どすぶいだにょ☆

第2号

This Page is Blank.

もくじ

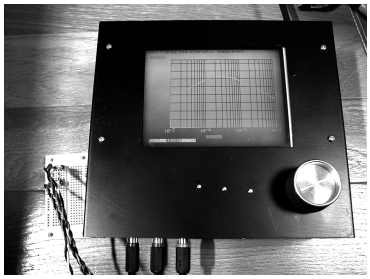
表紙：ChaN
裏表紙：もろぼし☆らむ

DDS とデジタル液晶を使った電圧比計測器の設計/製作

おやう

1. はじめに

アナログ回路の製作中に，製作したフィルタ回路が設計どおりの働きをしているのかを確認したいことがあります．そのとき，ｆジェネとオシロを使って入出力レベル比の測定を行い，位相差を調べ，グラフにプロットするという作業が必要になります．これを自動化することにより手を抜こうというのが本稿の目的です．ただし，ここで製作する機器は校正等されないため，測定器ではないことを認知しておいてください．位相差に関しては現段階では測定できません．将来的に対応するつもりですが….



2. 回路概要

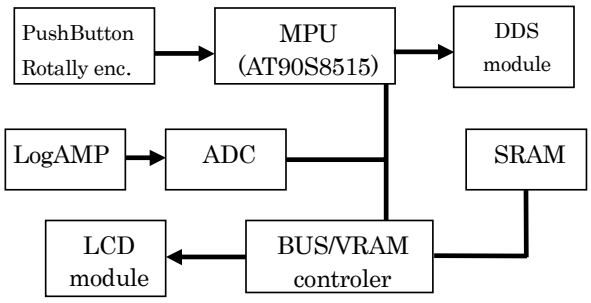
全体ブロック図を**図 1** に示します．マイコンによって DDS[1]を制御して任意の周波数の正弦波を出力させ，その信号を計測したい回路の入力に流し込みます．出力されてくる信号レベルを ADC[2]でデジタル値として取り込み，入出力比を画面に表示します．ここで ADC の前にはログアンプを設置し，幅広いダイナミックレンジを確保しています．

DDS は秋月電子にて購入したもので，そのまま使用し，出力バッファには CLC452 を使用しています．

LCD は制御信号やドットデータをタイミングどおりに送る必要があるため，CPLD にて制御します．マイコンとの接続は，AVR シリーズで外部バスを持った AT90S8515 を選択し，データメモリ空間中の外部 RAM への書き込みで画素の色データを書き込みます．I/O ポートへぶら下げてもいいのですが，mov 命令でサクっとデータが転送できたほうが便利かと思い，このような構成にしました．

Log-AMP は入力信号を対数化してくれる AD8307 を用います．本機で使用されている主な部品を表 1 に示します．

図 1 全体ブロック



[1] DirectDigitalSynthesizer：
デジタル制御による周波数発生器

[2] AnalogDigitalConverter:
アナログ信号をデジタル値に変換する石

2.1 AT90S8515 インタフェース

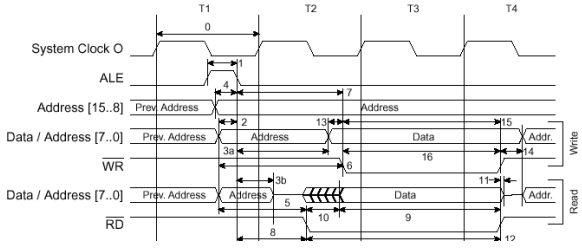
マイコンのピンアサインを見ればわかるように，アドレスの下位とデータが同一ピンから出ているのが判ります. これは使用ピン数を減らすためのものであり，外部で下位アドレスをラッチしておく必要があります. ラッチするトリガとなる信号線は ALE で，この立ち下がりで下位アドレスを保持します．RD,WR などは一般的なマイコンのそれと同じです. また，外部 RAM アクセスにウェイトを入れるかどうかの設定は，マイコンの設定レジスタで行います．実際に回路設計を行う際には十分に Setup/Hold Time を満足するか吟味する必要がありますので，このあたりのタイミングには注意しておく必要があります．参考に**図 2** にマイコンの外部アクセスタイミングを示します．詳細な時間等はデータシートをご覧ください.

ユーザ入力となるプッシュスイッチ，ロータリエンコーダと DDS モジュールの制御のための出力ビット

表 1 主要部品一覧

名称	型番	メーカ
1 チップマイコン	AT90S8515	ATMEL
CPLD	XC95108-10	Xilinx
CMOS	74HC541	Any
レベルコンバータ	ADM232A	Any
ADC	TC35094	東芝
LogAMP	AD8307	アナデバ
DCDC 用 IC	TLC497ACN	TI
SRAM	IS61C1024	ISSI

図 2 AT90S8515 外部アクセスタイミング



は汎用 I/O ポートに接続します. ADC は外付けしますが, 取り込みに高速なパラレル出力タイプを使います. 541 を介してマイコンのデータバスに挿しておき，データ出力タイミングを CPLD に作らせます.

2.2 LCD 信号タイミング

この LCD への入力信号は，V-Sync，H-Sync，DATA(9 本)，enable，DCK の 13 本です．液晶とはいえ，テレビと同じように画面上部より水平走査を逐次行っていくように考えていただければいいかと思います. テレビと異なるのはノンインタレース[3]方式ということで，さらに判りやすいでしょう.

モジュールということで,DCK,V/H-Sync 間のタイミングとデータの送り出しや各信号の Setup/Hold Time を守れば簡単に使えるようにできています.

それでは，簡単に各信号タイミングを紹介しつつ，水平方向のクロック数や垂直方向のライン数などを決定していきましょう．未表示部分は可変となりますので，使用者側で決定する必要があります.

⇨ DCK

DCK は DotClocK の略で, 液晶の一ドットごとのデータを運ぶ最小単位の信号です．タイミングチャートを見れば判るように，この信号を基準として考えていきます.

本機では, マイコンのクロックが 7.372MHz[4]であり, 規定内に収まっているのでこれをそのまま流用します.

⇨ H-Sync

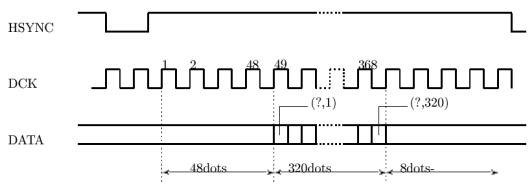
水平同期信号のタイムチャートを**図 3** に示します. この信号は水平方向のデータの開始(終了)タイミングを知らせるために使われます.

Hsync 信号の立ち上がりから 48 クロック送り，その後で有効ドット数分データを送りつけます．最後に 8 クロック以上送って一行終了です．したがって，一行あたり以下のクロック数分はカウント出来なければなりません.

$$4+48+320+8+\alpha=380+\alpha$$

ここで，αは任意の数値ですので，この値が 2 の累乗の和になるように設定するとして，α=4 とします.

図 3 水平方向タイミング

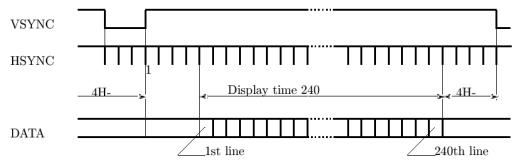


[3]一画面を一度の垂直走査で行う．テレビは二度に分けている.
[4] RS-232C で通信を行う場合，この周波数を 2 で割っていくと 115.2kbps のクロックが得られるため.

⇨ V-sync

次に垂直方向のタイムチャートを**図 4** に示します. 先ほどの H-sync を数えて，V-sync を作れば良さそうです．H-sync を 3 区間分，V-sync をアサートし，次の 4 区間は空回しし，次から一行目となります．240 行分のデータを送り終えると 4 区間以上の空回しが必要となります．そこで，キリのいい 255 とします.

図 4 垂直方向タイミング



⇨ **規格確認**

以上でタイムチャートを決定することができました．が，これに加えてフレームレート[fps]（1 秒間に画面を書き換える回数）も規定されており, 50～80fps ということです．前述した値での fps を計算すると,

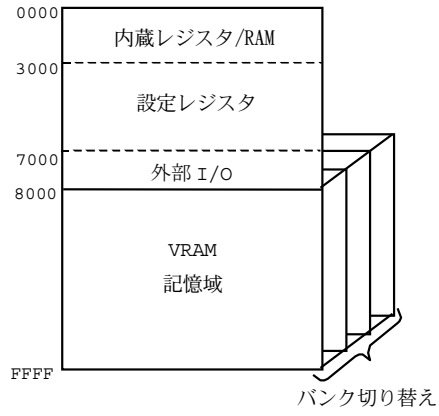
$$7.372[MHz]\div(385[DCK]\times256[line])\cong74.8[fps]$$

となり，規定値に収まっております．もし，規定外になった場合は，クロックを替えるか，ドット数やライン数が確定していない部分を加減して規定内に収まるように調整します.

2.3 メモリまわり仕様

今回製作したシステムのメモリマップを**図 5** に示します．下位アドレスには内蔵 I/O や設定レジスタなどが配置され，メモリ空間の上位半分(32Kbytes)をバンク切り替え方式として SRAM と接続しました．これは，下位アドレスに割り当てられている内蔵レジスタ等と重複するアドレスを使用した場合，ソフト作成時に余計なコードを書く必要が出てくると考えられたか

図 5 メモリマップ



らです．この配置ですと，最上位 bit が 1 であれば外付け SRAM へのアクセスとなるので,簡単そうです.

切り替え方法は CPLD 内に配置した設定レジスタへのバンク値書き込みで行います．この SRAM は LCD の VRAM として使用されます.

VRAM に必要なメモリは, 1Byte/pixel としたため, 0x12C00 バイトを要しますが､1MbitSRAM を用意したので 0xD400 バイト余ります．余った SRAM はデータ保持に使えるので，計測したデータを保持するのに有用です.

2.4 CPLD 設計

ここで今までに決定したタイミングで LCD へアクセスするような回路や，CPU と SRAM,バス接続された I/O 間の調停を行ってくれる回路を作ります．Xilinx 社の XC95108 を用い, HDL による回路設計を行います．汎用ゲートを組み合わせているとかなりの石の数になりますが，これひとつで検討してきた動作をする回路を実装することができます.

