、例えば今回使用するような 640 × 200、2 階調表示を実現する場合、垂直同期周波数を 70Hz で計算すると 640 × 200 ÷ 8 × 70=1120000 バイトとなり 1 秒間に 1 Mバイトものデータを送らなければいけないことになります。このため、このような処理は CPU が行わず DMA などを使って行うのが一般的です。

DMAの処理

DMA は CPU に代わってハードウェア的にデータの転送を行う機構で、ソフトウェアで行うよりもはるかに高速にデータ転送を行うことができます。この DMA を使って転送するとき、DMA はアドレスバスとデータバスを使用する必要がありますので、同じくバスを使用する CPU とのバスの使用権の調停をする必要があります。この機構は V53A の場合、DMA コントローラを内蔵している関係上、内部でそれを行うため、特にユーザーが考慮する必要はありません。

DMA に転送を要求する場合には、DMARQ というピンを H レベルにします。こうすると、DMA コントローラはバスの調停を行い、その使用権を得ます。そして、バスが利用可能になると DMAAK というピンを L レベルにし、この状態から DMA 転送が開始されます。なお、DMARQ は DMA 転送サイクルでは V53A の場合 T3 サイクルでサンプルされますので、もうデータ転送を停止するときには T3 サイクルの前に DMARQ を L レベルにしておく必要があります。

LCDを表示するためのクロック

前述のように LCD は 1 行 1 行を走査して表示しますが、このタイミングは普通は自分で生成しなければいけません。このタイミングを取るクロックは、この 1 行を表示するクロックの他に、表示データのラッチ用のクロックと、1 フレーム(画面)を送り終えた後に出力するクロックと 3 種類のクロックを入れて上げる必要があります。このクロックのタイミングチャートは今回使用した LCD パネル(EPSON TCM-A0824-2)の場合、図 10 上段のようになります。また、タイミング図には書いてありませんが、スチールの開始・停止はこの LCD パネルでは 4 ライン単位の位置からに開始する必要があります。これについては、ロジックとしては常に 0 ライン目からこれらを開始するようにして簡略化しています。

LCDのDMA転送タイミング

実際の DMA を使って転送するときの各クロックと DMA 制御信号のタイミングは図 10 中・下段のようなタイミングとなっています。なるべく ABEL で作りやすいように、簡略化したタイミングとなるようにしています。

MP3データ転送部分の設計

MP3 オーディオデータの転送は同期シリアル転送で行われます。このとき、MP3 ファイルのデータは MSB から順に送られることにますが、その際のパラレルーシリアル変換は CPLD で行われますので、特に CPU が 1 ビットごとに送るような処理はしません。さらにこのときのデータ転送では再生負荷を減らす為と転送能力を上げるために、DMA を使って転送されます。このため、CPU は CD-ROM からデータを DMA 転送バッファに読み込んだ後、DMA の転送設定レジスタに転送アドレスと転送サイズをセットするだけで、DMA はオーディオデータを CPLD に送り、さらに CPLD がデータをシリアルデータに変換して MP3 デコード DSP に送り、音楽演奏が行われます。

ダブルバッファリング

オーディオデータを再生するときには、そのデータ転送に滞りがおきると音がぶつぶつ切れますので、途切れなく送ってあげる必要があります。しかし、再生バッファに貯めてあったデータがなくなり、新たなデータを CD-ROM から読み出すときには、どうしても一時的に読み出す時間分の途切れが発生することになります。このようなことをなくすために、一般的にオーディオデータなどのストリーミング再生にはダブルバッファリングという手法が使われます。

これは、再生バッファを表バッファと裏バッファの 2 重構造にし、表が再生中のときに裏側にデータを読み込み、逆の場合は表に読み出すようにするものです。こうすると、表(裏)が再生し終わったときには裏(表)側はもう読み込みが終わっていますので、素早く次のバッファに再生処理を移すことができます。これにより途切れのないスムーズなオーディオ再生が可能になります。

CD-ROM読み出しとの平行動作の注意点など

本システムでは、基本的に MP3 データの再生処理はすべて割り込みだけで処理されます。この割り込みは DMA が 1 つのバッファを送信し終わったときに発生する信号/TC を使うものです。つまり、この割り込みが発生したときに、次のバッファに転送アドレスを切り替えるわけです。

なお V53A の場合、転送中に次に切り替えるアドレスを設定できます(オートイニシャライズ機能)ので、最初に表裏の 2 枚のバッファにデータを読み出し、1 つめの表バッファのアドレスとサイズを DMA のレジスタにセットして DMA を開始し、時間を空けて次の裏バッファのアドレスをセットした後は、割り込みが入るたびに CD-ROM から読み出したデータを格納したバッファのアドレスをセットすればよいことになります。

しかし、/TC は全てのチャンネルの DMA の転送終了で発生してしまうので、当然 LCD の 1 画面転送後にも発生します。このため、この信号による割り込みが発生したときは、どの DMA 転送完了でこれが発生したかをきちんとチェックする

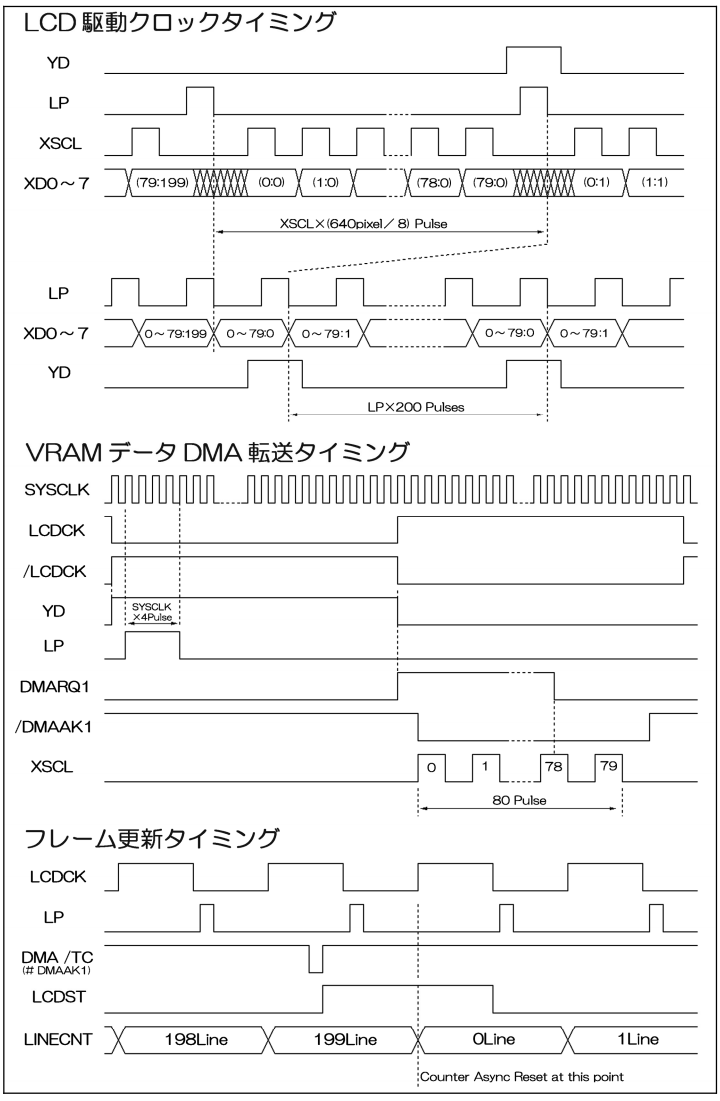
必要があります。
さらにこの割り込みの中で CD-ROM の読み出しも行いますが、前述のように読み出しに INTRQ 割り込みを使いますので、必然的に割り込み許可フラグを解除する必要があります。当然、これを行うと/TC 信号割り込み中に、さらに同じ割り込みが発生しますので、/TC 信号割り込みについては再入について注意を払う必要があります。そのため、/TC 割り込みでは次のように処理します。

MP3再生&CD-ROM読みだしシーケンス

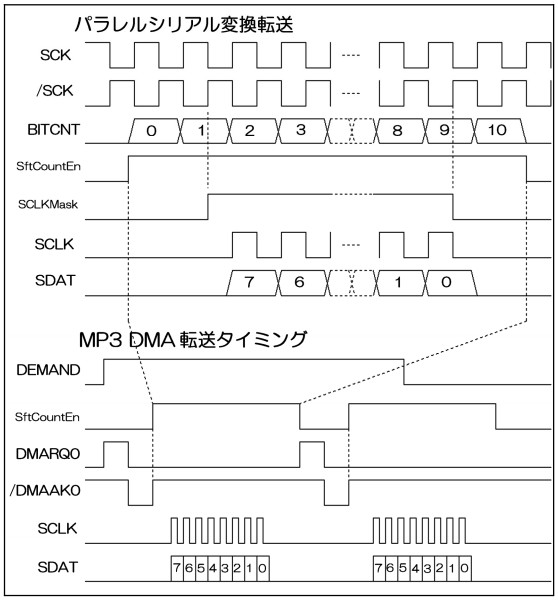
1./TC割り込み発生
2.割り込みコントローラにEOIを先行発行
3.再入フラグをみる。すでに/TC割り込み処理中であれば、そのまま終了。
4.再入でなければ、再入禁止フラグを立てて処理を開始。
5.DMAのレジスタをみて、MP3/バッファの転送完了の割り込みであればCD-ROM読みだし処理開始。そうでなければ、再入フラグを下ろして割り込み処理完了
6.割り込み許可フラグを立てて、CD-ROM読みだしルーチンを呼ぶ
7.また割り込み禁止にする
8.もう一度DMAのレジスタをみて、またMP3/バッファ転送完了の方のフラグが立っていたら、再度読み出し処理を行う（読み出しのアンダーラン対策）
9.8.の処理で、もう要因がなければ再入フラグを下ろして、割り込み処理を完了する（実際には5～9の処理はループで構成されます）

なお、読み出し中に CD-ROM の読み出しに失敗することもありますが、このときには読み出しを即座に中断します。中断は DAC をミュートさせてから、MP3 バッファの DMA 転送許可フラグをマスクし、関連の割り込みをマスクし、MP3 デコーダの動作許可ビットをマスクします。先にミュートを行うのは、強制停止時のブツッというノイズを出さないようにするためです。
さらに、このような停止時だけでなく開始時にもこのようなノイズは載ります。これを防ぐため、今度は逆にデコーダに MP3 データの DMA 転送を開始したあと、時間をおいてから DAC のミュート解除を行います。この遅延ミュート解除は転送開始から 70ms 後で設計しています。これにより、プログラム上では再生はちょっと遅れますが、気が付くほどではありませんので、特に問題はありません。。しかし効果は抜群で、入れないときは出ていたぶつっという音を完全に消すことが出来ます。

右図10
LCD関係のタイミングチャート



下図11
MP3 DMA転送タイミングチャート



CD-ROMドライブの制御

最初に書いたように、CD の制御はこのシステムの中ではもっとも重要なものの一つです。今回は、ATAPI インターフェイスの CD-ROM ドライブを使用した場合を例に取り、CD-ROM デバイスのアクセス手法のテクニックについて解説します。

●CDのトラックの解析

CD は、ある一連の分割単位である複数のトラックにより構成されます。それぞれ、データトラック、オーディオトラックの２種類があり、前者がいわゆるデータファイルなどを格納する領域で、後者が音楽データの格納されたトラックです。通常、データはデータトラックにあり、このトラックはデータ格納に使えるセクタとできるようにきちんと分割され、信頼性を高めるために通常オーディオ CD に付加される誤り訂正符号の CIRC だけでなく、さらに ECC データを付加しています。

このトラックの開始位置情報などは CD の TOC という領域にあり、この TOC を読み出すことで CD 全体の基本構造を知ることができます。ここでは、基本的なトラック情報の取得方法について簡単に説明していきます。

ATAPIデバイスの初期化処理

まず、ドライブの電源を入れた後にドライブのリセット処理を行います。この処理は、IDE のリセット信号(RESET#)を L レベルに 100ms 程度維持してハードウェアリセットをかけたあと、ATA レジスタの中のデバイスコントロールレジスタのソフトウェアリセットビットを立て、さらにソフトウェアリセットをかけています。

このあと、通常は ATA コマンドの IDENTIFY DEVICE コマンドでデバイスに ATA デバイスか、ATAPI デバイスが繋がっているか、どのレベルの転送が可能か(PIO モードや DMA 転送可能かなど）をチェックしますが、今回は ATAPI CD-ROM ドライブが繋がることが前提となっていますので、このようなデバイス判別はしていません。そのため、いきなりデバイスに対して ATAPI コマンドを発行しています。ここでは、必ずまず初めに REQUEST SENSE コマンドを発行します。これを発行することで、最初のリセットで発生しているエラーフラグを解除することになります。これを行わないと、他の ATAPI コマンドを受け付けません。

メディアの検出

トラック情報を取得など行う前にちゃんとドライブにメディアがあり、操作できる状態にあるか、という基本的なことを調べる必要があります。この検出では、TEST UNIT READY コマンドを使います。CD-ROM ドライブなどのリムーバブルメディアを使うデバイスでは、このコマンドが正常に実行されればメディアがセットされ動作可能な状態になっていると判断できます。