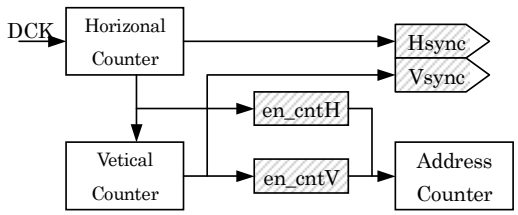
⇨ **LCD 制御部**

図 6 に制御部の同期信号用カウンタ・同期信号出漁・データ読み出し用アドレスカウンタ間の関連図を示します．網掛けの部分は F.F.で，それぞれの矢印の始点のデータを参照して H/L レベルを決定します．これら条件判定等の結果は，すべての F.F.(カウンタ含)が同じクロックで動いているため，それに同期してラッチ，出力されます.

En_cnt[H,V]は，CPLD 内の信号で，水平・垂直方向のカウンタの値が有効画素の範囲内であることを示しています．これを用いて VRAM のアドレスをインクリメントするかしないかを判定しています.

H/V-sync はそれぞれのカウンタの値が同期信号出力の開始／終了時に set/reset するようにします.

図 6 LCD 制御部 データフロー



⇨ **バスアービタ**

SRAM はシングルポートなので CPU と CPLD からの同時アクセスができません．そこで，同時にアクセスが無いように制御する必要があります．ここでは簡単のため, CPU アクセスを優先することにします．衝突を回避するだけの仕事しかしていませんが，格好良くアービタと名乗らせていただきました(笑)．もう少し上等なメモリコントローラを持った石なら WAIT 信号を持っていたりして，外部空間のアクセスに遅延を持たせられるのですが.

CPLD が LCD へデータ転送中に，マイコンから SRAM へアクセスが入るとそのデータ転送を優先し，LCD へ送るべきデータの取り込みは処理しません．ところが，ドットの出力データはレジスタでラッチをかけてから出力しているので，一つ前のドットと同じ色の出力を維持します.

下位アドレスは ALE の立下りで取り込むよう，システムクロックに同期を取らずに動く非同期回路としました．本当に同期回路で組むならば，システムクロックの数倍の CPLD 用クロックを用意してやる必要がありますが，同期回路の設計に関する詳細はここでは割愛させていただきます.

2.5 DDS モジュール制御

任意周波数の正弦波を出力する秋月電子のキットを使用します．付属の説明書にマイコンからの制御用タイムチャートも記載されています．Data，Clk，Strobe の三本を使い，シリアルでコマンドとパラメータを転送します．原理は簡単で，DDS 用 LSI 内部のシリパラ用Ｆ.Ｆ.のデータ・クロック信号に対してマイコンが出力し，所望のデータを内部レジスタに入力します. その後, ストローブ信号をアサートすると LSI が働いてくれます.

キットのままですと 1Hz 単位で周波数設定が可能で，送信する周波数データは希望する周波数の 26bit 即値そのままで OK です. 送信すべきデータは ID 3bit, コマンド 4bit, パラメタ 26bit の順にそれぞれ LSB first です.

2.6 ADC

とりあえず, 現地調達可能(デジット in 大阪日本橋)なもので安くてパラレルで少しでも早いものを選んでみました．カウンタでカタログを見せてもらいながら 500 円以下のものを物色したところ，マイコンバス直結できそうだったので購入．バスタイミングを見て検討したところ, Hi-Z からデータ出力までの遅延が長くて直結できませんでした. 意味がねぇ…. てことで 541 を介してバスへ接続しております.

また，解像度を上げたい場合はシリアル出力タイプのものを I/O ポートにぶら下げて使うといいでしょう．ピンが足りないようであれば CPLD から出してやれば OK です.

2.7 Log-AMP(対数アンプ)

AnalogDevices の対数アンプ AD8307 を入力段に使

用し，幅広いダイナミックレンジを得ています．データシートにある参考回路を流用してしまいます．動作理解のため，データシートを見ていたのですが，引っかかった点があったのでここに記しておきます．

データシート中に出てくる入力レベルの単位が[dBm]である点です．これは基準電力 1mW と対象となる電力との比をとり，その対数をとったものです．ということは入力レベルは電力ということになりますが，今回検討しているのは電圧比です．このままでは良くわからないので，アナデバの WEB サイトを検索してみました．同様の質問が寄せられており，ここでは入力インピーダンス 50Ω として考えられているとのことでした．したがって，電力レベル

$$Power[dBm] = 10 \times \log_{10}((V_{rms}^2[V] / R[\Omega]) / 1[mW])$$

は，基準電圧を 1 V とした入力電圧レベルの式

$$Voltage[dBV] = 20 \times \log_{10}(V_{rms} / 1[V])$$

と比較して，

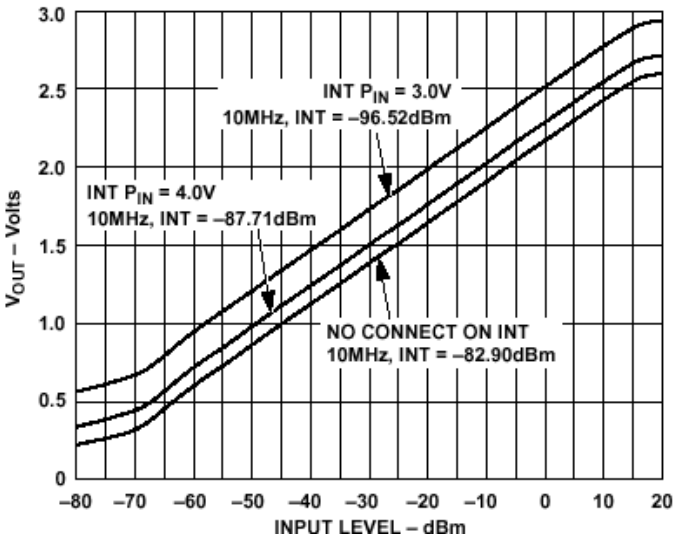
$$Voltage[dBV] = Power[dBm] - 13[dB]$$

を得られます．実際にV_{rms} に値を入れて計算すると，13[dB]の差となることがわかります．

さて，ここで**図 7** の入力レベル対出力電圧特性を見てみましょう．このデバイスの特徴として，出力電圧に任意のオフセットをかけることができます．図中の$INTP_{IN}$ は，オフセット値を決定するための入力電圧です．グラフから見てわかるように，-65～+15dBm くらいまではリニアに変動しているので，この範囲で動作するように設計します．これを入力電圧に換算する（$10^{Voltage[dBV]/20}$）と，0.13[mVrms]～1.26[Vrms]，0.35[$mV_{p.p.}$]～3.56[$V_{p.p.}$]となります．

ということで，電圧入力ピンにかけて測定できるのは 1 V くらいということで，これでは S/N 比がよろしくないので，入力時に 20dB ダウンさせる回路を採用します(データシート参照)．リニアで考えると 1/10 する事となり，入力インピーダンスが 1.15kΩ，外付けで 10kΩついているので確かに分圧されて 1/10 されているようです．これにより，最大 35.6V_{pp} 程度まで耐えられます．分圧されることで，電源電圧以上の計測も可能となるでしょう．

図 7　入力レベル対出力電圧特性



2.8 ボタン・ロータリエンコーダ

汎用ポートに入力します．プルアップしておいてボタン押下で GND に落とすようにします．それぞれ入力にチャタリングが発生しうるので，1mSec くらいに二度サンプルして連続して'L'であれば有効としました．

ロータリエンコーダは共立電子 in 大阪日本橋で買った 20pulse/回転のものです．**図 8** にエンコーダのスイッチングタイミングを示します．A,B の状態をポートから読み出した値が 'HH' から 'HL' もしくは 'LH' に遷移した時に 1 クリックされたとみなします．

当初はA相の立下りでB相のデータを見ていたのですが，CCW だと'0'を検出しなければならず，チャタリング防止のサンプルが遅いために高速な回転では正確に検出できませんでした．

2．A 全体回路図

図 9・図１０に回路図を示します．DDS モジュールはそのまま使用しているため，図中には掲載されておりません．

図 8　ロータリエンコーダ出力波形

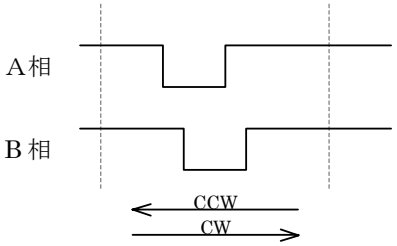
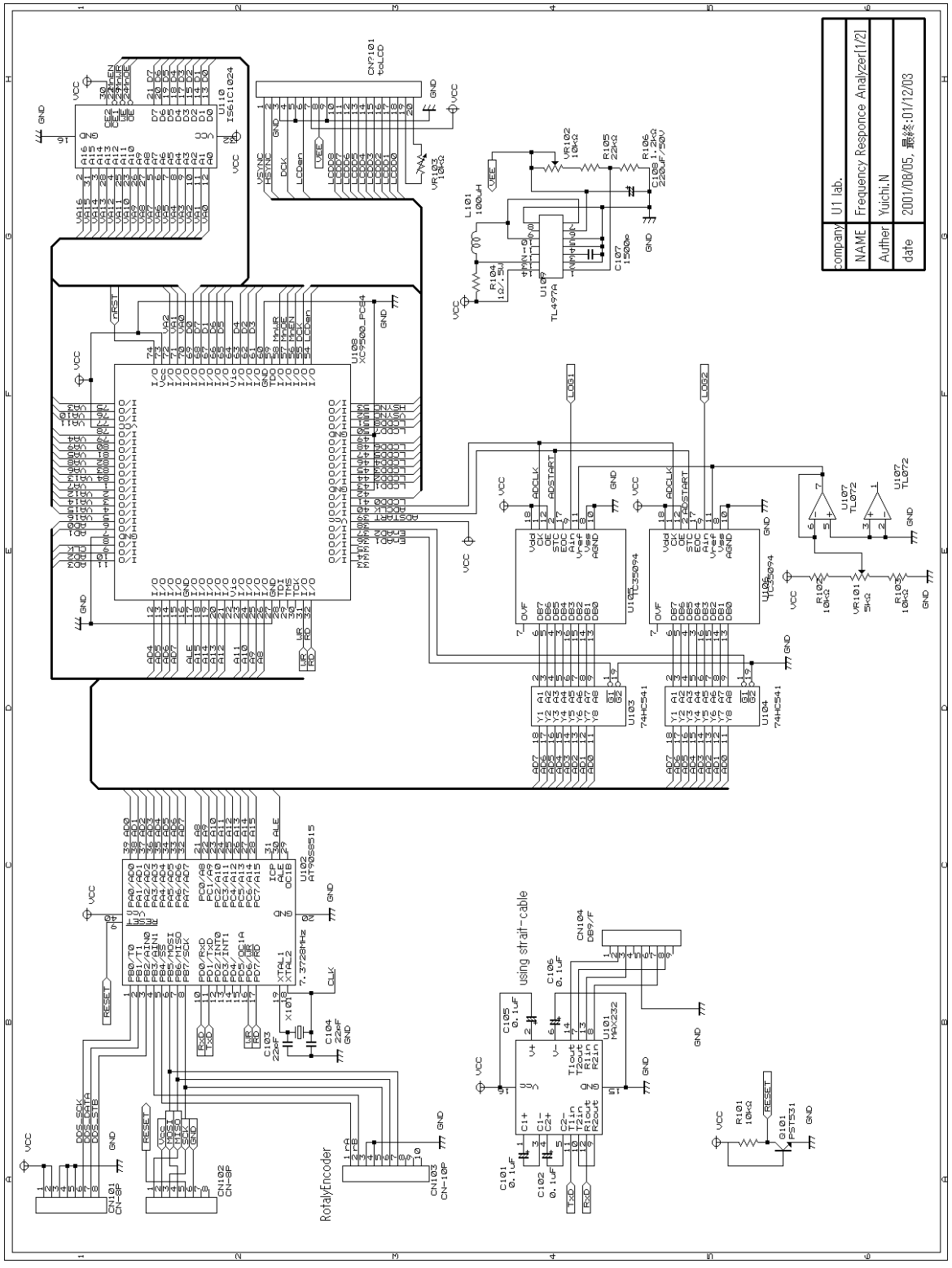


図９．制御 CPU/CPLD,液晶駆動回路之図



3. 制御ソフト

AVR-gcc[5]にて開発を行いました．アセンブラより code 開発効率が良く、複雑な条件式も考えずに済んで楽ができます．多少，アセンブラコードを思い描いて記述していますが…(^^;.

3.1 全体の流れ

図１１にプログラム全体のフローを示します．えらく簡素に書いていますが，要約するとこれだけです．ミソは Timer を用いて定期的にスイッチ状態をサンプリングしてチャタリングを回避している点と，**擬似的**にマルチタスク動作をしている自動計測制御部です．後者により，自動計測部と UI 制御部はそれぞれ別タスクとして動作しているとみなせます．

制御部は壱関数として一定時間ごとに実行されるので，その状態や必要なデータは保持しておく必要があります．詳細は以下の節にて記述します．

3.2 UI 制御部

UserInterface 制御，すなわちユーザーからの入力やユーザーへの情報提供を行う部分です．入力を要求したり結果を表示したり，入力に従って適切な動作を行います．現段階ではロータリエンコーダと PushSW を使って周波数スイープ範囲や変動方法などをメニュー形式で選択，値の変更を行います．

液晶画面上にメニューを表示し，選択されている項目はそれが判るように色を変えます．エンコーダの回転に従ってメニュー項目を移動し，選択ボタンが押されれば値変更を行います．ユーザ入力が生じるまで待ち状態を続けるので，割込タスクとしても良かったのですが，画面描画で割込み処理を長く引っ張るとよろしくないので最低レベルのタスクとしてメインルーチンで走っています．ユーザ入力データは Timer 割込部で取り込むため，人手による入力では取りこぼすことは少ないでしょう．

図１２に簡単な流れを示します．実際には遷移に伴って，選択項目番号や値などの情報の変化，画面描画が実行されます．

図１２ UI タスク流れ

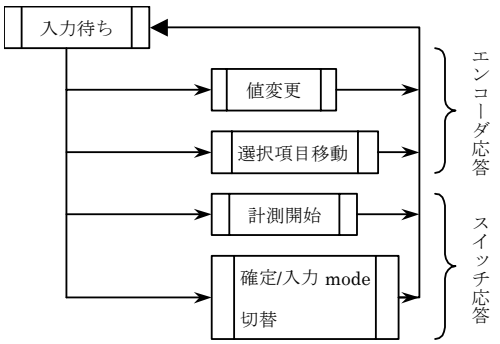


図１０ Log-AMP,出力 AMP 之回路図

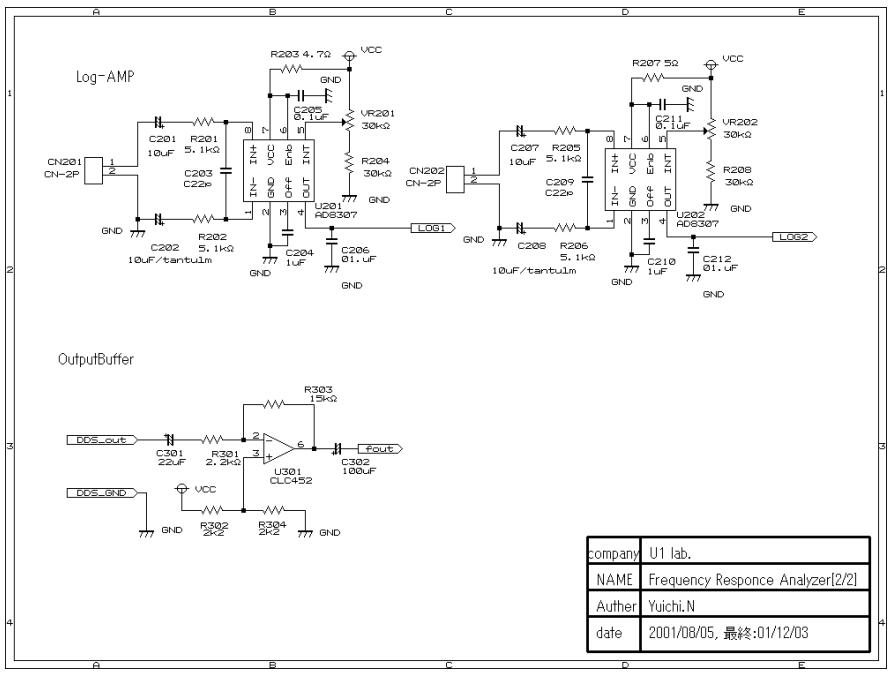
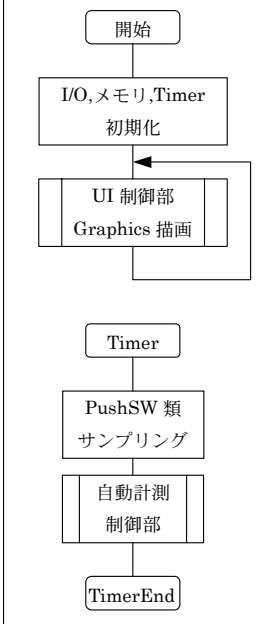


図１１ 全体フロー



[5] http://www.avrfreaks.com/ にて入手可能.

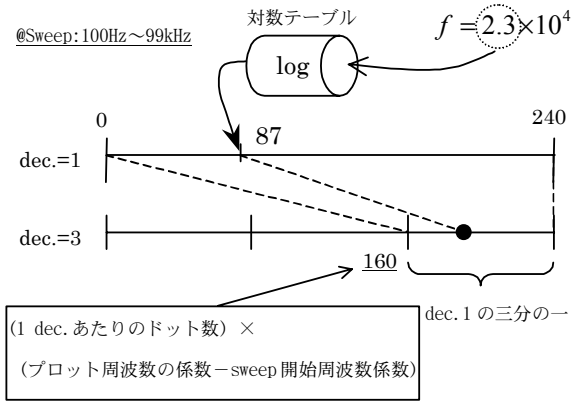
⇨ **波形描画**

一般的に周波数対入出力レベルの特性図は，ダイナミックレンジが広いため両対数グラフを用います．本機ではレベル計測時に log-AMP を用いているため，AD 変換後のデータは既に対数を取った後の値を得られています．しかし，周波数に関してはユーザーの指定した値を用いるため，対数をとる関数が必要となります．AVR-gcc にも math.h が存在し，log 関数が使えると思っていたのに AT90S8515 ではリンカではねられてしまい，使用できないようでした..

そんなわけで周波数－対数テーブルを用意する必要があります．画面描画にしか使わないため，周波数からプロットすべき座標へと変換することにして，あらかじめ必要な計算を全て行っておきます．周波数スイープ範囲から，１～５dec.までの表示が必要となります．そこで，計算しやすいように１～５の最小公倍数で 320 未満の最大値をグラフの横幅とします．ただし，軸メモリにラベルを配置する必要があるため，その分の空間は空けておく必要があります．結果として 240 ドットを選定し，表示する dec. の幅に応じて 1dec.あたりのドット数が決まります．

あとは対数の上位二桁を見て 1.0～10.0 に対応するドット数を配列から引き出す事によって，対応する dec.の始めの座標からの相対値を得られます．ここで，横軸の幅を表示すべき dec.数の最小公倍数と取ったことにより，dec.始めの座標は割り切れることがわかります．このため，除算による誤差で見栄えが悪くなることはありません．（**図１３**）

図１３ 周波数軸の座標導出例



ところで，計測データとして保持すべき値は，計測した周波数・入力レベル・出力レベルとなります．プロット時に周波数の表示精度が二桁しかないため，有効数字二桁＋指数という値でデータを保持することにしました．こうした理由にはメモリ量節約も絡んでおり，周波数をそのまま保持しようとすると 20bit 必要となり，4Byte 消費することになります．ところが，有効数字二桁と指数であれば２Byte で済み，半分になります．

実際にこの手段を講じた理由にはもうひとつありまして，周波数を入力するときにはレンジが広いためロータリエンコーダをぐりぐり回すのが使いにくいという点です．PC でのリモート制御時には，データ蓄積を行わないつもりなので，1Hz 単位での周波数設定が可能となるでしょう．

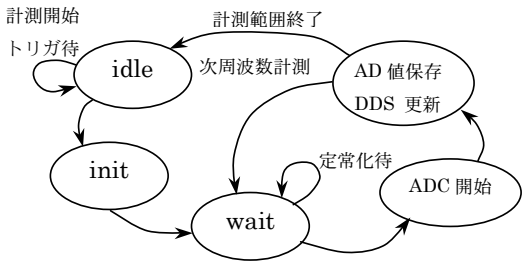
⇨ **データ入力**

HumanInterface の要，操作性に関してはここで全てが決定されます．今回の入力装置はロータリエンコーダということで，これでメニュー選択，数値決定を行います．メーカ製のもののように，回転速度に応じて数値の増減が加速するような実装をすれば快適ですが，簡単のためパルス入力をそのままカウントアップ／ダウンすることにしました．その代わりにあるボタンを押しながら回すと一桁上の値が変動するようにしてあります．特に凝ったことをしているわけではありませんので，プログラムソースを覗いていただければフローチャートどおりである事がわかるかと思います．

３.３ 自動計測制御部

自動計測制御部では，UI 制御部の指示に従って DDS の出力周波数を更新／ADC サンプリング制御／ADC データ保存等を Timer 割り込みごとに状態遷移します（**図１４**）．各状態について紹介しましょう．

図１４ 自動計測部状態遷移図

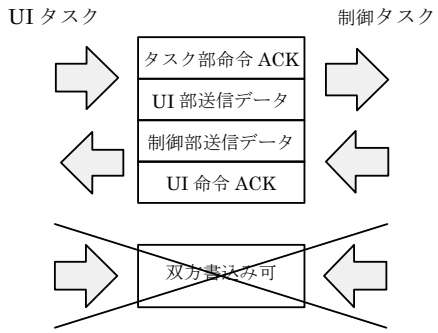


⇨ IdleState

UI 制御部からの計測開始命令を待ちます．RT-OS 上ならば MessageBox なんかが実装されていますが，ここではタスク間通信用に共有メモリを用います．（とはいえ，管理している OS は無いので全て自分で面度を見ることになりますが.）簡単な図解を**図１５**に示します．メモリ上にフラグを置いて，その書込みは UI 部からのみとします. それに対する応答として自動制御部からのみ書込みを行うエリアも用意しておきます．割込処理とメイン部とで排他アクセス制御をするのが面倒なので，それぞれにのみ書き込むことを許可するエリアとそれを読み込んだ事を

送信者側に伝えるフラグで実現しています.