トラック情報を取得

TEST UNIT READY コマンドで正常にデバイスが動作することを確認したあと、オーディオトラックとデータトラックの数や位置を取得します。ここでデータトラックが存在しなければデータのある CD ではないと認識できますので、データを検索処理をしないようにすることとなります。

このような情報は READ TOC コマンド(表 6)を使って取得します。このコマンドにより、各トラックの先頭位置(LBA か MSF アドレス)やトラックの属性(オーディオトラックかデータトラックかなど)を取得することができます。情報を取得する場合、コマンドの中のトラック指定フィールドに 1 をセットして、十分なアーロケーションサイズを取れば全トラックの情報を一度に取得することができます。

なおトラックのうち 100(AA)番トラックは、CD では一番最後のトラックの後ろにあるものでリードアウトトラックいい、特別な意味を持っています。この位置を取得することで、最終トラックの位置と差を取ることで、最終トラックの長さを取得することができます。また、逆に先頭のトラックとして、リードイントラックというものもあります。

注意すべき点としては、普通 READ TOC を行った場合に取得される情報は全セッションにあるトラックの情報であることです。このため、マルチセッションのメディアである場合には、セッション間にあるリードイン・リードアウトトラックをきちんと認識できずに、正確に各トラックの情報やメディアの状態を取得できません。このような場合には、さらに詳細な情報を取得するフォーマット 010 を使って、全サブチャネル Q の内容を取得して、この内容を使ってトラックやセッション情報を解析します。

LBAアドレスとMSFアドレス

CD は、元々オーディオ用のメディアであるため、メディア内のアドレス情報は時間情報である MSF アドレス(MinuteSecondFrame)で基本的に表現されます。その名の通り、分、秒、フレームで示され、各桁は 1 秒は 75 フレーム、1 分は 60 秒と決められています。TOC 内のアドレス記述も実際には MSF アドレスですが、一般的な LBA アドレスを出力することもできます。LBA と MSF の相互変換は規格で決まっており、オーディオ CD ではトラック 1 の先頭が LBA=0 であり、データ、または Mixed CD では、MSF アドレス=00:02:00 が LBA=0 として、1LBA=1Frame（MSF=00:00:01）で変換します。

●オーディオCDの演奏制御

CD を演奏するのは簡単で、PLAY AUDIO コマンドで演奏を開始するアドレスと演奏長を指定するだけです。あとは勝手にドライブは指定された時間分の演奏を行います。

このとき、演奏中のトラックや時間などが知りたいときは、READ SUB CHANNEL コマンドを使って、定期的に自分で問い合わせる必要があります。このコマンドを実行すると、演奏中の位置やトラックの情報を返してきますので、これを元に時間情報などを表示します。その他の演奏停止など、は表 5 にあるようなコマンドで制御します。その他のコマンドについては、SCSI 関連書籍などに書いてありますので特に書きません。

表5　CD-ROMドライブ制御で使用される一般的なコマンドの例

コマンド	動作
Test Unit Ready	ドライブの準備確認。実際にはリムーバブルディスクでのメディア確認で使われる。
Request Sense	エラーのクリアと原因取得。デバイスにリセットをかけたときは必ず最初に発行すること。
Seek	指定 LBA 位置へのシークする。基本的には Read コマンドと対で使う。
Read	指定 LBA 位置からの読み出しを行う。読み出しセクタ長は Mode Select コマンドで設定。
Play Audio	CD の演奏開始を行う。MSF 設定で行うものもある。
Read Sub Channel	CD の演奏状態取得する。これで、どの位置を再生しているかなどを検出できる。

※ここらへんのコマンドは SCSI 関連書籍などに詳細が載っていますので、そちらを参照した方がいいです(^^;

表6 READ TOCコマンドの詳細

READ TOC(MMC-2)

Bit / Byte	7	6	5	4	3	2	1	0
0	オペレーションコード(43h)							
1	予約						MSF	予約
2	予約				フォーマット			
3	予約							
4	予約							
5	予約							
6	開始トラック、またはセッション番号							
7	(MSB)　アーロケーション長							
8	(LSB)							
9	予約							
10	予約							
11	予約							

READ TOC(SONY Vendor Unique)

Bit / Byte	7	6	5	4	3	2	1	0
0	オペレーションコード(43h)							
1	予約						MSF	予約
2	予約							
3	予約							
4	予約							
5	予約							
6	開始トラック、またはセッション番号							
7	(MSB)　アーロケーション長							
8	(LSB)							
9	予約							
10	予約							
11	フォーマット	予約(コントロール)						

CONTROL

Bit	1	0
3	4チャンネルオーディオ	2チャンネルオーディオ
2	データトラック	オーディオトラック
1	デジタルコピー許可	デジタルコピー禁止
0	プリエンファシスあり	プリエンファシスなし

ADR(このセクタにあるサブチャネルQの情報の種類)

データ	サブチャネル情報
00h	モード情報なし
01h	カレント位置データ (トラック・インデックス・絶対位置)
02h	メディアカタログ番号
03h	トラック ISRC
04h-0Fh	予約

フォーマットコード
(SONY Vendor Uniqueは00～10まで有効)

フォーマット	ソース	機能	トラック/セッション項の指定
0000	TOC	トラック情報	トラック
0001	TOC	最初と最終のセッション番号、及び最終セッションの開始トラック	予約
0010	TOC	すべてのQサブコード	セッション
0011	PMA	PMA エリアデータ	予約
0100	ATIP	ATIP データ	予約

フォーマット　0000(一般的なTOCデータ)

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)　TOC 情報長							
1	(LSB)							
2	開始トラック番号							
3	最終トラック番号							
トラック詳細情報(指定された開始トラックから、アーロケーション長で指定された長さまで、順に以下の0～7のトラック情報が連続して繋がる)								
0	予約							
1	ADR				CONTROL			
2	トラック番号							
3	予約							
4	(MSB)							
5	トラック LBA/MSF アドレス							
6	(MSF 指定のときは、順に M,S,F,0 と入る)							
7	(LSB)							

フォーマット　0001(セッション情報)

Bit / Byte	7	6	5	4	3	2	1	0
0	(MSB)　TOC 情報長							
1	(LSB)							
2	最初の確定セッション番号							
3	最終の確定セッション番号							
トラック詳細情報								
0	予約							
1	ADR				CONTROL			
2	最終セッションの開始トラック番号							
3	予約							
4	(MSB)							
5	最終セッションの開始トラック LBA/MSF アドレス							
6	(MSF 指定のときは、順に M,S,F,0 と入る)							
7	(LSB)							

データファイルの読み出し

MP3 ファイルを CD-ROM メディアから読み出すためには、データ CD のフォーマットを解析して指定されたファイルがどこにあるかを判別して、該当するセクタよりデータを読み出す必要があります。このようなフォーマットは ISO9660 という規格で基本的な部分が定義されています。ここでは、基本的な ISO9660 と Windows などで使用されている Joliet ファイルシステムを解説します。

●ISO9660

ISO9660 ファイルシステムは CD-ROM の読み出し専用という特性に最適化されたものです。他のディスクなどで使用される FAT などのように、データのあるセクタすべての位置情報ではなく、先頭位置と長さだけの情報だけがあり、読み出す場合は、先頭から順にセクタを長さ情報分を読み出すだけで済みます。ISO9660 では、このようなデータの塊をエクステント(Extent)と呼びます。

このファイルシステムでは互換性を広く保つため、数値の格納形式が LittleEndian(インテル方式)、BigEndian(モトローラ方式)の両方で入っているパラメータが複数あるのも特徴です(表 7)。さらに、格納できる文字コードや長さもかなり制限されており、表 10 の通りになっています。

ボリュームデスクリプタ

ボリュームデスクリプタは、ISO9660 におけるメディアの基本的な情報を格納した連続した LBA=10h より始まるセクタ群です。このセクタ群を解析するときは先頭から順に、それぞれのセクタの先頭にあるデータが ??CD001 と書かれていることを確認し、??の値を見ることで、どのようなセクタなのかを判別します(表 9)。この中で一番重要なものがプライマリボリュームデスクリプタ(PVD)で、ISO9660 フォーマットでは必ずこのセクタが存在します。この PVD には、ルートディレクトリ情報のあるセクタの位置などが書き込まれており、ここからデータの探索を開始します。

この他にサプリメンタリボリュームデスクリプタ(SVD)というものもあります。この SVD を別のファイルシステムを含めるときにはそれに対応したものを置いておき、そのシステムを使うときだけその SVD を使うことにより、ISO9660 以外のシステムを混在させることができます。これらの PVD,SVD のあるセクタの終端はボリュームデスクリプタ終端識別子のあるセクタ(先頭が 0xFF'CD001')で終了します。

ディレクトリとファイルの読み出し

PVD や SVD にあるルートディレクトリ情報や、そのルートディレクトリから下にぶらさがるディレクトリセクタに複数入っているレコードは表 10 のような構造になっています。それぞれ存在するファイル名やディレクトリ名、その位置とサイズ情報が記述されています。ファイルを読み出すときは、目的とするファイル名の所にある位置情報とサイズを取得し、その位置からサイズ分だけ連続的にセクタを読み出せばよいことになります。

また、指定したディレクトリの下に降りたいときには、そのディレクトリ位置の先頭セクタよりデータを読み出して、そのデータのディレクトリ情報を同じように解析します。

ファイル名は、ディレクトリセクタの中にファイル識別子として入っています。ISO9660 では、普通はその規定内の文字(D 文字列:制限のある ASCII コード)が入っているだけですので、ファイラーに表示したい場合には、そのまま文字コードを表示するだけです。この他に ISO9660 の制限が緩いレベルを使って、ShiftJIS 文字列をそのまま入れていることもありますが、これも Windows 系では一般的な文字コードですので、特に変換する必要もなく表示ができます。ただし、ファイル名には改版番号という番号(数値文字)がセパレータ";"とともについていますので、表示するときにはきちんと切り離す作業を行う必要があります。

この他に特別なファイル名として 00h と 01h の 1 バイトのものがあり、それぞれカレントディレクトリと親ディレクトリを示します。このため、この情報を使えばそれらに関する情報を常に保持していなくても、そのディレクトリ内の情報だけで、ディレクトリを上下に行き来することができるわけです。

パステーブル

パステーブルは適切にソートされたディレクトリセクタの位置のみを記述したテーブルです。このテーブルを先行して読み出しておくことで、ルートディレクトリから階層を降りていかなくても、すぐに目的のディレクトリ階層に入ることができます。なお、今回のシステムでは使用していないので説明は省略します。

●Joliet

Joliet ファイルシステムは ISO9660 を拡張したもので、Microsoft が提唱したものです。いくつかの特徴がありますが、ファイル名を最大 60 文字まで設定可能で、ファイル名などの文字コードに UNICODE を採用することで、地域による文字コードの違いに左右されることなく、ファイル名表示が可能となったことがあります。

この Joliet ファイルシステムは先に解説した SVD を使っています。Joliet では、この SVD のエスケープシーケンスに特定文字列が入っており、ここを見て Joliet かどうかを判別します(表 8)。他の部分においては、基本的にすべて同じで、格納されている文字コードが UNICODE、つまり 2 バイト文字列となっているだけですが、カレントディレクトリと親ディレクトリを示す 00h、及び 01h に関しては 1 バイトのままとなります。なお、これらの文字コードは BigEndian(モトローラ形式)で入っていることに注意します。

UNICODE

UNICODE は全世界の文字を 2 バイトコードに押し込んだものです。「押し込んだ」というのは、例えば中国語で日本語で使われる漢字と似たようなものもありますが、そういう文字に同じコードを振っていたりすることから言えます。お陰で、いろいろな方面で評判最悪らしいですが(^^;。

取り敢えず、Joliet ではこの文字コードが使われていますが、Windows などで一般的な文字コー

ドの Shift JIS とは違うので変換してあげる必要があります。しかし、JIS コードからの変換のように一意の計算だけで変換できず、全くコード的には互換性がないため、普通はテーブルを使って変換します。ただし、日本語だけを表示すると限定した場合には、全ての UNICODE 文字コードを変換する必要はないので、適当なコードを間引きしたテーブルを用いるだけで十分で、今回設計したものでも半分のコードは変換していません。

●マルチセッション

セッションとは CD における区画のことで、構造としては先に出たリードイントラックとリードアウトトラックに挟まれた部分です。基本的なオーディオ CD ではこれが一つだけで、これをシングルセッションといいます。これが複数あるものがマルチセッションです。このマルチセッションは一般に CD-R に追記するときなどに使われ、新しくデータを追加するたびに新しいセッションが追加され、そこに追加ファイルが書かれます。

マルチセッションにおける読み出しは Joliet でも定義されており、最終セッションの開始トラックの先頭から相対アドレスで 10h の場所から、全体のボリュームデスクリプタが始まる、とされます。これはシングルセッションの場合でも最終セッション＝そのセッションというだけで、意味としては同じですからフォーマットとしては上位互換と言えます。このため、プログラム上ではマルチセッションもシングルセッションも同じプログラムで対応することができます。