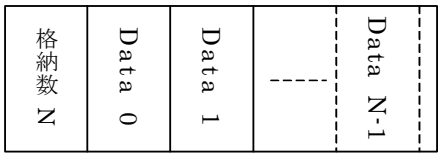
図１５ タスク間メッセージ実装概要



⇨ InitState

計測開始・データ蓄積のための準備を行います. 使用する変数の初期化，特にデータ保持用のカウンタをクリアします．Pascal 配列のように，データ蓄積用バンクの先頭 2 バイトに保持しているデータ数を格納します．（**図１６**）

図１６ データ保存方法



⇨ WaitState

正弦波出力後に Log-AMP 出力電圧が安定するまでの時間稼ぎを行います. Log-AMP 出力部に付ける積分用コンデンサの値によって待ち時間もかわります．大容量のほうが低い周波数まで安定した電圧を得られますが，そのぶん電荷の蓄積に時間がかかります．Log-AMP 出力は 2uA の電流源とされていますから，おおよその待ち時間は検討できるかと思います．周波数帯ごとにリレーで容量を変えれば良かったのでしょうが手を抜きました.

⇨ ADC State

ADC 開始トリガを出力し，変換時間経ぶんだけ待ちます．実際にループで待っているわけではなく，Timer 割り込み周期がわかっているので必要な回数だけ待ち状態を取ります.

⇨ UpdateState

変換が終わったデータを取り込み，データ蓄積バンクに計測データを追加します．ここでスイープ範囲の最後に達したときは IdleState に, そうでなければ DDS の出力周波数を変更して WaitState に遷移します.

３.４ 下位モジュール

ここでは，主に *HumanInterface* で必要とされる機能を実装します．とはいえ，一般的に使用可能なグラフィック関数を用意しましたので，点描画部分さえ各自のハードウェアに合わせて書き換えていただければ別のものはそのまま使えます．静的データ（文字データ）格納場所はアーキテクチャ依存ですが.

⇨ **点描画**

関数：pset(Word *x*, Byte *y*, Byte *col*)

グラフィック描画の基本といえばドットを１つ打つことでしょう．１ドットあたり１バイト 256 色を割り当てているため，座標（*x*，*y*）へ任意の色のドットを置くためには VRAM アドレスの計算が必要となります．本機の場合は以下の式により求まります.

$$offset = y \times 320 + x$$

但し，$\{x, y \mid 0 \le x \le 319, 0 \le y \le 239\}$，*offset* は VRAM 先頭からのオフセット値です．このままでは演算性能に問題がありますので，自前で展開しておきましょう. 320 倍は 256 倍と 64 倍の和と同値ですから，Byte で受け取った y の値を Word の上位へ転送し，左に 6 ビットシフトした値と加算すれば OK です．２の累乗倍であれば gcc でもシフト演算にしてくれたのですが，320 倍は展開してくれませんでした．型変換によるものと思いますが，このあたりは今のところ人手のほうが最適化できますね.

⇨ **線分描画**

関数：g_line(Word *sx*, Byte *sy*, Word *ex*, Byte *ey*, Byte *col*)

線分は点の集合です．始点と終点を与えれば線分を描画する関数を用意しました．基本的に描画枠内の座標から離れることを想定して作っておりませんので，使用の際にはご注意ください．アルゴリズムとしては，縦・横方向のドット数の変化の大きいほうを１ずつ変化させ，それに対応する他方の座標を求めることでドットの隙間ができないように作られています．このあたりも基本的なアルゴリズムですから説明は割愛させていただきます.

⇨ **四角描画**

関数：g_box(Word *sx*, Byte *sy*, Word *ex*, Byte *ey*, Byte *col*)

関数：g_boxfill(Word *sx*, Byte *sy*, Word *ex*, Byte *ey*, Byte *col*)

線分４本で長方形が描画できますので，線分描画関数を４回呼び出せば実現できます．関数化することにより code 使用量を減らしました．また，長方形の内側を塗りつぶすものも作り，画面クリアや文字の背景塗りつぶしなどに使用します.

⇨ **文字描画**

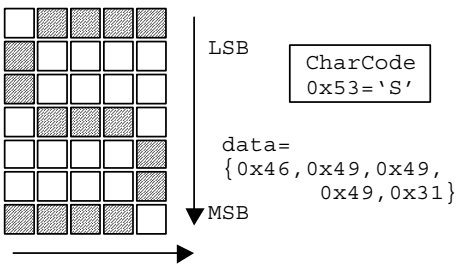
関数:puttext(Word x, Byte y, Byte col, Byte c)

基本的にドットを打つだけですが，文字の形という情報が必要になります．今回は外付け ROM を使わないという考えから，できるだけフォントサイズを小さく抑えるように考えます．

回りを捜索してみるとキャラクタ液晶モジュールのデータシートが見つかりました．ここには文字コード表としてビットマップデータが記載されています．無論，半角文字だけで５ｘ７ドットのものです．縦方向を１バイトとすれば，**図１７**に示すように，一文字を５バイトで数値化することができます．制御コードや表示する文字のないあたりを差っぴけばそれほど容量は必要ありません．フォントデータ採取決定です．

使用されるであろう文字を，文字コードで 0x20～0x7F,0xA1～0xDF とすると，１５９文字／７９５バイトですみました．実際にプロットされるのは指定された座標を文字の左上始点とした５ｘ７ドットの領域です．バックグランドカラーの描画は行われませんので，あらかじめ塗りつぶしておく必要があります．背景色の描画も必要であれば，少しだけ書き加えるだけで済みますので適当にやってください．

図１７ フォントデータの作成



4. 使用例

ここでは実際に製作した機器でウィーンブリッヂ回路の周波数特性を計測します．

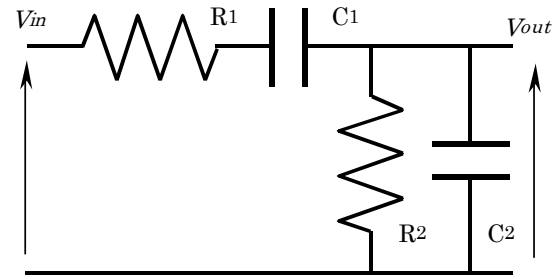
図１８に示すウィーンブリッヂフィルタ回路を用いますが，おおよその特性を考えてみましょう．

R1，C1 がハイパス，R2，C2 がローパスフィルタであると考えられます．したがって，合成されたフィルタはバンドパスフィルタであることがわかります．ただし，フィルタ間の結合では何もしていないので，単純に二段のフィルタとはなりません．おおよその動きということで[6]ご容赦願います．

R1，R2，C1，C2 はそれぞれ，5.5kΩ，5.1kΩ，1nF，1.5nF としました．その結果，LPF 側はおよ

[6] 原稿納期がまずいので，詳細は別途フォローしたいと思います(^^; WEB のほうで要求してください….

図１８

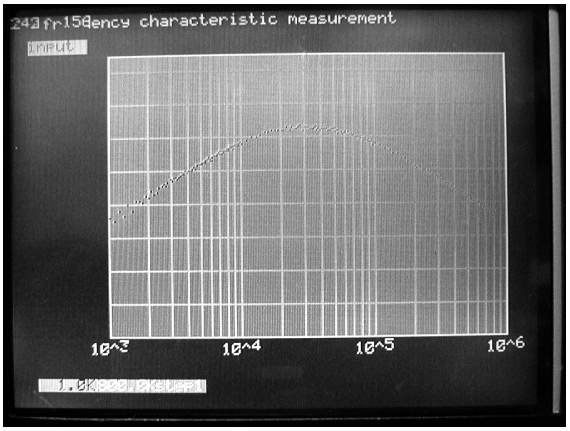


そ 28.9kHz，HPF 側は 20.8kHz となりました．その間はおおよそフラットな出力を得ることになりそうです．（もちろん，減衰はします．）

この条件で計測をしたときの画面写真を**図１９**に示します．かなり見にくいですが，20kHz から 40～50kHz 程度まではフラットっぽく見えます．計測した点だけをプロットしたので対数軸の広いところではまばらに点在しています．誤差も結構あるらしく，きれいな直線とはなっていません．

画面も殺風景なのでいろいろと装飾していきたいですね．画面左上の数値はデバッグ用の入出力レベル値です．

図１９ 計測画面例



5. 検討・拡張

現段階では電圧レベルしか測っていないため，フィルタ回路の計測で重要な位相差はぜひともほしいところです．入力している周波数が既知なので，入出力のゼロクロス点でトリガをかけて時間差を出せば位相差が求まります．分解能に問題がありそうですが，この方法で位相差検出も行ってみましょう．

また，より細かな動きを見たい場合，ADC が 8bit では分解能が不足だと感じられます．画面解像度を考えると 8bit で十分ですが，PC に取り込んでグラフを書こうとするならば 10bit 以上欲しいところです．これは適当な石を探して差し替えればいいでし

ょう.

ここで PC による制御を検討しているとありますが，回路は用意されてケースにも穴はあいてたりします．しかし，マイコンのメモリ問題がくるかもしれないことと，LCD まで付けておいて PC で制御というのはどうかという点が引っかかって手付かずです[7]．液晶駆動を除けば USB デバイスとしての作成も何とかなりそうな感がありますが, OP アンプの入手性と出力アンプの電圧が弱くなる点を克服する必要がありそうです.

最後に，CPLD の動作シミュレーション波形を**図２０・図２１**に示します．横軸の時間値は，シミュレーション上のものなので相対時間を得るために使ってください．ハードウェア編で紹介したタイミングどおり，正常に動いているように見られます．シミュレーションを行うことによって，実機で動作確認をする前に問題が見つかるので大変便利です.

s 液晶駆動部分の V-Sync と H-Sync の立ち上がりが同時ではいけないという仕様があったらしく，動かし始めてから気づきました．CPLD だったので基盤をいじることなく修正ができて楽でした.

6. まとめ

秋月電子のデジタル液晶を使っている記事を見かけたことが無いので液晶表示を…と作っているうちに，DDS と LogAMP で f 特もはかれるなぁと考えていたモノを付け足してできたものです．同じ趣向のものもありますので，本記事がそれらの理解につながることを期待しております．これ，そのまま作ると結構な金額になります…….

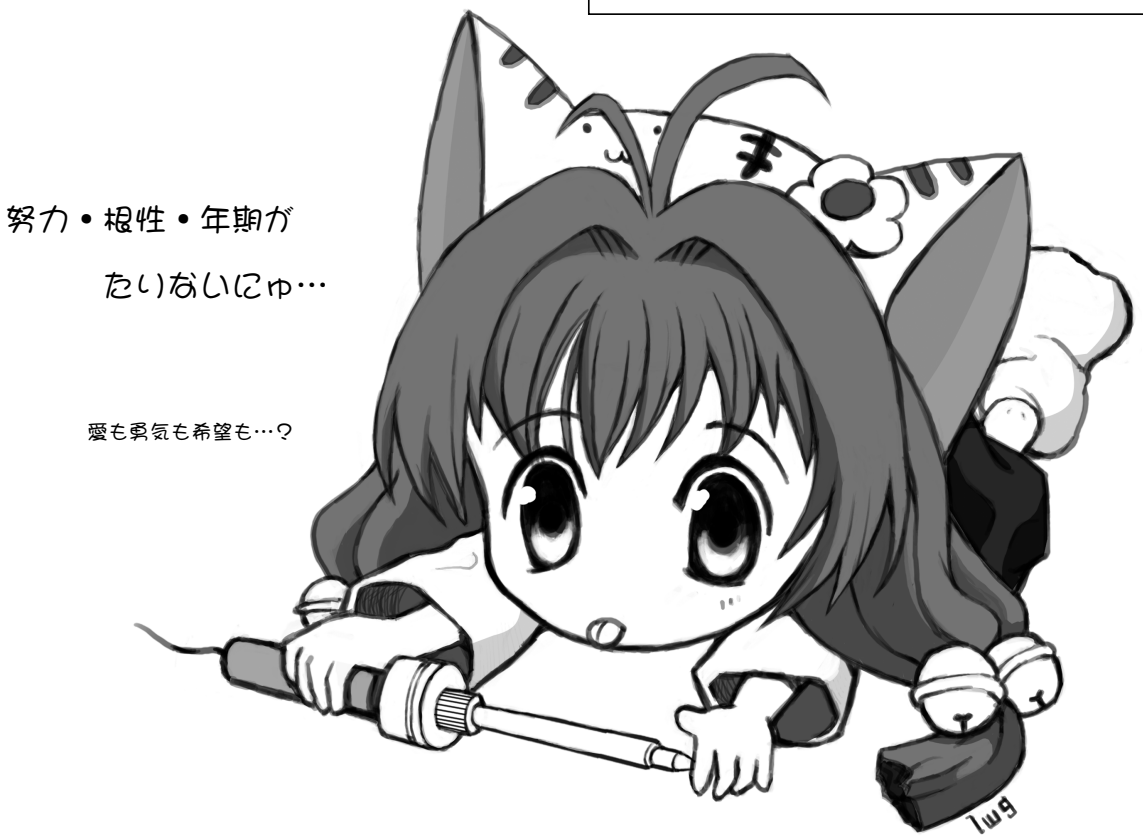
行く末は以下で紹介するフォロー頁上でご紹介いたします(^^;.

最後に，この場を借りて，**快く**ぷちこたれイラストを描いていただいた lwg 様に感謝をいたします.

参考文献

- **CQ 出版**,トランジスタ技術 special
  『基礎からの電子回路設計ノート』No.36
- **Analog Devices, Inc.**,AD8307 DataSheet
- **EPSON?**,LM32C041 DataSheet
- **ATMEL,** AT90S8515 DataSheet

※回路図，ソフト，アフターフォローは以下の URL にて…
http://homepage2.nifty.com/u1lab/C61/danyo.html



[7]原稿に追われていたという話もありますが…(w

図２０ 外部メモリアクセスタイミング

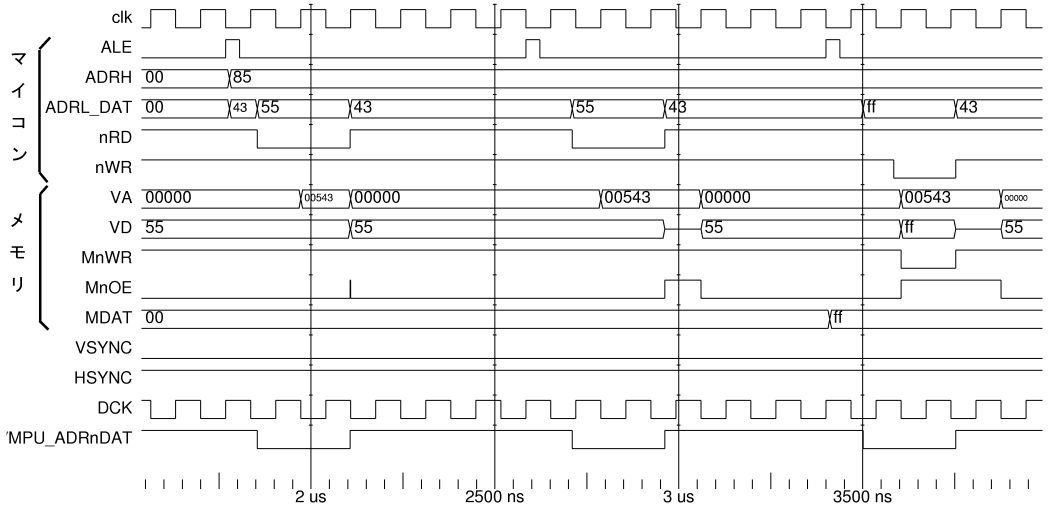
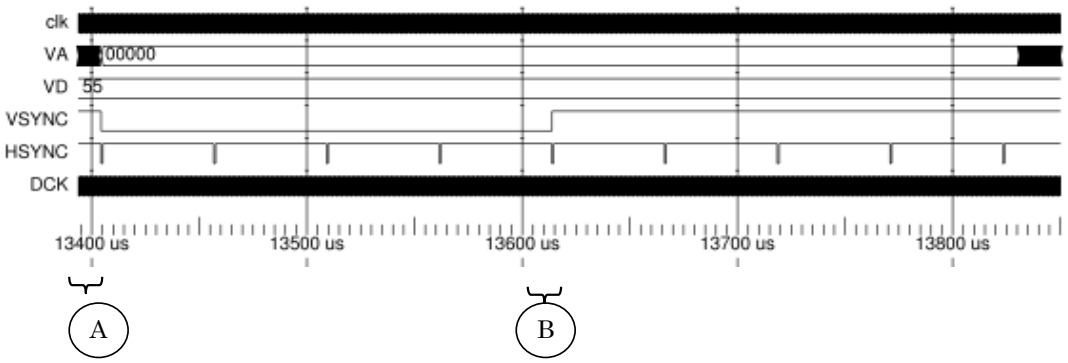
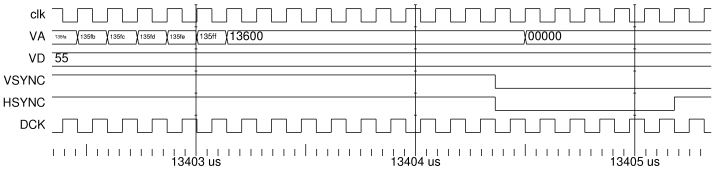


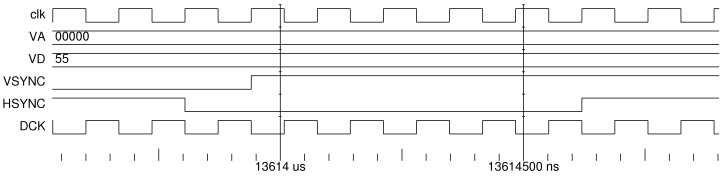
図２１ LCD 駆動タイミング



A 部分拡大



B 部分拡大



スパコンを作ろう

ＲＯＭ男
cma59466@biglobe.ne.jp

クラスタリングって？

クラスタリングの手法を用いたスーパーコンピューターが注目を集めています。クラスタリングとは一つの仕事を分割して複数の PC にネットワークを通じて送り、負荷を分散させて高速化を図る手法です。仕事の種類によって向き不向きがあり、相互干渉が少ない、沢山の要素に分割できる仕事ほどクラスタリングで処理するのに向いています。