セッション情報の読み出し

セッション情報を読み出す場合、最近の MMC-2 規格では READ TOC のフォーマット設定エリアにセッション読み出し指定 0001 を設定すれば読めることには表向きなっています。しかしながら、実は最近のドライブでもこれをまともにサポートしているものはあまりありません。では、どのように読み出すのかというと、別のベンダユニークフィールドでフォーマットを指定して読み出します[*1]。これで、最終セッションの開始トラックの LBA アドレスがわかりますので、この LBA に 10h を足したアドレスからボリュームデスクリプタの解析を開始します。

さらに正確に解析するには前述の全サブチャネル Q データを取得し、その内容を詳細に解析する必要がありますが、普通にデータを読み出す分にはそこまでする必要はありません。

●実際のデータの読み出し

基本的には、ISO9660 と Joliet に準拠したディレクトリ解析で目的のファイルを探し出して、エクステントの最初(データ開始 LBA)とデータサイズを取得し、その LBA より SEEK コマンドと READ コマンドを使って読み出していきます。

経験上、ATAPI CD-ROM ドライブでは、あまり間隔をあけずに頻繁に連即的にコマンドを発行するとドライブがこけたりするようなので、なるべく複数のセクタはまとめて読むことや、連続的にコマンドを発行する場合にはウェイトを入れるなどの細工をする必要があるようです。

表7 ISO9660における数値や時間の定義(型名は便宜的につけています)

数値型

型名	サイズ	型
UINT8	8 ビット	8 ビット符号なし数値
SINT8	8 ビット	8 ビット符号あり数値
UINT16L	16 ビット	16 ビット符号なしリトルエンディアン(インテル方式)数値
UINT16B	16 ビット	16 ビット符号なしビッグエンディアン(モトローラ方式)数値
UINT16W	16 ビット×２	16 ビット符号なしリトル&ビッグエンディアン数値
UINT32L	32 ビット	32 ビット符号なしリトルエンディアン(インテル方式)数値
UINT32B	32 ビット	32 ビット符号なしビッグエンディアン(モトローラ方式)数値
UINT32W	32 ビット×２	32 ビット符号なしリトル&ビッグエンディアン数値

ボリュームデスクリプタなどで使用される時間情報(DT_BCD)

オフセット	型	内容	範囲
0～3	DSTR(4)	西暦年	"0000"～"9999"
4～5	DSTR(2)	月	"01"～"12"
6～7	DSTR(2)	日	"01"～"31"
8～9	DSTR(2)	時	"00"～"23"
10～11	DSTR(2)	分	"00"～"59"
12～13	DSTR(2)	秒	"00"～"59"
14～15	DSTR(2)	1/100 秒	"00"～"99"
16	SINT8	UTC からの偏差(15 分単位)	-48 ～+52(-12 ～+13 時間)

ディレクトリレコードで使用される時間情報(DT_BIN)

オフセット	型	内容	範囲
0	UINT8	西暦年	0 ～ 255(1900 ～ 2155 年)
1	UINT8	月	1 ～ 12
2	UINT8	日	1 ～ 31
3	UINT8	時	0 ～ 23
4	UINT8	分	0 ～ 59
5	UINT8	秒	0 ～ 59
6	SCHAR	UTC からの偏差(15 分単位)	-48 ～ +52(-12 ～ +13 時間)

文字列

DSTR(N)	N バイトの D 文字列	**ASTR(N)**	N バイトのＡ文字列

*1 この指定は、SONY のベンダユニークコマンドに由来します。

表8　Jolietのエスケープシーケンス(ISO2022 UCS2 Escape Sequence)

Standard Leve	Decimal	Hex Bytes	ASCII
UCS2 Level 1	2/5 , 2/15 , 4/0	(25) (2F) (40)	'%¥@'
UCS2 Level 2	2/5 , 2/15 , 4/3	(25) (2F) (43)	'%¥C'
UCS2 Level 3	2/5 , 2/15 , 4/5	(25) (2F) (45)	'%¥E'

表9　ISO9660ボリュームデスクリプター一覧

プライマリボリュームデスクリプタ(PVD)とサプリメンタリボリュームデスクリプタ(SVD)

オフセット	型	メンバ名	内容・備考
0	UINT8	ボリュームデスクリプタの種類	1: ボリュームデスクリプタ 2: サプリメンタリボリュームデスクリプタ
1 ～ 5	DSTR(5)	規格 ID	“CD001”
6	UINT8	ボリュームデスクリプタバージョン	1
7	Flag	ボリュームフラグ(SVD のみ有効)	ビット 0 =1 のとき、ISO2375 登録外のエスケープシーケンスを含む。0 のときは含まれない。ビット 1 ～ 7 は未定義。PVD は 00 h が入る。
8 ～ 39	ASTR(32)	システム ID	
40 ～ 71	DSTR(32)	ボリューム ID	MS-DOS/Windows ではボリュームラベル
72 ～ 79	-	未定義	ALL 00h
80 ～ 87	UINT32W	ボリュームスペースの大きさ	ボリュームスペースの論理ブロック数
88 ～ 119	STRING(32)	エスケープシーケンス(SVD のみ)	G0/G1 図形文字集合を指示。PVD では 00h が入る。 Joliet では、ここに判別のためのコードが入る。
120 ～ 123	UINT16W	ボリュームセットのサイズ	ボリューム集合の大きさ
124 ～ 127	UINT16W	ボリュームシーケンスナンバー	ボリューム順序番号
128 ～ 131	UINT16W	論理ブロック長	論理ブロックのサイズ(セクタサイズ)。普通は 2048。
132 ～ 139	UINT32W	パステーブルの大きさ	パステーブルのバイト数
140 ～ 143	UINT32L	L 型パステーブルの位置	LittleEndian 形式で格納される L 型パステーブルの位置(LBA)
144 ～ 147	UINT32L	オプション L 型パステーブルの位置	LittleEndian 形式で格納される L 型オプションパステーブルの位置(LBA)
148 ～ 151	UINT32B	M 型パステーブルの位置	BigEndian 形式で格納される M 型パステーブルの位置(LBA)
151 ～ 155	UINT32B	オプション M 型パステーブルの位置	BigEndian 形式で格納される M 型オプションパステーブルの位置(LBA)
156 ～ 189	ディレクトリレコード	ルートディレクトリ	ルートディレクトリの内容がはいる
190 ～ 317	DSTR(128)	ボリュームセット ID	
318 ～ 445	ASTR(128)	出版社識別子	_で始まる場合はファイル識別子。
446 ～ 573	ASTR(128)	データ編集者識別子	_で始まる場合はファイル識別子。使用した CD-R ライタソフトのソフトウェア名称が入る場合が多い。
574 ～ 701	ASTR(128)	応用システム識別子	_で始まる場合はファイル識別子。
702 ～ 738	DSTR(128)	著作権ファイル識別子	セパレータ 1,2 も含む
739 ～ 775	DSTR(128)	抄録ファイル識別子	セパレータ 1,2 も含む
776 ～ 812	DSTR(128)	書誌ファイル識別子	セパレータ 1,2 も含む
813 ～ 829	DT_BCD	ボリューム作成日時	
830 ～ 846	DT_BCD	ボリューム更新日時	
847 ～ 863	DT_BCD	ボリューム失効日時	
864 ～ 880	DT_BCD	ボリューム発行日時	
881	UINT8	ファイル構造版数	
882	-	予約	00h
883 ～ 1394	-	アプリケーション用	
1395 ～ 2047	-	予約	ALL 00h

ボリュームパテーションデスクリプタ

オフセット	型	メンバ名	内容・備考
0	UINT8	ボリュームデスクリプタの種類	3: ボリュームパテーションデスクリプタ
1 ～ 5	DSTR(5)	規格 ID	“CD001”
6	UINT8	ボリュームデスクリプタバージョン	1
7	-	未使用	00h
8 ～ 39	ASTR(32)	システム ID	
40 ～ 71	DSTR(32)	ボリュームパテーション ID	
72 ～ 79	UINT32W	ボリュームパテーションの位置	先頭位置の LBA
80 ～ 87	UINT32W	ボリュームパテーションのサイズ	論理ブロック数表示
88 ～ 2047	-	システムが使用	-

ブートレコード

オフセット	型	メンバ名	内容・備考
0	UINT8	ボリュームデスクリプタの種類	0: ブートレコード
1 ～ 5	DSTR(5)	規格 ID	“CD001”
6	UINT8	ボリュームデスクリプタバージョン	1
7 ～ 38	ASTR(32)	ブートシステム ID	
39 ～ 70	DSTR(32)	ブート ID	
71 ～ 2047	-	ブートシステムが使用	-

ボリュームデスクリプタセット終端

オフセット	型	メンバ名	内容・備考
0	UINT8	ボリュームデスクリプタの種類	255: ボリュームデスクリプタの終端レコード
1 ～ 5	DSTR(5)	規格 ID	“CD001”
6	UINT8	ボリュームデスクリプタバージョン	1
7 ～ 2047	-	予約	ALL 00h

表10　ディレクトリレコードの詳細

ファイル名とディレクトリ識別子

【形式】ファイル名．ファイル拡張名；ファイル版数番号

ファイル名や拡張名に使用できる文字列	D 文字、または D1 文字
ファイル名のサイズ ※必ずピリオドがあること。	ISO9660 水準 1:　8.3 文字列(DOS と同じ) ISO9660 水準 2,3: 30 文字まで。
ディレクトリ識別子 ※拡張名は使えない。	ISO9660 水準 1:　8 文字まで ISO9660 水準 2,3: 31 文字まで。 ただし、以下のデータは特別の意味をもつ。 00h：カレントディレクトリ 01h：親ディレクトリ
ファイル版数番号	ファイルの更新履歴を記録する(1 ～ 32767)。通常は"1"

ディレクトリレコード形式

オフセット	型	内容
0	UINT8	ディレクトリレコードの長さ(LEN_DR)
1	UINT8	拡張属性レコードの長さ
2 ～ 9	UINT32W	エクステントの位置(LBA)
10 ～ 17	UINT32W	データ長(バイト)
18 ～ 24	DT_BIN	記録日時
25	Flag	ファイルフラグ*
26	UINT8	ファイルユニットの大きさ
27	UINT8	インターリーブギャップサイズ
28 ～ 31	UINT16W	ボリュームシーケンスナンバー
32	UINT8	ファイル識別子の長さ(LEN_FI)
33 ～(33+LEN_FI)	STRING	D,D1,セパレータ 1,2 キャラクタ、または 00h,01h
33+LEN_FI	UINT8	パディングバイト 00h(LEN_FI が偶数のとき)
LEN_DR-LEN_SU ～ LEN_DR-1	LEN_SU Byte	システムが使用

ファイルフラグ

ビット位置	ビットフィールド名	0	1
0	存在	可視ファイル	不可視ファイル
1	ディレクトリ	ファイル	ディレクトリ
2	関連ファイル	主ファイル	関連ファイル
3	レコード	ファイルがレコード形式でない	ファイルがレコード形式である
4	保護	許可条件無効	許可条件有効
5 ～ 6	予約	常にゼロ	-
7	複数エクステント	ファイルの最終ディレクトリレコードではない	ファイルの最終ディレクトリレコードである

設定の保存と読み込み

今回は DAC の音声ボリュームを使用します。しかし、そのままでは電源を切ったり、リセットしたりしたら、前回設定したボリューム値は消えてなくなってしまいます。そのため、本システムではプログラムを納めている NOR タイプのフラッシュメモリでプログラムに利用していない領域にこのようなパラメータを保存します。

フラッシュメモリにデータを書き込むときには、プログラムから特定のアドレスに特定のデータを書き込む操作を何度か行い、プログラム状態にします。この状態にすると、フラッシュメモリは書き込みが完了するまでは、特定のデータしか吐かなくなるために、プログラムを動作させる状態でなくなります。このため、このような操作は RAM にそれを行うプログラムをコピーして、一時的に RAM にコピーしたプログラムを呼び出してあげるというようなことをします。

その他の部分の設計

ここからは全体的なことや、プレイヤーの演奏、表示部分に関する解説を行っていきます。

システム全体の設計

インターフェイスの基本となる GUI は、簡単に「実行」「キャンセル」ボタンのあるオーバーラップウィンドウを基本に作られています。ボタンの表示などはパラメータで簡単に制御できるようにしており、これをベースにほとんどの部分を作っています。

GUIの例(演奏開始問い合わせダイアログ)

基本構造

本システムは大きくは MP3 プレイヤー部分と CD プレイヤー部分の２つの部分に分かれます。MP3 プレイヤーや CD プレイヤーのどちらを有効にするかは、最初のトラック検出で設定されます。データトラックがあれば MP3 プレイヤーが、オーディオトラックがあれば CD プレイヤーが有効になり、両方が有効なときの切り替えは FUNC キーでトグルで行われます。

●MP3プレイヤーの構造

MP3 プレイヤー部分はファイラーから選択されたファイルをプレイリストバッファに登録し、演奏ルーチンがこのプレイリストバッファの内容をみて演奏します。このルーチンの中で、実際に選択されたファイルが MP3 ファイル、WAVE ファイル、あるいは RMP3 ファイルかを判断し、適切に演奏開始位置と演奏サイズを設定し、ID3 タグを検出してパラメータの表示を行います。