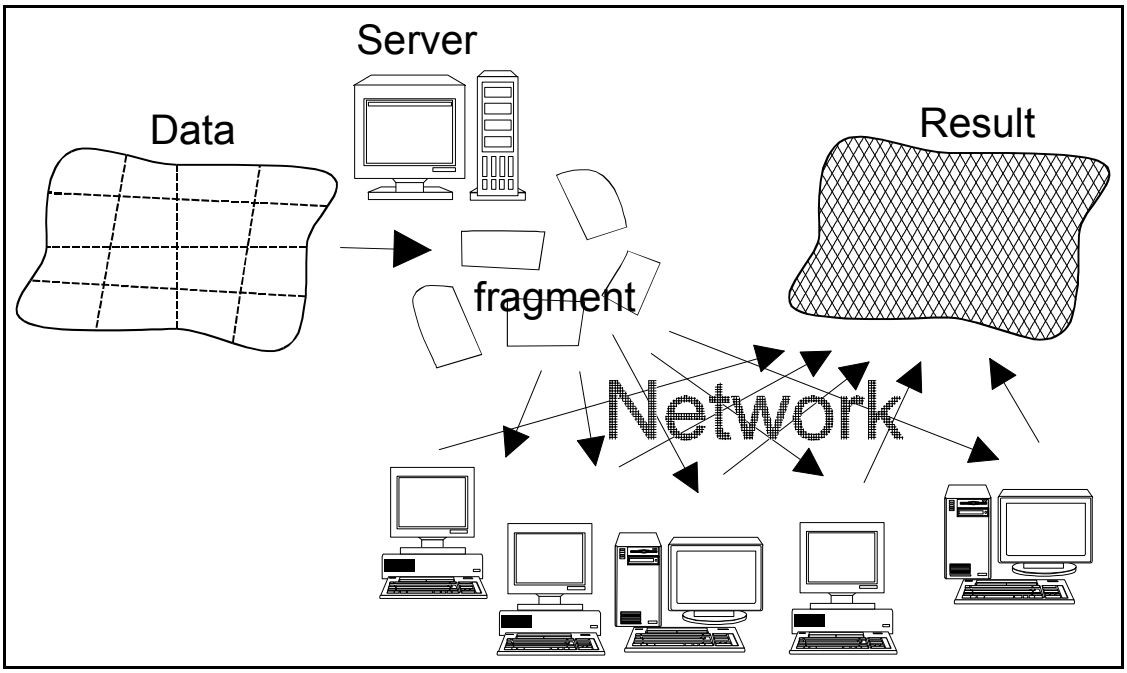


図 1 クラスタリングの概念例
巨大だが分割可能なデータを多数の PC で分担して処理する

市販の PC を多数接続してスーパーコンピューターを作るプロジェクトとしては Beowolf Project が有名です。また、Beowolf Project では PC 間の通信がボトルネックになるということでデータ交換領域への DMA 利用等ネットワーク周りに徹底して手を入れた Score がります。

このクラスタリング技術を用い、インターネットに接続している PC を使って巨大なデータを解析しようという試みがあります。SETI@home では有志を募り、宇宙から届く電波を

解析して宇宙人の存在を探っています。RC5 cracking では通信情報の暗号化の重要性を訴えるために暗号化したサンプルを総当たりで解読しています。最近になって白血病の治療薬を探すために特定の蛋白質に対して活性を示す薬品を探すプロジェクトや AIDS の治療薬を探すプロジェクトも始まっています。これらのプロジェクトでは要素 (電波データや薬の候補) の間に相互作用がないので、通信速度は比較的問題になりません。

私も白血病治療薬解析プロジェクトにTeam 2ch の一員として参加しています。プロジェクトの主催者側はプロジェクトを進めたいので、参加者に順位をつけるなどして競争を煽るので私も自分の PC を増やして上位に食い込んでみたくなりました。最初はお役御免にしたマザーボードを復活させました。次に PCI スロットに挿して使う、カード PC を使いはじめました。しかしお役御免だったマザーボードの PCI スロットは埋まっていました。これ以上 PC を増やすためにどうすればいいか考えた末に多数のカード PC をバックプレーンボードに挿すようなシステムを作ることにしました。本当のクラスタリングでは多数の PC を協調して動かすための OS あるいはソフトが必要になりますが、今回その部分はプロジェクトを始めた団体任せということにしました。

製作

PC ユニット

コンピューターユニットにはアシストのカード PC を使うことにしました。 AT/ATX タイプのマザーボードの PCI スロットに差し込んで使用するものです。カード PC にはタイプ A (マザーボード側にネットワークチップを一つ割り振る) とタイプ B (マザーボードとカード PC が完全に独立している) があり、個人的な都合上タイプ A を購入した分もありますが、ここではタイプ B を利用することを前提に話を進めます。

表 1 と表 2 にカード PC のカードエッジにランドが乗っている位置とその機能を示します。ほとんどが電源でした。3.3V は使っていないので必要な電源は +5V と +12V, -12V ということが分かりました。

PnP 用の信号線が出ています。これは原則としてプルアップすることにしました。割込み信号の位置にもランドが乗っていますがどのように利用しているのかよくわかりません。PRSNT1#, PRSNT2# はマザーボードに該当スロットがどれだけ電力を消費するかの目安を知らせる信号です。PCI の規格では最大 25W ということになっているのでコンピューターユニットの消費電力もそのくらいに収まっていると考えられます。

表 1 アシスト Card PC (Type B) のランドとピンアサインの関係

	A			B		
	Assign	PC	HDD	Assign	PC	HDD
1	TRST#	○		-12V	○	
2	+12V	○	○	TCK	○	
3	TMS			Ground	○	○
4	TDI			TDO	○	
5	+5V	○	○	+5V	○	○
6	INTA#			+5V	○	○
7	INTC#	○		INTB#	○	
8	+5V	○	○	INTD#	○	
9	Reserved	○		PRSNT1#	○	
10	+5V	○	○	Reserved	○	
11	Reserved			PRSNT2#		
12	Ground	○	○	Ground	○	○
13	Ground	○	○	Ground	○	○
14	Reserved			Reserved	○	
15	RST#			Ground	○	○
16	+5V	○	○	CLK		
17	GNT#			Ground	○	○
18	Ground	○	○	REQ#		
19	Reserved			+5V	○	○
20	AD[30]			AD[31]		
21	+3.3V			AD[29]		
22	AD[28]			Ground	○	○
23	AD[26]			AD[27]		
24	Ground	○	○	AD[25]		
25	AD[24]			+3.3V		
26	IDSEL			C/BE[3]#		
27	+3.3V			AD[23]		
28	AD[22]			Ground	○	○
29	AD[20]			AD[21]		
30	Ground	○	○	AD[19]		
31	AD[18]			+3.3V		

表2 アシスト Card PC (Type B) のランドとピンアサインの関係(続き)

	A			B		
	Assign	PC	HDD	Assign	PC	HDD
32	AD[16]			AD[17]		
33	+3.3V			C/BE[2]#		
34	FRAME#			Ground	○	○
35	Ground	○	○	IRDY#		
36	TRDY#			+3.3V		
37	Ground	○	○	DEVSEL#		
38	STOP#			Ground	○	○
39	+3.3V			LOCK#		
40	SDONE			PERR#		
41	SBO#			3.3V		
42	Ground	○	○	SERR#		
43	PAR			+3.3V		
44	AD[15]			C/BE[1]		
45	+3.3V			AD[14]		
46	AD[13]			Ground	○	○
47	AD[11]			AD[12]		
48	Ground	○	○	AD[10]		
49	AD[09]			Ground	○	○
50	Key Way			Key Way		
51	Key Way			Key Way		
52	C/BE[0]#			AD[08]		
53	+3.3V			AD[07]		
54	AD[06]			+3.3V		
55	AD[04]			AD[05]		
56	Ground	○	○	AD[03]		
57	AD[02]			Ground	○	○
58	AD[00]			AD[01]		
59	+5V	○	○	+5V	○	○
60	REQ64#			ACK64#		
61	+5V	○	○	+5V	○	○
62	+5V	○	○	+5V	○	○

バックプレーンボード

PCI ソケットのピンは 1.27mm (1/20inch) の整数倍で出ているので、一般的な 2.54mm (1/10 inch) ピッチのユニバーサル基板は使えません。また、大きなサイズの 1/20inch ピッチのユニバーサルボードは手に入りません。PICMAG 仕様の PCI バックプレーンボードを流用しようかとも思いましたが

- 今回必要なバックプレーンボードは PCI 側からは電源のみつながっていれば良い
- しかし PICMAG なバックプレーンボードはその他の信号線も接続されていてその分高価
- ほしいのは PCI コネクタが多数並んでいるものだが、そういうタイプはほとんどない

ということで電源供給用バックプレーンボードは自作することにしました。また製作にあたっては

- まず、1/20inch ピッチの蛇の目基板を探し、それに PCI ソケットを二つと電源用コネクタを搭載する
- 蛇の目基板には電源のみ配線する。
- 複数の蛇の目基板をベースボードに取り付ける

という構成を採用しました 。ただし、ベースボードの作成は間に合いませんでした。
蛇の目基板については秋月電子通商が販売している AE-2M が カード PC 1 ユニットを乗せるのにちょうど良いサイズ (72mm×95mm) であり、一枚￥270 と非常に安価だったので採用しました。次の図 に蛇の目基板の加工工程を示します。順に

1. 未加工 (左上)
2. PCI コネクタ用 (中上)
3. 電源コネクタとリセット用マイクロスイッチ取り付け (右上)
4. PCI スロット取り付けと電源の配線 (左下)
5. リセット用コネクタの取り付け (中下)
6. 裏面の様子 (右下)

となります。

PCI コネクタ取り付け用の孔あけはミニドリルを使いました。SECC Celeron の Dual 化改造のために購入したものです。写真では 3mmϕ の刃をつけています。孔の位置は現物あわせで決定しました。蛇の目の孔の位置と PCI コネクタの突起の間隔は合わないので孔径を大きくして対応しました。