演奏ファイル名の拡張子は、一般に内容との一致が信用できないので[*1]、きちんと内部構造をチェックする必要があります。基本的には WAVE や RMP3 のように Windows RIFF ファイルであるものかどうかをチェックして、そうであれば実際のデータ開始位置や、データサイズを"data"チャンクを解析して設定します。そうでなければ生の MP3 ファイルかどうかを簡単にチェックして演奏を行います。なお、WAVE ファイルの場合は、重ねて内部の"fmt "チャンク内にあるフォーマット情報の中のフォーマットタグメンバが MP3 であるか(55h であるかどうか)もチェックします。

演奏を開始すると、演奏時間などの更新を定期的に行います。これは、タイマを使って非同期に変えています。実際にはフレーム数から判断すべきですが、さほどメリットもないので今回は実時間表示をしています。

演奏のメインループではキー入力のみポーリングでチェックして、なんらかのキーが押されたら適当なアクションを行います。

一般的なWindows RIFFファイルの構造

オフセット	サイズ	内容
0	4	"RIFF"
4	4	このメンバ以降のサイズ
8	4	クラス。"WAVE"や"RMP3"、"AVI "などの文字列が入る
12+n	4	チャンク名。以下にどういうデータが入るかを示す。 "fmt " フォーマット情報 "data" データ本体 "fact" サンプル数
16+n	4	そのチャンクのサイズ。
・・・・		サイズ分のデータ
上記チャンク部分が連続して続く・・・		

演奏が完了すると、プレイリストに演奏する曲が残っていないかチェックし、まだあれば、また演奏を開始します。なければ、演奏時間ウィンドウを閉じて、またファイラーに戻ります。

回復不能なエラーの発生に対する対処

読み出しエラーが起こった場合、何度かリトライを行いますが、それでも回復できない場合には強制停止処理を行って演奏ループを脱出し、トップメニューに戻ります。このとき、デバイスチェックを行ってメディアがなければ待機状態となります。特に本システムのような場合、イジェクトボタンなどを押して強制的に停止される場合が頻繁にありえますので、このような状態はきちんと対処しておく必要があります。

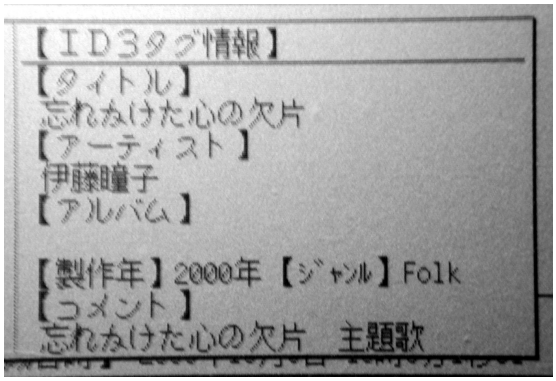
*1 ほとんどのプレイヤーはきちんと先頭ヘッダをみて演奏制御しているために拡張子は無関係に再生可能なので、RIFF/WAVE と MP3 のファイルを使用者が「ファイル名を変えるだけで変換できる」と勘違いしている場合が多かったりします(^^;

●ID3タグの表示

ID3 タグ(Ver.1.1)は MP3 ファイルの最後に付加される 128 バイトの曲情報データです。このデータは単純なテキストデータで、項目は特定のオフセットから並んでいるだけですので、そのまま内容を分解して表示すればいいだけです。なお、ID3 タグの先頭には必ず"TAG"の 3 文字があるので、プログラム内部では最初の表示時に最終データのあるセクタを計算して読み出し、最後から 128 バイト手前がこの ID であるかどうかをみて、そうであれば、下の写真のように ID3 タグを表示します。

注意が必要なのは、WAVE ファイルであっても ID3 タグが付く可能性があることです。これは脚注で触れたように使用者の問題ではありますが、Windows RIFF ファイルの構造上、そのようなゴミがあっても「きちんと data チャンクサイズを解析すれば」ファイル構造上問題にはならないので、きちんとこれを考慮した解析はする必要はありません。まぁ、ID3 タグも吐き出しても演奏終了地点なので、実害はないのですが(苦笑)

ID3タグ表示例(PC Angel 12月号付録CDより)



ID3タグ V1.1の構造

オフセット	サイズ	内容
0	3	"TAG"
3	30	タイトル名
33	30	アーティスト名
63	30	アルバム名
93	4	作成年度
97	30	コメント
127	1	ジャンル番号

ID3 タグの 128 バイト内のデータの配列は上記の通りです。ジャンル番号以外はすべて ASCII 文字列ですが、ヌル文字でターミネーションはしません。たとえば、タイトル名が 8 文字であれば、残りの 22 文字はすべてスペース(20h)で埋めておきます。

このような ID3 タグは Ver2.0 というものもあります。これは V1.1 のようにファイルの固定領域に付加するものではなく、フレームの間に埋め込む方式のもので、V1.1 よりもかなり多機能となっています。しかし、現状は特に普及しているわけでもないので、いまのところ本システムでは実装していませんが、今後は必要になってくるかもしれません。

最後に

以上、駆け足で説明をしていきました。ページの都合上、重要なポイントや引っかかりそうな点に絞って解説していきましたので、この記事とソースコードを併せて読むと大体わかるのではないかと思います。

今回、初めて CPLD などいろいろ自分の使ったことのない部品や、滅多に使わない(笑)DMA 機能を使ったりと、苦労はありましたがなかなか楽しい製作でした。ただ、設計規模が大きかったので、配線が滅茶苦茶になってしまっているのが残念。

完成したあとに自分の Web ページの英語ページに簡単に掲載しましたが、MPROM は結構人気のあるネタなので、ユーゴスラビアの技術者さんとかインドネシアの学生さんやら、いろいろな人からメールを貰いました。こういうのも製作の醍醐味かな。

さて次は、何をネタにしようかな・・・(^^;;;;

なお、ここで紹介した MPROM のソースコードや回路図などはわたしの Web ページにてダウンロード可能ですので、印刷が見難いところなどは(^^;そちらを参照してください。

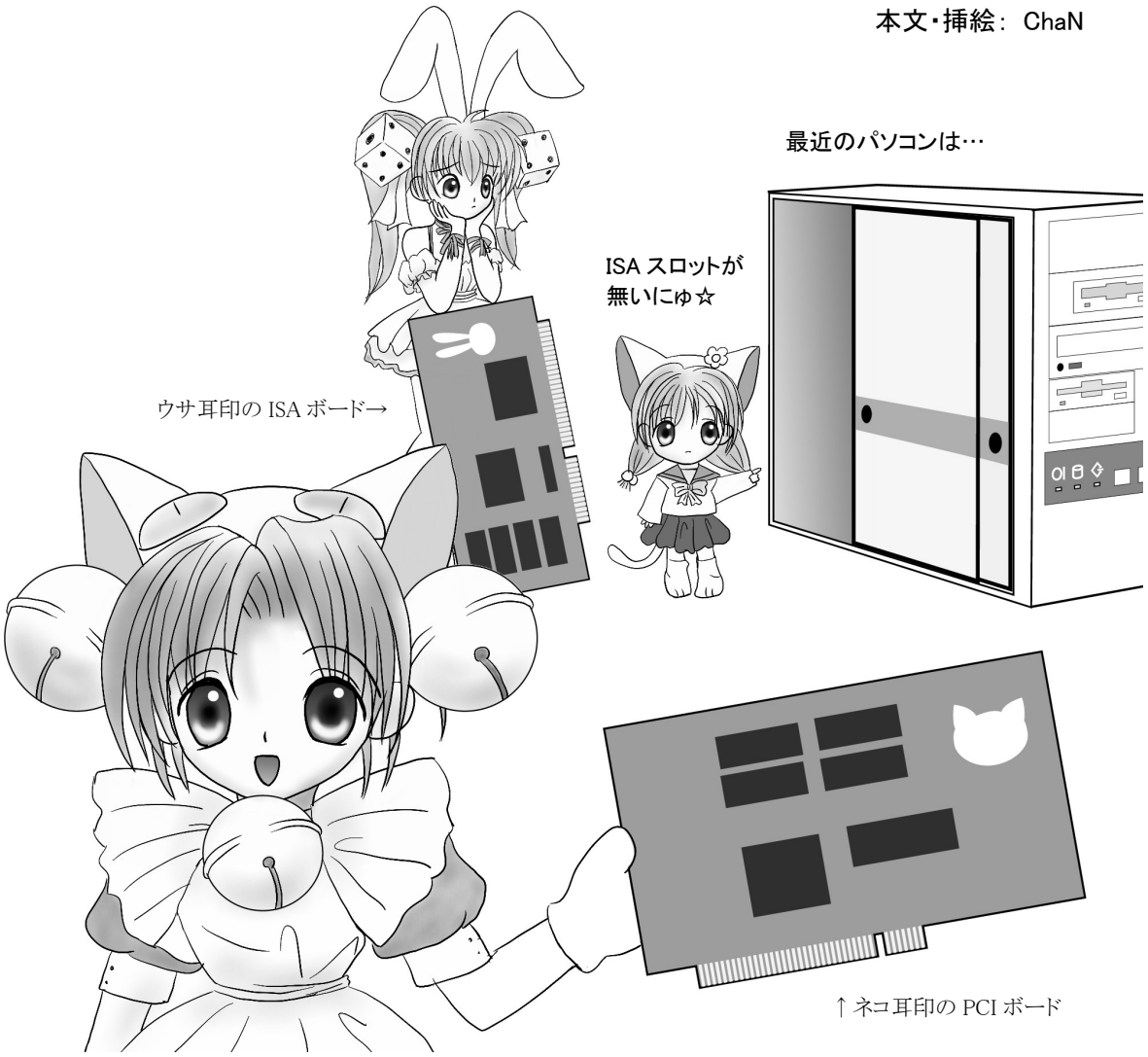
みゃん☆みゃん☆ふぁくとりぃ☆
ハードウエアのページ
http://www2s.biglobe.ne.jp/~elfin/hardware.html

※ PC-VAN が閉鎖したら引っ越しするかも知れません。引っ越しするとしたら、http://member.nifty.ne.jp/ELFIN/ になると思いますので、URL 不明だったらベース URL をこれにしてアクセスしてみてください。

■ PCI-PIO ボードの製作 ■

PCIで自作だにょ☆

本文・挿絵： ChaN



これからはＰＣＩだにょ★

● PCI の普及と ISA の衰退

いまさら言うまでもありませんが、現在 PCI はマイコンシステム用のバスとして事実上の標準規格となっています。そして発表されているパソコンは全て PCI を採用するようになっています。

以前は業界標準だった ISA バス製品も PCI の普及に伴い徐々に姿を消しつつあります。PCI の初期の頃は**高価で高性能な PCI** と**低性能だけど安価な ISA** という形で共存してきたのですが、最近は PCI 関係も枯れてきて ISA の価格面でのメリットが完全に無くなってしまいました。既に市場からは ISA のアドインボードがほとんど姿を消し、パソコンから ISA スロットが全滅するのはもう時間の問題といえるでしょう。

でも、こんなに身近になっているにもかかわらず、このバスを活用した製作モノにはほとんどお目にかかることがありません。PCI はその高機能さからか複雑難解でアマチュアの手に負える物ではないと思われているようです。実際、PCI の規格を満たす機能をフル実装したアドインボードを TTL や SPLD などの個別部品で自作することは不可能といえます。

このことは趣味の電子工作にとっては大問題です。今までのように TTL や SPLD を使って手軽にアドインボードを自作することができなくなってしまうのです。

● とにかくいじってみるにょ！

TTL や SPLD で自作不可能ではもうどうにもならないじゃないかというところですが、とりあえず逃げ道もあります。PCI はシステム全体としては多くの動作モードや機能が定義されていて複雑ですが、基本的なバスサイクルはきわめて簡単な手順で動作しています。このため、**PCI のターゲットデバイスとしてとりあえず動作するという程度のものなら、SPLD などで自作することも可能**なのです。

ただし、そのようにして自作したボードは、PCI デバイスとして必須とされる機能を一部省いた不完全な PCI デバイスということになります。したがって、どのようなシステムでも動作するとは限りませんので、製品として使用することはできません。**飽くまで PCI の動作を理解するための実験**に限られます。

ＰＣＩバスの基本動作にょ★

まず、基本的な PCI バスの動作を説明しておきましょう。PCI は完全な同期バスなので、ISA のような組み合わせロジックの概念は通用しません。全ての動作はクロックで駆動されるステートマシンにより制御されます。次に代表的なバスサイクルについて簡単に説明します。

※ここで取り上げるものは多くの種類のバストランザクションのうち一部にすぎません。

● シングルライトサイクル

最も単純なバスサイクルです（**図 1**）。イニシエータは FRAME#をアサートしてバスサイクルを開始します<1>。FRAME#と同時にバスコマンド（C/BE#[3:0]）とアドレス（AD[31:0]）を示します。

イニシエータは<2>のタイミングでバスイネーブル（C/BE#[3:0]）と転送データ（AD[31:0]）を出力して IRDY#をアサートします。それと同時に FRAME#をデアサートして次のデータ転送でそのサイクルが終了することを示します。

ターゲットは<2>のタイミングでバスコマンドとアドレスを取り込み、自身に対するアクセスの場合は DEVSEL#をアサートしてバスサイクルに応答します（この場合は 1 クロックで応答しているので高速デコード）。同時に TRDY#をアサートしてデータ転送が可能であることを示しています。

ターゲットは<3>のタイミングで IRDY#、TRDY#をチェックして、IRDY#と TRDY#が共にアサートの場合は AD[31:0]のうち C/BE#[3:0]の示す有効なバイトレーンからデータを取り込みます。イニシエータも IRDY#、TRDY#をチェックしていて、データがターゲットに取り込まれたことを知り、サイクルを終了します。

<3>でデータ転送が行われたとき FRAME#はデアサートであるため、ターゲットはそれが最後の転送であることを知ります。最後のデータが転送されたらターゲットは DEVSEL#と TRDY#をデアサートしてサイクルを終了します。

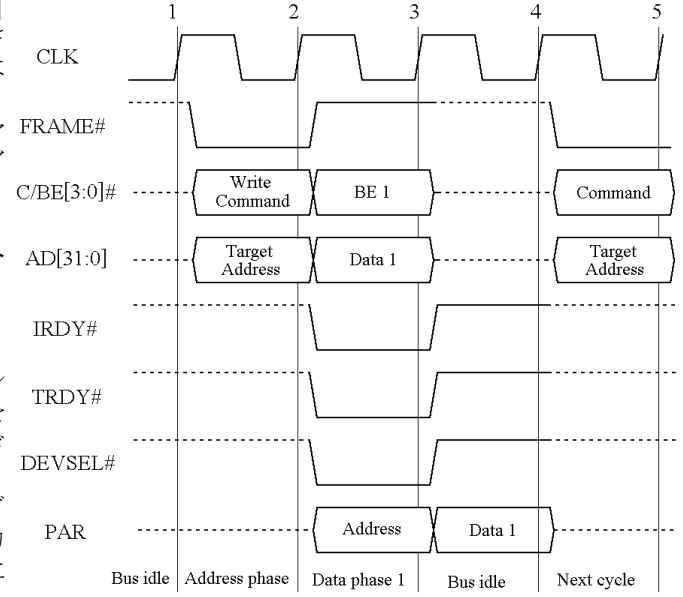
● シングルリードサイクル

最も単純なリードサイクルです（**図 2**）。リードサイクルではコマンドフェーズからデータフェーズに変わるとき AD[31:0]をドライブするデバイスの切り替えが発生します。このため、バスの衝突防止のため緩衝時間をコマンドフェーズとデータフェーズの間に 1 クロック設けています。したがって、リードサイクルはライトサイクルに比べて少なくとも 1 クロックは余計に必要となります。

イニシエータは FRAME#をアサートしてバスサイクルを開始します<1>。FRAME#と同時にバスコマンド（C/BE#[3:0]）とアドレス（AD[31:0]）が示されます。

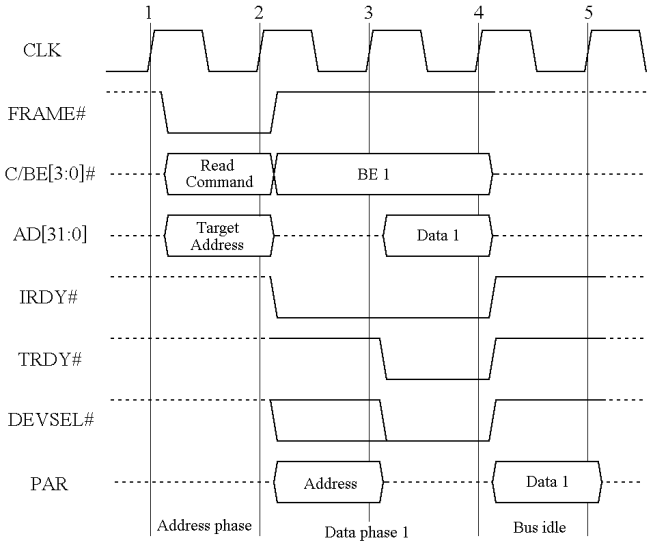
イニシエータは<2>のタイミングでバスイネーブル（C/BE#[3:0]）を出力して IRDY#をアサートします。それと同時に FRAME#をデアサートして次のデータ転送でそのサイクルが終了することを示します。 ターゲットは<2>のタイミングでバスコマンドとアドレスを取り込み、自身に対するアクセスの場合は DEVSEL#をアサートしてバスサイクルに応答します（この場合は 1 クロックで応答しているので高速デコード）。ただし、バスの衝突防止のため、AD[31:0]にデータを出力するのは<3>以降のタイミングでなければなりません。このとき、C/BE#[3:0]で指定されたバイトレーンに有効なデータを出力するのは当然ですが、AD[31:0]全てがパリティ計算の対象となるため、それ以外のバイトレーンにも何らかの確定した値を出力しなければなりません。

<図 1> シングルライトサイクル



イニシエータは<4>のタイミングで IRDY#、TRDY#をチェックして、IRDY#と TRDY#が共にアサートの場合は AD[31:0]からデータを取り込み、サイクルを終了します。ターゲットは IRDY#、TRDY#をチェックしていて、データがイニシエータに取り込まれたことを知ります。このとき FRAME#はデアサートであるため、ターゲットはそれが最後の転送であることを知り、DEVSEL#と TRDY#をデアサートしてサイクルを終了します。

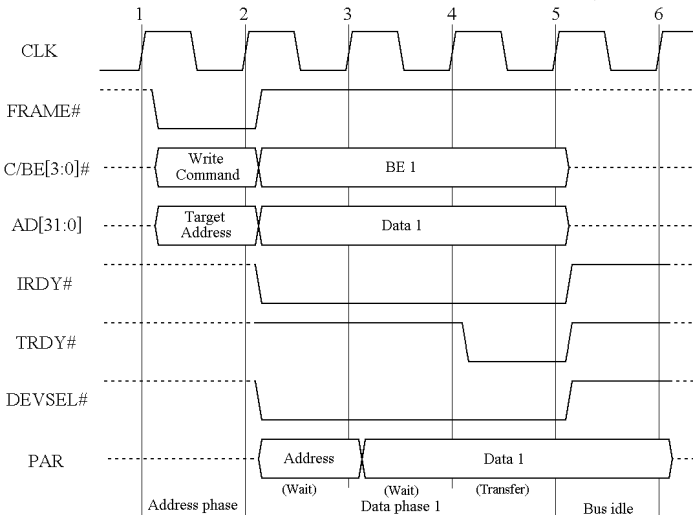
<図 2> シングルリードサイクル



● シングルライトサイクル(ウェイトあり)

PCI ではイニシエータとターゲットがそれぞれレディ信号(IRDY#と TRDY#)を出していて、共にレディ状態のときだけデータ転送が行われます。他のバスのようにイニシエータ主導でターゲットだけがウェイトを要求するのとは違って、PCI は同期バスのためそれぞれが対等の立場でハンドシェークが行われているのです(**図 3**)。

<図 3> シングルライトサイクル(ターゲットのウェイトあり)



● バースト転送(ライトサイクル、ウェイトあり)

バースト転送はシングル転送サイクルと区別されているわけではありません。ターゲットは、データが 1 回転送される毎に FRAME#をチェックしていて、FRAME#がアサートされいる間は次のデータ転送が続くことを知り、それによりバースト転送を実現しています。イニシエータは最後のデータ転送で IRDY#をアサートするのと同時に FRAME#をデアサートしてその転送でサイクルが終了することをターゲットに通知します(**図 4**)。

バースト転送はメモリデバイスに対して行われるのが普通ですが、I/O デバイスへのバースト転送もありです。

PCI バスがバースト転送で最も性能が出るように設計されていることは、アドレスとデータが時分割であることからも容易に想像できます。なぜなら、バースト転送においてアドレスは最初に 1 回だけ指定すれば十分だからです。これにより信号線数を大幅に抑えて信頼性の確保とコストの低減を図っているのです。逆に言うと、シングル転送ばかりでは PCI 本来の性能の数分の 1 程度しか発揮できないということになります。

● 高速バック・トゥ・バック・トランザクション

バスを効率よく使用するために、高速バック・トゥ・バック・トランザクションというバスサイクルが定義されています。

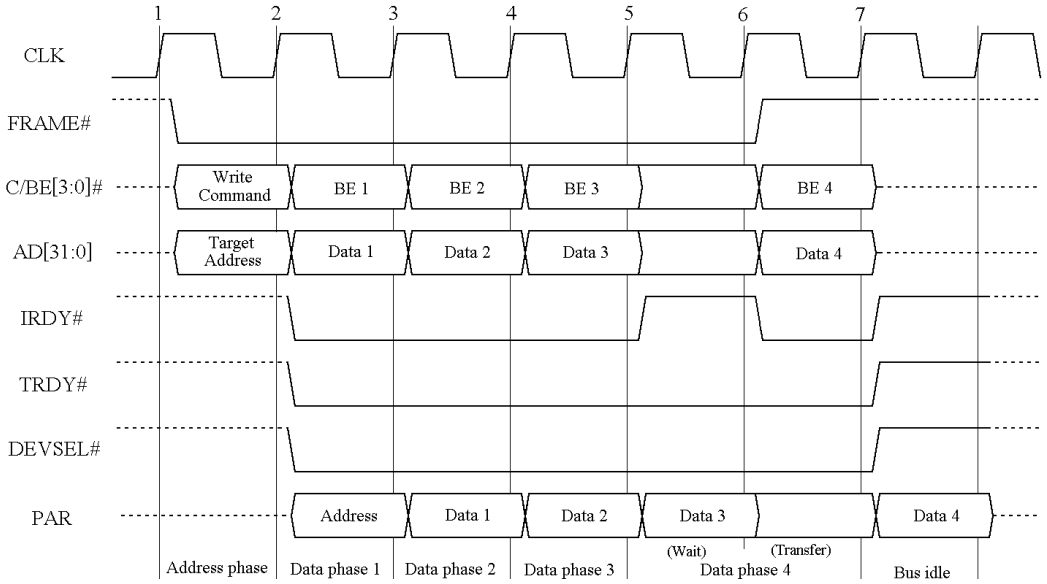
図 1 に示すように通常はそれぞれのバスサイクル(トランザクション)の間には最低 1 クロック期間のアイドルサイクルが存在します。これは出力の衝突を防ぐための緩衝期間なのですが、それぞれのバスサイクルの間でバスをドライブするエージェントの切り替えが発生しないという条件が成立する場合に限り、このアイドルサイクルを省略してバスの利用効率を上げることができます。

たとえば、一つのマスタデバイスが同じターゲットに対して連続して書き込む場合で、これは常に使用可能です。ここでは触れませんが、異なるターゲットへのアクセスにおいても一定の条件下で高速バック・トゥ・バック・トランザクションを適用することができます。

● 制御信号の動作の一般的ルール

コントロール信号の動作には一定のルールがあって、どのような場合であっても例外なく適用されます。

〈図 4〉 バーストライトサイクル（イニシエータのウェイトあり）



〈表 1〉 PCI バスコマンド一覧

C/BE[3:0]#	コマンド
ＬＬＬＬ	割り込み応答
ＬＬＬＨ	スペシャルサイクル
ＬＬＨＬ	I/Oリード
ＬＬＨＨ	I/Oライト
ＬＨＬＬ	〈予約〉
ＬＨＬＨ	〈予約〉
ＬＨＨＬ	メモリリード
ＬＨＨＨ	メモリライト
ＨＬＬＬ	〈予約〉
ＨＬＬＨ	〈予約〉
ＨＬＨＬ	コンフィグレーションリード
ＨＬＨＨ	コンフィグレーションライト
ＨＨＬＬ	メモリリード・マルチプル
ＨＨＬＨ	デュアルアドレスサイクル
ＨＨＨＬ	メモリリード・ライン
ＨＨＨＨ	メモリライト・アンド・インバリデード

◆FRAME#、IRDY#

イニシエータにより駆動される制御信号。サスティンドトライステート。

・FRAME#、IRDY#共にデアサート状態のときは何も実行されていないアイドルサイクルであり、この状態から新たなバスサイクルを開始できる。

・イニシエータがFRAME#をアサートすることによりバスサイクルがスタートする（アドレスフェーズ）。アドレスフェーズでは IRDY#はデアサート状態である。

・イニシエータはデータ転送準備ができたら IRDY#をアサートする。いったん IRDY#をアサートしたらそのデータフェーズが終わるまで FRAME#、IRDY#の状態を変えてはならない。

・最後のデータフェーズの転送準備ができたら IRDY#をアサートすると同時に FRAME#をデアサート(最終データであることを示す)する。

・一旦 FRAME#をデアサートしたら(サイクルの終了を示したら)そのサイクルの中で再び FRAME#をアサートしてはならない。

◆ C/BE#[3:0]

イニシエータにより駆動される制御信号。アドレスフェーズではバスコマンド（そのバスサイクルの種類）を示す（**表 1**）。データフェーズではデータ転送が行われるバイトレーンを示す。データフェーズで全てデアサートしてデータ転送をしないというのもあり。