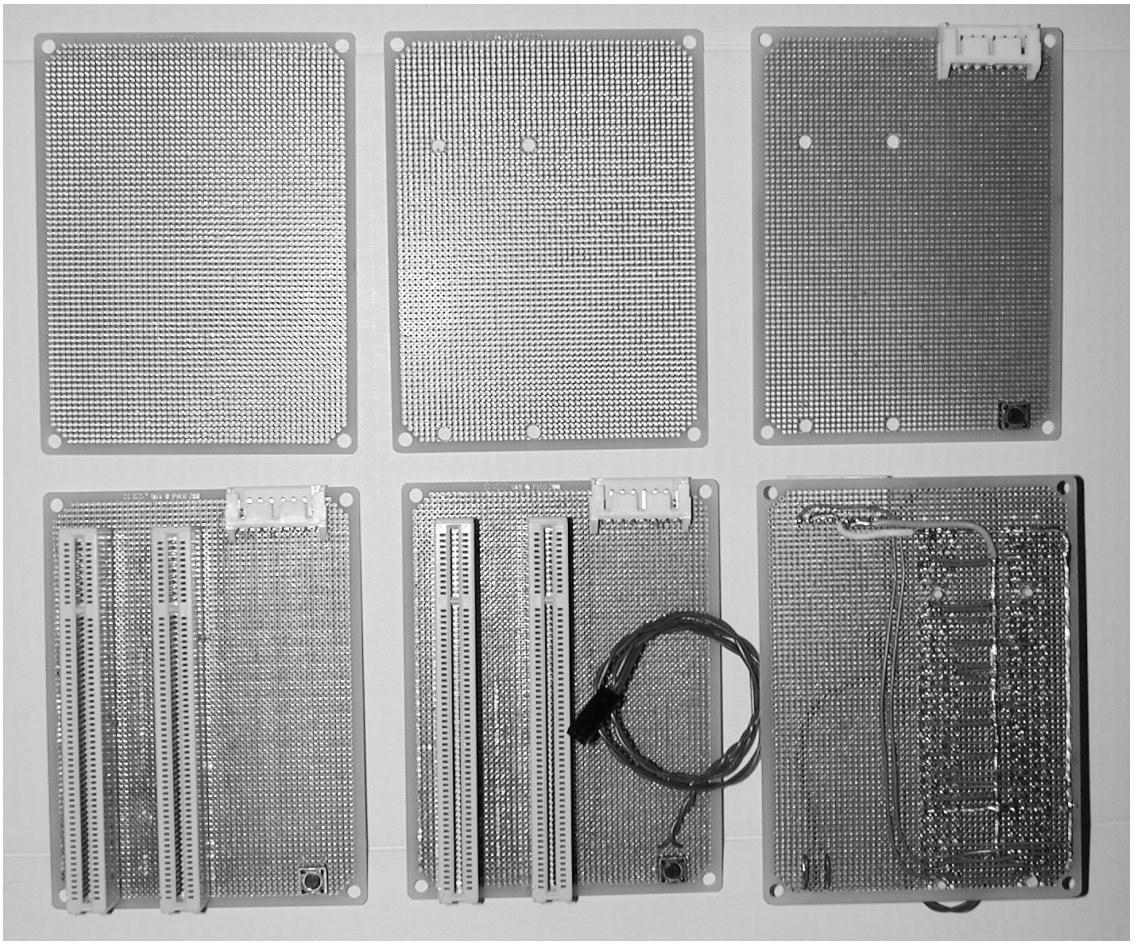


図 2 バックプレーンボードの製作過程

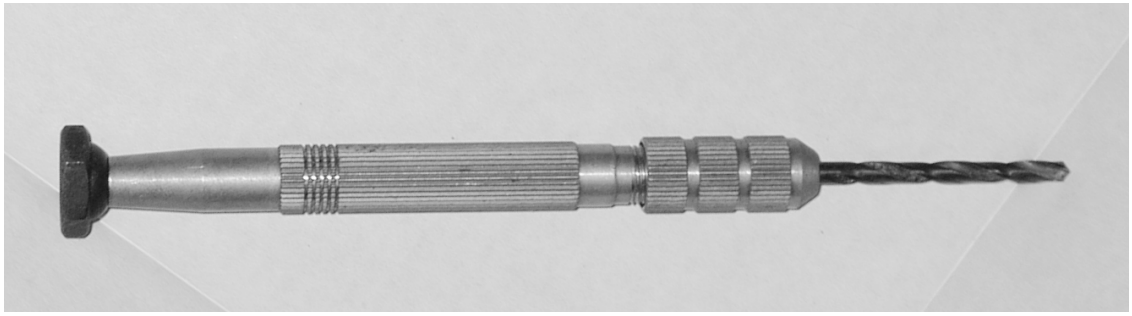
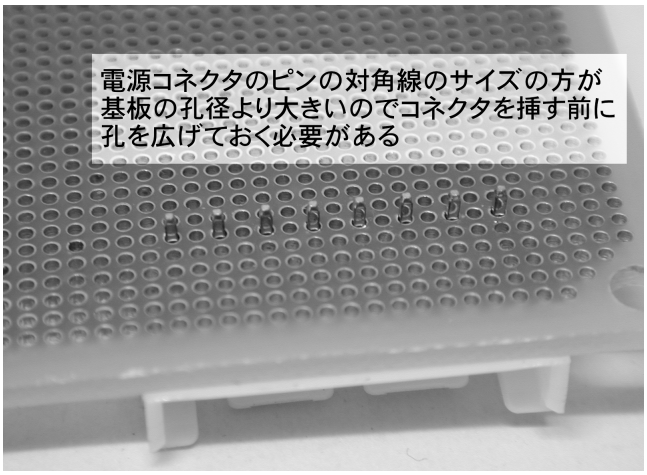


図 3 ミニドリル

コネクタ・スイッチ取り付け上の注意点

今回採用した電源コネクタのピンは蛇の目基盤の孔径より大きいので、単純に取り付けようとしても入りません。そこでコネクタのピンが入る孔はあらかじめ同形のピンをペンチで押し込んで孔を角型に広げておきます。その後コネクタを取り付けると比較的うまく行きま



す。

PCI コネクタのピン間隔と蛇の目基板の孔間隔はどちらも 1.27mm ピッチだとはいえ、微妙な誤差はあるようですしコネクタを挿し終えると基板が微妙に反っていることに気が付くでしょう。PCI コネクタは気をつけて挿し込まないと 足を折ってしまいます。

図 4 電源コネクタのピンを基板に挿した様子

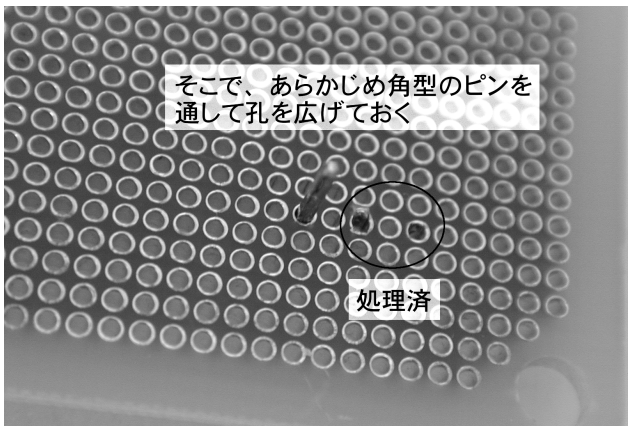


図 5 電源コネクタ取り付けの前処理

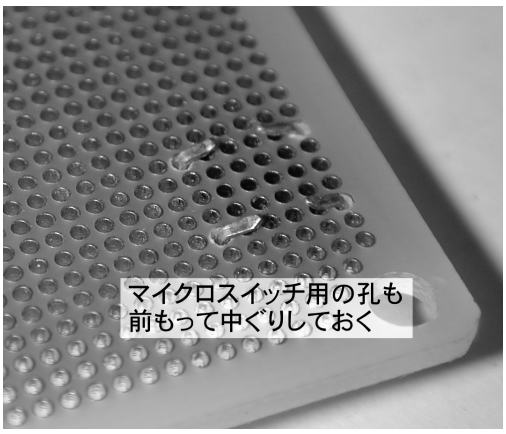


図 6 マイクロスイッチ取り付け時の注意

ベースボード

各コンピューターユニットを固定するのには樹脂板を使うことにしました。ただ、生来のサボり癖のためにベースボードを作るところまでは進めませんでした。ただ、どんな樹脂を使えばよいか調査は行なったのでその結果について記します。

- ベークライト：廉価なプリント基板の素材として使われています。
- アクリル (ポリメチルメタクリレート)：非晶質なので光学的異方性が出にくく、透明度が高い素材です。 LD のような大判の光学メディアに使われています。しかし非晶質ゆえに脆かったり水分が比較的透過しやすかったりします
- ABS：アクリル (アクリロニトリル) －ブタジエン－スチレンの混練物で、それぞれの成分が互いの界面で化学的に結合しています。耐衝撃性に勝れています
- ポリカーボネート：配向性があるので LD のような大判の光学メディアには使いにくい

素材です。そのため多くの場合 CD に使われています。静的強度は PMMA に劣りますが耐衝撃性には優れています

- ポリアセタール：ジュラコンなどの商品名で販売されている素材です

検索エンジンで調べたところ、建築模型の材料を扱っているショップがでプラスチックシートを扱っているようでした。例えば日本エマ

http://www.twics.com/~jema/

http://www.twics.com/~ema/ptlist02.html

そしてケイエス商事

http://www.nscs-net.ne.jp/ks/index.html

しかし、樹脂シートの強度は樹脂の分子量や組成によって変わります。模型材料としての樹脂シートの特性が今回のような、比較的重いユニットを支えるものとして役に立つかわかりにくい部分があります。

ポリカーボネートは旭硝子が販売しています。

http://www.agc.co.jp/polycarbonate/kakaku1.html

ネットワーク

ハブは NETGEAR 社の FS2108 を使うことにしました。手に入るもので廉価なものなら何でもいいと思います。ケーブルも安売りされている 10 BASE-T のもので十分です。

電源

ATX の電源規格はインターネット上で見つけられます。電源起動のタイミングを次の図 7 に示します。

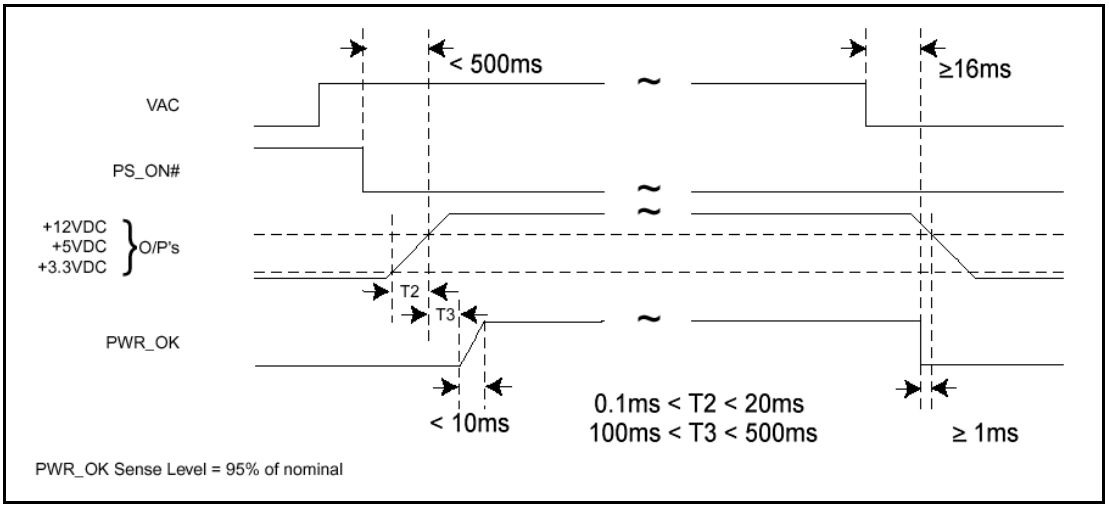


図 7 ATX 電源起動のタイミング (ATX_ATX12V_PS_1_1.pdf より抜粋・編集)