◆DEVSEL#、TRDY#

ターゲットにより駆動される制御信号。サスティンドトライステート。

・サイクルにポジティブデコードで応答するターゲットは、アドレスフェーズのあと 3 クロック以内に DEVSEL#をアサートしなければならない（**図 6**）。

・DEVSEL#より先に TRDY#をアサートしてはいけない。

◆AD[31:0]

アドレスフェーズにおいてイニシエータが転送アドレスを示す。また、データフェーズにおいては転送データが乗せられる。

・メモリサイクルの場合、転送アドレスの下位 2 ビットは常に 00 として扱われ、AD[1:0]はバースト転送でのアドレスインクリメント方法を示す。

・I/O サイクルの場合、AD[31:0]全てが有効なアドレスを

示す。I/O 空間はバイト単位で割り当てられるため。

● パリティ信号(PAR)について

バスパリティの計算は C/BE#[3:0]と AD[31:0]の 36 ビットに対して行われ、その AD[31:0]を駆動するエージェントにより常に 1 サイクル遅れて PAR に出力されます。リードサイクルでは、ターゲットは受信した C/BE#[3:0]と自分の出力する AD[31:0]でパリティを計算して PAR を駆動します。

今回はパリティ機能の詳細についての説明は割愛します。

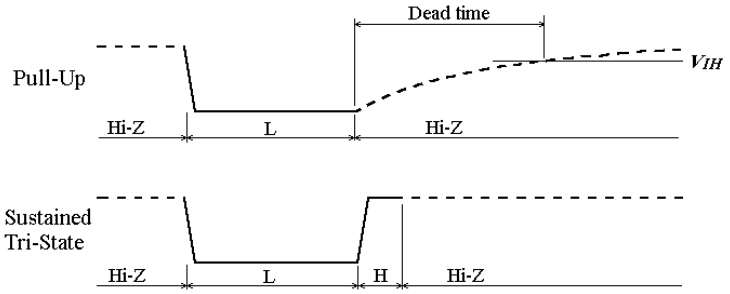
● サスティンド・トライステート

ISA のような非同期バスでは複数のデバイスにより共有されるコントロール信号(たとえば-IOCS16 信号)はオープンコレクタで駆動されていました。PCI でも共有信号がいくつかあり、誰も使用していないときはプルアップによりデアサート状態を保ちます。

しかし、プルアップだけではＬからＨに移行するのに時間がかかりすぎてしまいます。ISA のようにバスサイクルの間隔が長かったころはプルアップで問題ありませんでしたが、PCI ではサイクルタイムが最小 30ns と ISA と比べて非常に短いため、信号レベルの確定が遅れると高速動作の妨げとなってしまいます。

このため、PCI ではサスティンド・トライステートという方法がとられます。**図 5** に示すように信号を開放する際一旦 H レベルに駆動して速やかにデアサート状態に戻し、あとはプルアップにより H レベルを保つのです。非同期バスでは H と L が衝突する可能性がありますが、PCI では全てのデバイスがクロックに同期して動作して衝突が発生しないようになっています。

〈図5〉 サスティンド・トライステート



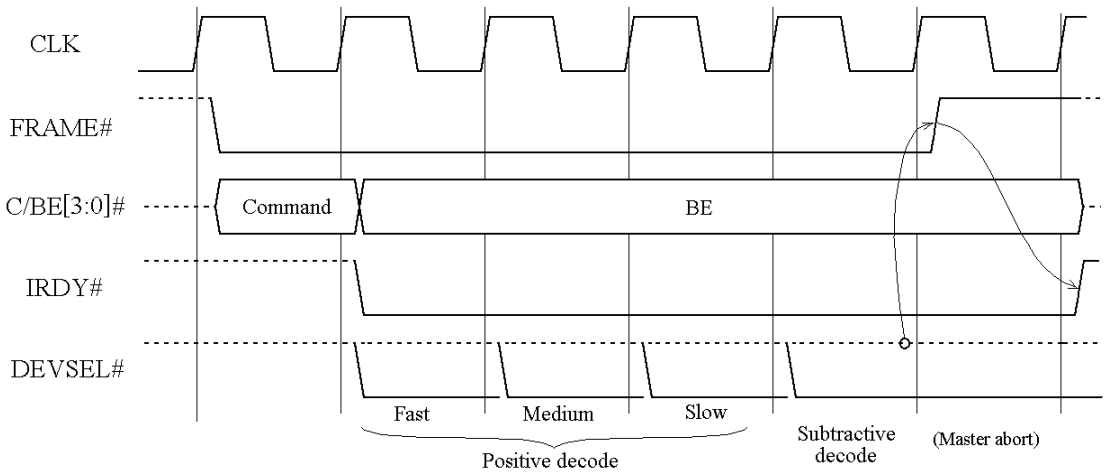
● デバイスの応答(DEVSEL#)

バスサイクルの開始により選択されたデバイスは、アドレスフェーズの後 3 クロック以内に DEVSEL#をアサートして応答しなければなりません。これをポジティブデコードといい、DEVSEL#をアサートするタイミングにより、高速・中速・低速デコードとなります(**図 6**)。

ポジティブデコードで応答するデバイスが無い場合、4 クロック目で代わりに応答するという動作があります。これをサブトラクティブデコード(誰も応答しないなら僕が)といい、この種のデバイスは一つのバス上に 1 個だけ存在できます。AT アーキテクチャのパソコンでは PCI-ISA ブリッジがこれにあたります。ISA バス上のデバイスのアドレスを知る手段がないので、そのサイクルのターゲットが PCI バス上に無いのなら、ISA デバイスであると仮定して、とりあえず ISA バスにバスサイクルを発生させるしかないからです。当然ですが、ISA バスにとって無効なアクセスには応答しません(16MB 以上のメモリアドレス等)。

4 クロック目で誰も応答しなければそのバスサイクルは失敗します(マスタ・アボート)。

〈図 6〉 デバイスの応答タイミング



バスステートコントローラの動作にょ★

今回はとりあえず自作ボードを PCI で動かしてみるという実験が目的ですので、製作するインターフェース形式は単純なパラレル I/O とします。

PCI は完全な同期バスですので、全ての動作の単位はクロックを基準にしたものとなります。製作する拡張ボード(ターゲットデバイス)にはイニシエータからのコマンドを解釈して順次実行するステートマシンの概念が必要になってきます。このボードのバスコントローラ(pci_pio.abl)では各ステート番号を**図 7**のように定義しています。

◆ STATE 4: バスアイドル

この状態から新たなバスサイクルの始まりを検出します。このデバイスに対して I/O サイクルが開始された場合は、STATE<2>または STATE<5>に移ります。それ以外のバスサイクルが始まった場合は、STATE<0>に移行します。

◆ STATE 0: バス非アイドル

実行中のバスサイクルが終了してバスがアイドル状態になるまで待ちます。この状態からは新たなサイクルには移行しません。

◆ STATE 2: ライトサイクル実行中

DEVSEL#をアサートしてライトサイクルに応答します。同時に TRDY#をアサートしてデータ受信準備ができたことをイニシエータに知らせます。この状態で IRDY#がアサート(ライトデータが準備 OK)で、なおかつ FRAME#がデアサート(最終データフェーズ)の場合は、次のクロックで STATE<3>に移行して AD[31:0]のうち C/BE#[3:0]で示される有効なバイトレーンのデータをレジスタにラッチます。

◆ STATE 3: ライトサイクル終了

DEVSEL#と TRDY#をディアサートして、最終データを出力レジスタにラッチします。通常は次のクロックで DEVSEL#と TRDY#を開放してアイドル状態に移行しますが、この状態でこのデバイスに対して I/O サイクルが開始された場合には、アイドル状態を経由せずに直接 STATE<2>または STATE<5>に移行します(高速バック・トゥ・バック・トランザクション)。

◆ STATE 5: リードサイクル開始

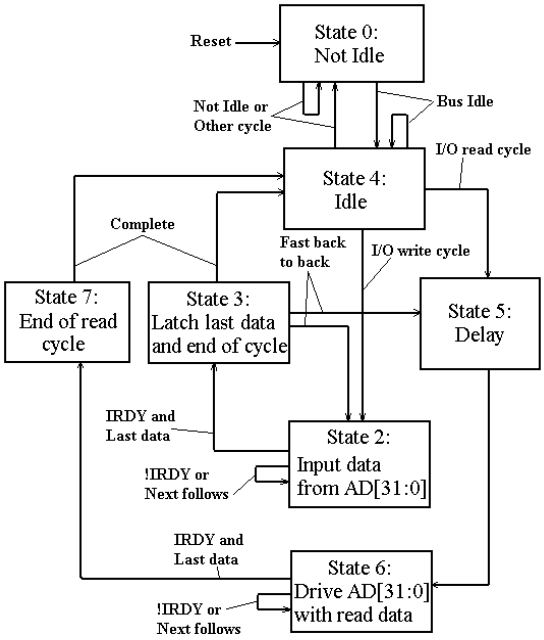
リードサイクルではコマンドフェーズとデータフェーズの間で AD[31:0]をドライブするエージェントの切り替えが発生するので、バスの衝突を避けるために必ず 1 クロックの緩衝期間を取らなければなりません。

次のクロックで STATE<6>に移行してリードサイクルに応答します。

◆ STATE 6: リードサイクル実行中

DEVSEL#をアサートしてリードサイクルに応答します。同時にリードデータを AD[31:0]に乗せて TRDY#をアサートしてデータの準備ができたことをイニシエータに知らせます。この状態で IRDY#がアサート(イニシエータが受信準備 OK)で、なおかつ FRAME#がデアサート(最終サイクル)の場合は、STATE<7>に移行します。

<図 7> 本ボードでのステートダイアグラム



◆ STATE 7: リードサイクル終了

AD[31:0]を開放して DEVSEL#と TRDY#をデアサートします。次のクロックで DEVSEL#と TRDY#を開放してアイドル状態に移行します。

◆ STATE 1: 未定義

このステート番号は定義されていません。もし誤動作によりこのステートに入ってしまっても、次のクロックで STATE<0>に移行して正常ループに復帰します。

規格破りだにょ！

さて、この製作では PCI デバイスとして要求される機能を一部省くことにより回路を単純化して再現性を確保しています。今回の実験にあたり省いてしまった機能とそのことによる弊害について説明しておきましょう。ちゃんとしたPCIボードを設計しているエンジニアが見ると間違い無く卒倒するような内容ですが、パソコン上で実験的に使う程度なら特に問題になることはないでしょう。

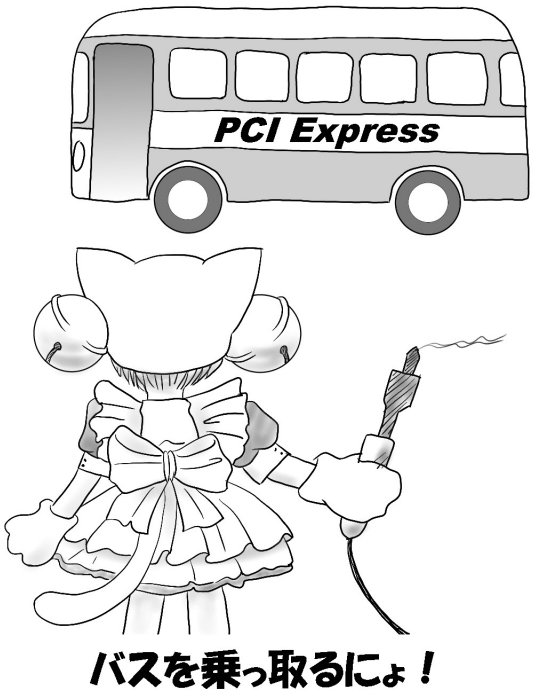
● コンフィグレーションレジスタが無い！

PCI ではコンフィグレーションレジスタを実装することによりデバイスの存在がシステムに認識されます。この機能

により PCI デバイスとしての機能を設定したり、衝突が発生しないようにデバイスアドレスをリロケートしたりできるようになっています。

しかし、このボードでは**コンフィグレーションレジスタを実装せずにアドレスを固定的にデコード**しています。デバイスの存在もシステムに認識されないので、バスの衝突が発生しないようにI/Oアドレスを設定してやる必要があります。つまり **PCI バスを勝手に乗っ取る**わけです。ディップスイッチでアドレス設定する PCI ボードなんて笑えますね(笑)。

ただし、その時点でアドレスの衝突が無かったとしても、あとで他の PCI デバイスがリロケートされてきて衝突する可能性もあります。まぁ、そうクルクル変わるものではないので大丈夫だとは思いますが。



バスを乗っ取るにょ！

● パリティ生成・チェック機能が無い！

PAR 信号のチェック及び駆動をしませんので、**リード動作ではイニシエータによりパリティエラーが検出され**ます。イニシエータ(この場合はホストーPCI ブリッジ)のコンフィグレーションレジスタのコマンドレジスタ(byte4)のビット6(パリティエラー応答ビット)が立っている場合はパリティエラーを検出するとそれに応じたアクション(NMI など)が発生します。しかし、パソコンではこのビットが 0 に設定されているのが普通なので、パリティ機能を実装しなくても動作に影響はありません。

● アドレスの上位 16 ビットをデコードしない！

PCI では I/O デバイスであっても AD[31:0]をフルデコードしなければなりませんが、このボードでは下位 16 ビット(AD[15:0])しかデコードしていません。でも、ATアーキテクチャのパソコンで使うぶんには 64K バイト以上の I/O 空間に対するアクセスが発生することはないので、特に問題にはなりません。

制作だにょ！

● 部品を集める

まずは部品を揃えましょう(**表 2**)。PCI 用のユニバーサルボードにはサンハヤトの MCC-331 (¥5,000 程)を使用しました。このボードにはマウント金具も付いているので、なかなか使いやすいです(写真 1)。

PLD は 7ns 以下の物を使用してください。実はバスタイミングの規格を厳密に適用すると 0ns でないと間に合わないのですが、そんなものは入手不可能ですのでマージン頼りで使用しました(^^;(まぁ、実際はこれで全然問題無いですけどね)。

それ以外の部品はごくありふれた物ですので、メーカー等まで選定する必要はないでしょう。入出力は、それぞれ 32 ビット欲しいところでしたが、今回はコネクタの**実装スペースの問題で、それぞれ 16 ビット**としました(高密度コネクタの手持ちが無かったというのもありますが)。回路図の破線内のパーツは載せていません。32 ビット欲しい場合は回路図のパーツを全て実装してください。

● 組み立てる

本ボードの回路図を**図 8** に示します。単なるディジタル回路なので、組み立てもそれほど気を使うことはないと思います。ただ、なにぶん配線の本数が多いので、**ラッピングワイヤだと線材が山盛り**になって収拾がつかなくなってしまいます。高密度配線には UEW(ウレタン線)が最適です。

そして何よりいちばん注意しなければならないのは、**配線を間違えないこと**です。エッジコネクタのキーの部分で端子番号が不連続になっていますので、この辺りで間違えやすいです。実際わたしも何度か間違えました(笑)。特に電源端子の間違いは M/B を壊す恐れがありますので十分に確認してください。

また、最大 33MHz とかなり高速動作しますので安定動作のため、**電源ラインは十分に強化**しておいたほうが良いでしょう。信号線もなるべく最短で配線できるようにパーツのレイアウトをよく考えてください。

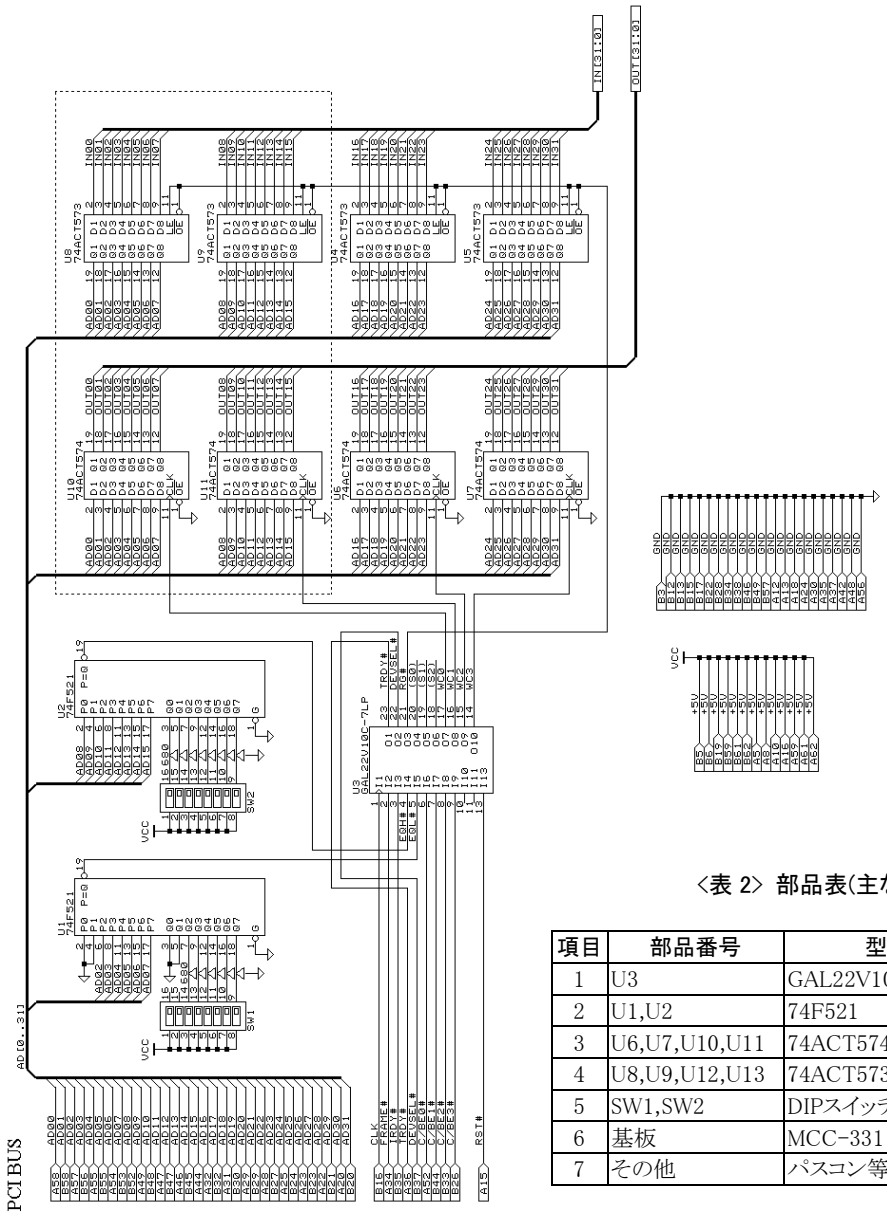
火入れだにょ！

● アドレスの設定

まず最初に壊れてもいいM/Bを用意してください(笑)。まぁ、間違い無くちゃんと配線できていれば M/B を壊す心配はないのですが、電源系の配線を間違っていたりすると最悪 M/B が即死する可能性もあります(12V と 3.3V をつないじゃったとかね(^^;)。

ボードをセットする前に**ディップスイッチでこのボードの I/O アドレスを指定**します。このボードはアドレスの下位 2

<図 8> PCI パラレル I/O ボードの回路図



<表 2> 部品表(主なもの)

項目	部品番号	型番	数量
1	U3	GAL22V10B-7LP	1
2	U1,U2	74F521	2
3	U6,U7,U10,U11	74ACT574	4
4	U8,U9,U12,U13	74ACT573	4
5	SW1,SW2	DIPスイッチ	2
6	基板	MCC-331	1
7	その他	パスコン等	

ビットをデコードから外しているので、連続した 4 バイト(バス幅分)の I/O 空間を占有することになります。このとき他の PCI デバイスと I/O アドレスが競合するとバスが衝突しますので、誰も使わない I/O アドレスを選ばなければならないのは言うまでもありません。

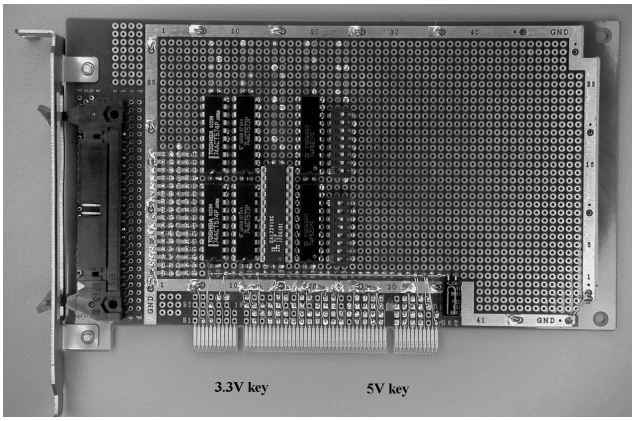
ISA バスでは*300h～*30Fh が試作ボード用に開放されていますので、この辺りを使うといいでしょう。例えば、ディップスイッチで F300h を設定すると、F300h～F303h を占有します。つまり、この範囲への I/O アクセスには、このボードが応答するというわけです。しかし、このアドレスが空いているとは限りません。念のため Windows のデバイスマネージャ等でチェックして何も使用していないことを確認し、そのアドレスを予約して他のデバイスが割り当てられないようにしておいた方が良いでしょう。

● 電源を入れる

電源を入れてみて画面が表示されたらとりあえず M/B は生きています。もしも BIOS の起動画面が出なかったら急いで電源を切って調べてください。無事パソコンが起動したら、デバッガ(DEBUG.EXE 等)でポートを読み書きして動作を確認します。今回は回路図の破線内を省略して AD[31:16]だけに I/O レジスタを接続しているので、F302h～F303h が有効な I/O アドレスとなります。F300h～F301h への書き込みデータは失われ、読み出しでは無意味な値が読まれます。当然ですが、32 ビット分のレジスタをフル実装していれば F300h～F303h 全てが有効となります。

〈写真 1〉　製作した PCI パラレル I/O ボード

使用したボードは MCC-331(サンハヤト)。このボードは 3.3V/5V 両用なので、電圧キーの切り欠きが 12,13 と 50,51 の両方に入っている。切り欠きの部分で端子番号が不連続になっているので、間違えないように注意。引き出しランドの並びに規則性が無いのでパターンを追って確認するほうが良い。



最後にょ★

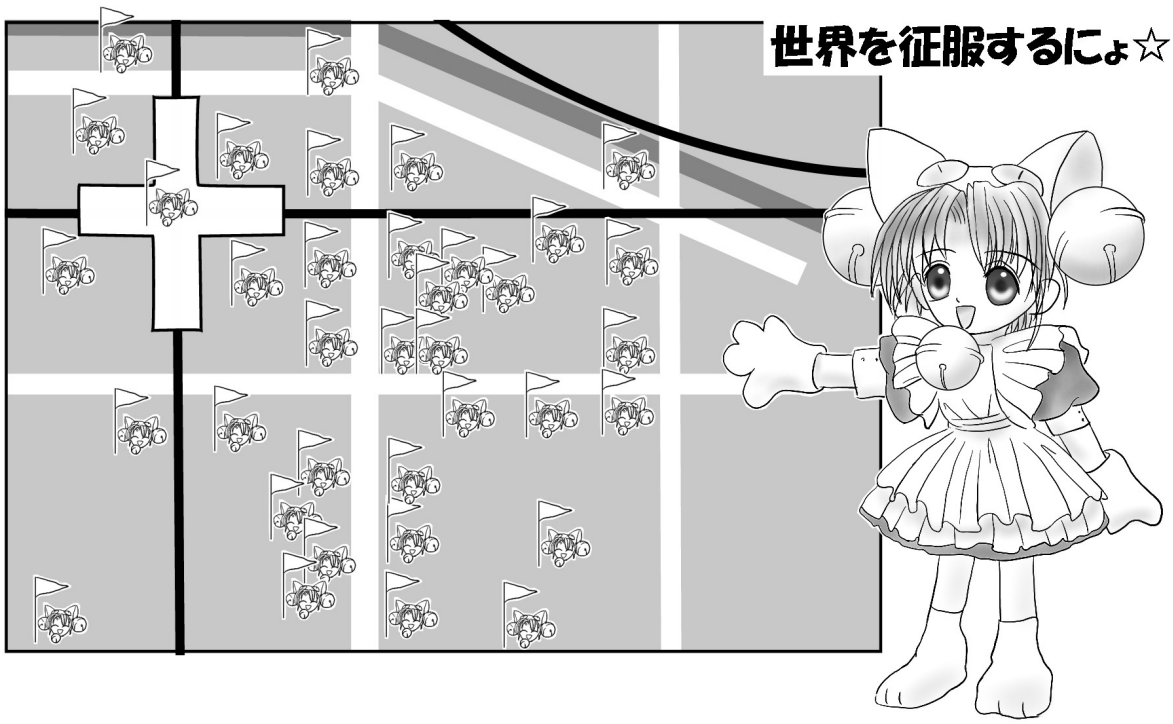
これまで PCI が難しいと思っていて、「こんなに簡単なの？」と驚かれた方も居るかと思います。これにより PCI バスも意外と手軽に遊べるということが分かるでしょう。もちろん、この記事で示すような回路は PCI の規格に準拠した物ではないので、その利用は実験の範疇に留まります。

最近は、FPGA メーカ各社から発売されているデバイスを使って PCI 規格にフル対応したボードを自作することができるようになってきました。デバイスの販売合戦のためフリーで使える PCI 周りの IP も同時に配布されています。FPGA の普及に伴い、今後はアマチュアにとっても PCI を利用した電子工作はより身近な存在になってくることでしょう。

PCI に興味を持ち、もっと詳しく知りたいのなら、次に示す文献をおススメします。とても良く解説されているので、PCI を理解して実際に PCI デバイスを設計するにはこれだけでとりあえず十分といえるくらいの資料です。私もこの実験にあたり、ほとんどの知識をこの文献から得ました。

OPEN DESIGN No. 7
PCI バスの詳細と応用へのステップ、 CQ出版

※使用した ABEL ソースファイル等(pci_pio.zip)は、ftp://elm-chan.org/pub/ で公開しています。



InterBase の紹介

ROM 男 (cma59466@biglobe.ne.jp)

１．はじめに

オープンソースは RDBMS の世界にも広がってきました。そして InterBase 6 が Mozilla Public License に準拠した形で無償利用可能となったので紹介します。

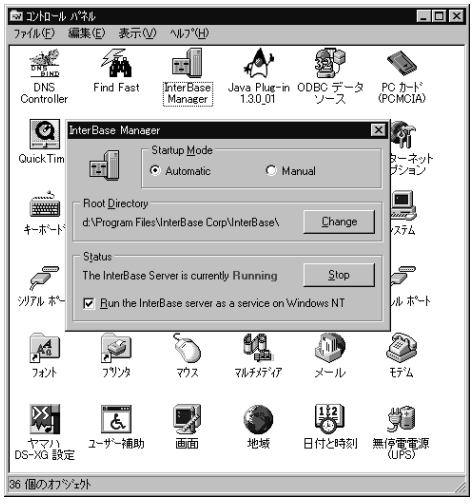
２．InterBase の入手法

InterBase 6 は inprise のサイト (http://www.inprise.com/interbase/downloads/) から手に入ります。マニュアルは interbase のサイト (http://www.interbase.com/) に置いてありますが、β版なので一部未完成です (ftp://ftp2.interbase.com/pub/products/beta6.0/ib_b60_doc.zip)。

ファイル名	タイトル	内容
ApiGuide.pdf	API Guide	
DataDef.pdf	Data Definition Guide	データベースの設計（デザイン・データの取り扱い方法・セキュリティの考え方等）について書いてあります
DevGuide.pdf	Developer's Guide	C++ や Delphi などを使って InterBase と連携するソフトを開発することなどについて書いてあります。現時点では ODBC の使い方の部分が空白になっています
EmbedSQL.pdf	Embedded SQL Guide	プログラム言語に SQL を埋め込む方法について書いてあります
GetStart.pdf	Getting Started	インストールの仕方と旧バージョンの InterBase システムの移管方法について書いてあります
Langref.pdf	Language Reference	主に SQL 文の文法について書いてあります
OpGuide.pdf	Operations Guide	データベースの管理・ネットワークの管理・セキュリティ管理等について書いてあります

３．InterBase のインストール

ファイルは zip 形式でアーカイブされているされているので、適当なツールで展開し、setup を実行し、setup からの質問に対して自分の環境にあうように答えればインストールが完了します。アプリケーションには isql のようなコマンドライン版 (DOS 窓の環境で実行するもの) の tool がありますが、インストーラーは環境変数に isql などのディレクトリを追加してくれないので手動で対処しなければなりません。インストールが完了すると、コントロールパネルに "InterBase Manager" が出来ます。ここから InterBase の稼働状態を変更することもできます。



４．SQL を入力する

４．１．InterBase を使うためのツール

InterBase には次のようなツールが付属しています。

名前		内容
IB console	GUI	GUI 版のコンソールで、DB の作成や削除・ユーザー管理・以下のツールの実行などが行なえます
isql	コマンドライン	コマンドラインから入力される、あるいはあらかじめファイルに記述した SQL 制御文を実行するためのツールです
gsec		新規ユーザーの作成・削除・アクセス権限の付与といったユーザー管理を行なうツールです
gpre		プログラムなどに SQL 制御文を埋め込むときに使うプリプロセッサです
gfix		壊れたデータベースファイルを修復するためのツールです
gbak		バックアップを作成するためのツールです

４．２．isql を使って InterBase を動かす

４．２．１．InterBase 6 で扱えるデータ

データ型	サイズ	範囲／精度	詳細
BLOB	可変長	なし (セグメントサイズは 64k まで) グラフィクスやテキスト、音声データ等のための動的にサイズが変化できるデータ型セグメントを単位とする構造をもつ サブタイプを持ち、それにより BLOB の内容を示せる	
CHAR(n)	n 文字	1～32767byte	文字サイズが固定長の文字列を扱う
DATE	64bit	西暦 100 年 1月 1 日～西暦 32768 年 2 月 29 日　ISC 日付	
DECIMAL(精度,スケール)	可変長 (16, 32, 64bit)	精度：全桁数の意味 1～ 18　スケール：小数点以下の桁数 1 ～ 18 (ただし全桁数以下であること)	
DOUBLE PRECISION	64bit	$\pm 2.225 \times 10^{-308}$ ～$\pm$ 1.797×10^{308} IEEE の倍精度で 15 桁が有効	
FLOAT	32bit	$\pm 1.175 \times 10^{-38}$ ～ $\pm 3.402 \times 10^{38}$ IEEE の単精度で 7 桁が有効	
INTEGER	32bit	-2,147,483,648 ～ 2,147,483,647	
NUMEIC(精度,スケール)	可変長 (16, 32, 64bit)	精度：全桁数の意味 1～ 18　スケール：小数点以下の桁数 1 ～ 18 (ただし全桁数以下であること)	
SMALLINT	16bit	-32768~32767	
TIME	64bit	0 時 0 分 0 秒 ～23 時 59 分 59 秒9999　ISC 時間	
TIMESTAMP	64bit	西暦 100 年 1月 1 日～西暦 32768 年 2 月 29 日　時間情報も含む	
VARCHAR(n)	n 文字	1～32765byte	文字サイズが可変長…日本語や中国語の様に 1Byte のものと 2Byte 以上の符号が混じるような…の文字列を扱う。

４．２．２．データベースファイルを新規作成あるいは既存 DB に接続する

isql を管理者権限で起動 (`-user SYSDBA -pass masterkey` というオプションをつける)

し、プロンプトから以下のように打ち込むと 'book_list.gdb' というデータベースファイルを作成できます。

```
SQL>CREATE DATABASE  book_list.gdb;
```

ユーザー名やパスワードを指定してデータベースを作れます。管理者権限でデータベースを作成する場合などに使えます。

```
SQL>CREATE DATABASE book_list.gdb user 'SYSDBA' password 'masterkey';
```

既存のデータベースファイル名を指定して CREATE DATABASE を実行してもエラーになります。新しくファイルを作成したいときにはファイル名を変えるか既存のファイルを削除しなければなりません。既存のファイルを使いたいときは CONNECT で接続します。

```
SQL>CONNECT book_list.gdb;
```

４．２．３．テーブルを作成する

CREATE TABLE 文を使えばデータベースファイル中に表を作成できます。isql のプロンプトから以下のような命令を打ち込む、あるいは以下のようなファイルを作っておいてファイル取り込みオプションをつけて isql を起動するとテーブルを作れます。

```
SET NAMES SJIS_0208;
CREATE DATABASE "booklist.gdb";
CREATE TABLE COMPANIES
(
       company_code  CHAR(3)       NOT NULL,
       company_name  VARCHAR(60)   CHARACTER SET SJIS_0208 NOT NULL,
       PRIMARY KEY(company_code)
);

CREATE TABLE books
(
       book_code     CHAR(4)       NOT NULL,
       book_name     VARCHAR(60)   CHARACTER SET SJIS_0208 NOT NULL,
       company_code  CHAR(3),
       price         INTEGER,

       PRIMARY KEY( book_code ),
       FOREIGN KEY( company_code ) REFERENCES COMPANIES( company_code )
);

COMMIT;
QUIT;
```

```
C:\>isql -i 上記のような定義ファイル
```

４．２．４．テーブルを編集する

INSERT, DELETE, UPDATE コマンドを使うことで表中にデータを挿入／削除／表中のデータを更新できます。

```
SQL>INSERT INTO 表名 (列名1, 列名2, …) VALUES (値1, 値2, …);
```

この文の場合、列名の並びが宣言されているので CREATE TABLE で宣言した順とは無関係に値を指定できます。

```
SQL>INSERT INTO 表名 VALUES (値1, 値2, …);
```

この文の場合、列名の並びが宣言されてないので CREATE TABLE で宣言した順に値を指定しなければなりません。

```
SQL>DELETE FROM 表名 WHERE 条件;
SQL>UPDATE 表名 SET 列 1 = 値1, 列2 = 値2… WHERE 条件;
```

なお変更した情報は COMMIT 命令を実行して初めて表に反映されます。変更した情報を反映したくない時は ROLLBACK 命令を実行します。

４．２．５．テーブル中のデータを検索する

SELECT 文を使ってデータを検索することができます。省スペースのために正規表現的な表現をしてみにくいことおわびします。isql のプロンプトから SELECT 文を打ち込めば表示されます。

```
SELECT [列名|列名, 列名…|*|ALL|DISTINCT] FROM 表名
   [WHERE 検索条件|        /* 検索条件が成り立つ列を表示します */
   GROUP BY [列名] HAVING 検索条件]
      /* 先に GROUP BY で指定した列でまとめたうえで検索条件に適合した列のみ表示します。InterBase の場合、GROUP BY で指定する列と SELECT の次の表示を指定する列が等しくないとエラーになる場合があります（そうならない RDBMS もあります）
      */
   [ORDER BY 列名 [ASC|DEC] 列名 [ASC|DEC]];
      /* ORDER BY は列名の要素の内容に従って表示の順序を変更します。ASC と書くと昇順（省略可能）で DEC は降順です */
```

５．最後に

ページ数の都合上、世の Web Site で公開されている SQL のさわりにも及ばぬ程度のことしか書けませんでした。InterBase に関する本は数も少なく、世の中は Oracle か Postgre SQL しかないような雰囲気ですが、参考文献に示したように InterBase に関するものもいくつか存在します。 InterBase にも興味をもってもらえればありがたいと思います。

６．参考文献

1	データベース Linux InterBase 4.0 For Linux	インプライズ株式会社　監修 下田雅彦　著	アスキー出版局 ISBN4-7561-2013-X	￥4500+税
2	データベース Linux Java Servlet による Web サーバー構築編	インプライズ株式会社　監修 加藤大受・田原孝　著	アスキー出版局 ISBN4-7564-3379-7	￥3800+税
この二つは古いバージョンだが InterBase を実際に使った例が書いてあり、非常に役に立つ。但し、InterBase 6 は上記の内容から拡張されている部分があるのに注意したほうがよい				
3	初心者のための SQL 入門 (1)はじめてのデータベース操作	北本隆・渋井俊昭　著	翔泳社 ISBN4-88135-870-7	￥2600+税
取り扱っている SQL の内容は非常に簡単。Oracle 8i Personal のお試し版 (30 日間利用可能) がついており、手持ちの PC にインストールすれば SQL を試せるようになっている				

変更履歴

2000年12月30日　コミケット59頒布版（東京ビッグサイト）
表紙モノクロ、50P オフセット誌、頒布価格 700 円
【完売しました】

2002年2月10日　インターネット配布版 Revision 0
原稿削除　インターネット配布版掲載辞退の方のページを削除
関連する目次、あとがき分を削除

著者あとがき(50音順)

Kimさん

硬派な記事の中で、僕のおバカ記事はきっと浮いている事でしょう…。みなさん、ごめんなさいねー。DOS／Vワールドは常連の方がお互い信頼しあい、それでいて排他的にならず、とてもクールなSIGでした。記念本に参加させてくださってありがとうございました。では、また～。

きよりんさん

最近のマザーボードはフラッシュ ROM が大容量化して 4メガビットの PLCC タイプのものが多く使われてきています。こうなると ROM の入手も難しく、「ROM 焼きだいじょうぶ」という市販品を使わざるをえません。でも、ハード派としてはなんかつまんないですね。

ChaNさん

DOS/VワールドSIGOPのChaNです。いつか来ることとは思っていたものの、いざその時が来るとなると複雑な心境です。インターネット全盛となりPC−VANがその役目を終えたとはいえ、いままで慣れ親しんだものが消えるとなるとやっぱり寂しさを感じますね。でも、これも時代の流れということで前向きに考えないと。というわけで、今後はweb上で活動を続けますのでよろしゅうに。

ROM男さん

東京に行く機会が出来て以来、コミックマーケットに一般参加するようになりましたが、今回初めてサークル参加することになりました。個人的には異動のどたばたがあり、また、たいした内容のものも書けないなど完全燃焼とは行きませんでしたがまた機会があったら参加したいと思います。

もろぼし☆らむ

今回の本の企画の言い出しっぺ&編集人です。本当はコピー誌の予定がなぜかオフセットに…。なにしろはじめてのオフセット&データ入稿のため、不慣れでライターの皆様には迷惑かけっぱなしでしたが、ちゃんと本ができてよかったです（これを書いている時点は違うけど(^^;)。記念本でもありますので、表紙はるりあ046先生にお願いしちゃいました。うーん、贅沢(笑)忙しい中かわいい表紙を描いてくれて感謝です(^^)
とわいえ、来年でPC-VANも消滅（；＿；） 最後にこの企画で花を咲かせることができたのがせめてもの救い・・。でも、夏もなんか出せたらいいな～(^^;;; 次はDSPネタかな。

奥付

発行	どすぶいだにょ☆
発行年月日	2000年12月30日(コミケット59頒布版) 2002年2月10日(インターネット配布版)
印刷	(株)POPLS(コミケット59版)
著者	PC-VAN DOS/V SIG メンバーs
編集・発行責任者	もろぼし☆らむ
連絡先	http://homepage2.nifty.com/~maid/ maid@nifty.ne.jp
配布元	http://elm-chan.org/

This Page is Blank.


DOS/V World ★ Final Project
AVR