作成した電源制御回路を図 8 に示します。

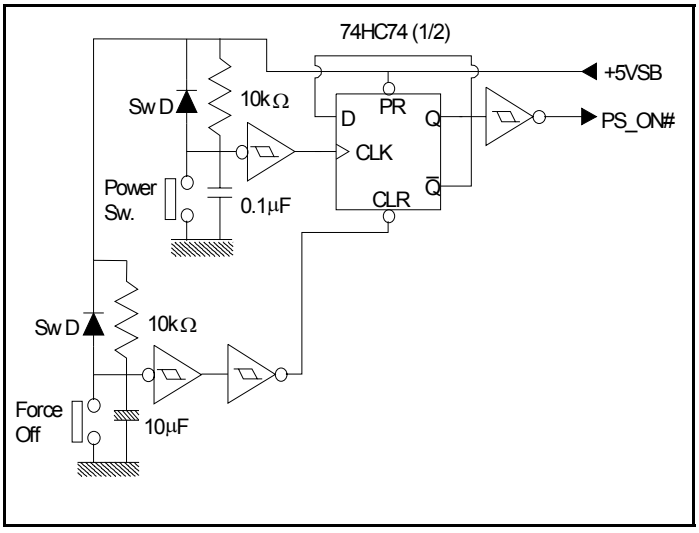


図 8 電源制御部の回路

ATX 電源のコネクタは大阪の共立電子で売っていました。ATX 電源コネクタのピンのピッチは約 4.2mm 間隔で 1.27mm ピッチの蛇の目基板とはどうしても孔の位置が合いません。そこで図 9 に示すように場所によっては隣接する二つの孔を広げて対応しました。図中の記号の意味をそれぞれ以下に示します。

- 白抜きの丸：蛇の目基板本来の孔
- 灰色の丸および長円？：ドリルで拡張した孔
- 黒いコの字：ATX 電源から出ているピン

茶色のように孔を拡張することで基板上に ATX 電源ソケットを取り付けられるようになります。

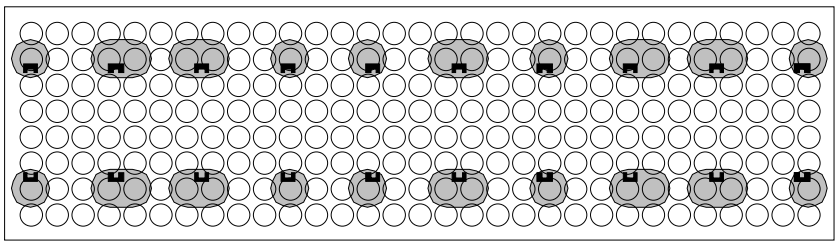


図 10 電源ソケット取り付けのための孔あけの模式図

キーボード・マウス・モニタ

PC カードにキーボードを取り付けていない場合、異常を感知して OS の軌道まで進まない場合があるようです。現時点では一台ずつキーボードとマウスを取り付け、リセットを押し

て起動するようにしていますがスロットに刺さった基板を押さえつつコネクタを抜き差しするのは気持ちよくありません。市販のパソコン切り替え器を使う手もありますが 4 to 1 程度のものしかありませんし、それでも比較的高価です。PIC のようなワンチップマイコンを使う手もあり、**“あたかも最初からどのユニットにもキーボードやマウスが取り付けられているように感じさせる”**にはワンチップマイコンを使う方法が一番よいのですが私にワンチップマイコンを扱う技量がありません。

そこで現在は起動する毎にコネクタを抜き差しする (本当はよくないが) ことにします。しかし将来的には CMOS アナログマルチプレクサを使った切り替え器を使うことを考えています。次の図 10 にその回路図を示します。マウス・キーボードともに同一の回路が必要になるので実際の回路はロータリー Sw の部分を除いてもう一組必要になります。

なお、AT および PS/2 キーボード・マウスの仕様について書いてあるサイトを見つけたので参考資料の欄に示しました。

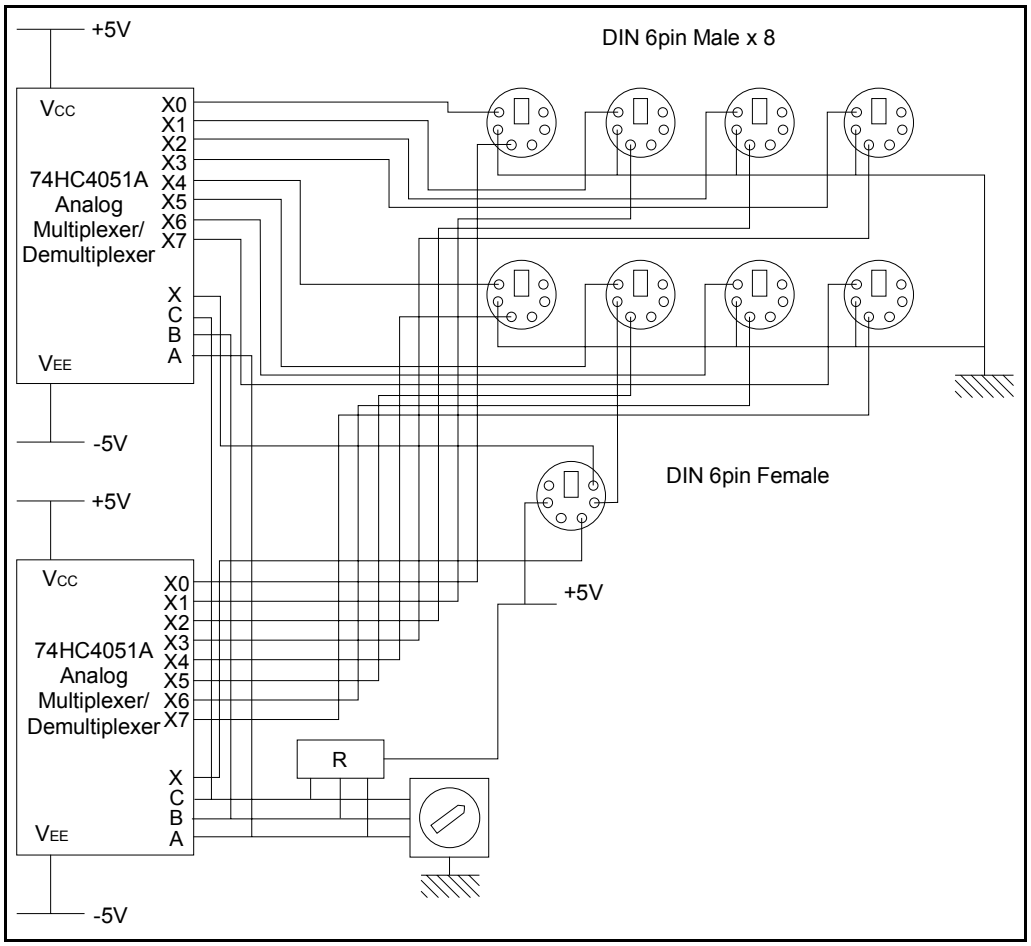


図 10 キーボード／マウスの切り替え回路 (1/2) enable (図中では省略) は常にアクティブ

モニタは取り付けてなくても PC は起動するので、モニタの分配器は作成する予定はありません。

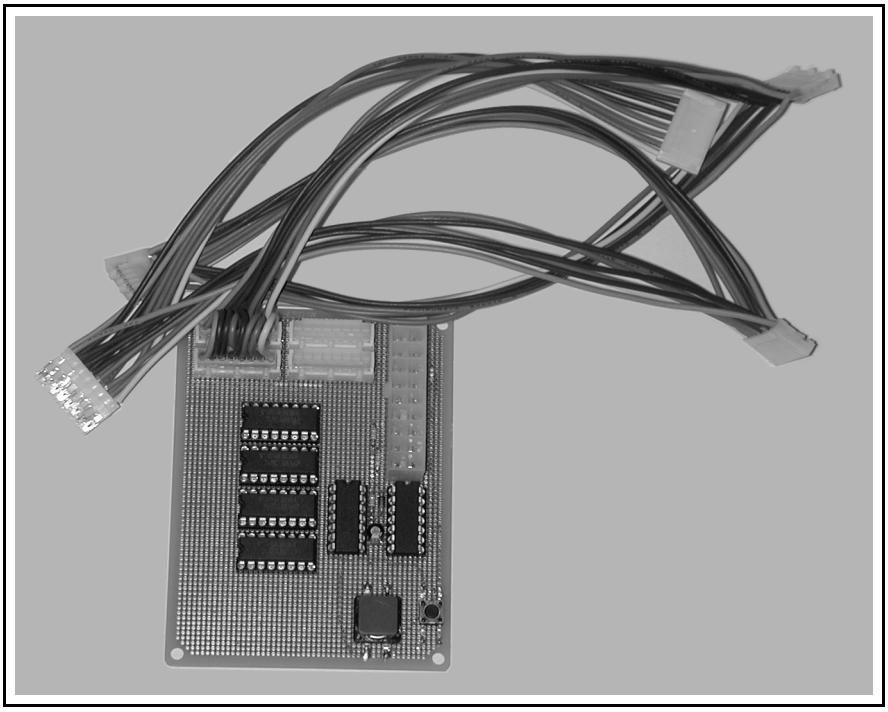


図 11 電源制御部　洒落で 74HC4051A も乗せてみました

起動

各バックプレーンボードを固定せずに PC を取り付け、動作させている様子を図 12 に示します。基板の裏面でのショートを避けるためにスペーサーを介してブランクな蛇の目基板を取り付けました。

思い切って電源を入れると…ショートしてしまったようです。この基板、電源制御部のみでの起動は確認していました。ということは電源の配線を間違えている可能性が大です。配線の間違い確認をもっとじっくりやるべきだったのですが、あせってしまいました。

これらのカード PC は手元で余っている ATX マザーボード (Dual CPU 用だが電気的に、あるいは物理的に片方のソケットが使えなくなっているので Single CPU で動かしている) のスロットに挿して使っていました。改めて電源を購入して確実に動かしつつ、ベースボードまで含めたクラスタ PC の実現を目指そうと思います。

最後に

久々にはんだ付けをやったので楽しめました。電源を吹っ飛ばしたのはショックでしたがこの 1 年 PC パーツを壊しまくったのでその続きと思えば…ハハハ (涙)。ともあれ、12 ペー

ジも使っておきながら『動きませんでした』は非常に問題なので、原因を突き止め、どすぶいだにょ☆第 3 巻がもし出るならその誌上で、出なくとも私の Web ページで片をつけたいと思います。

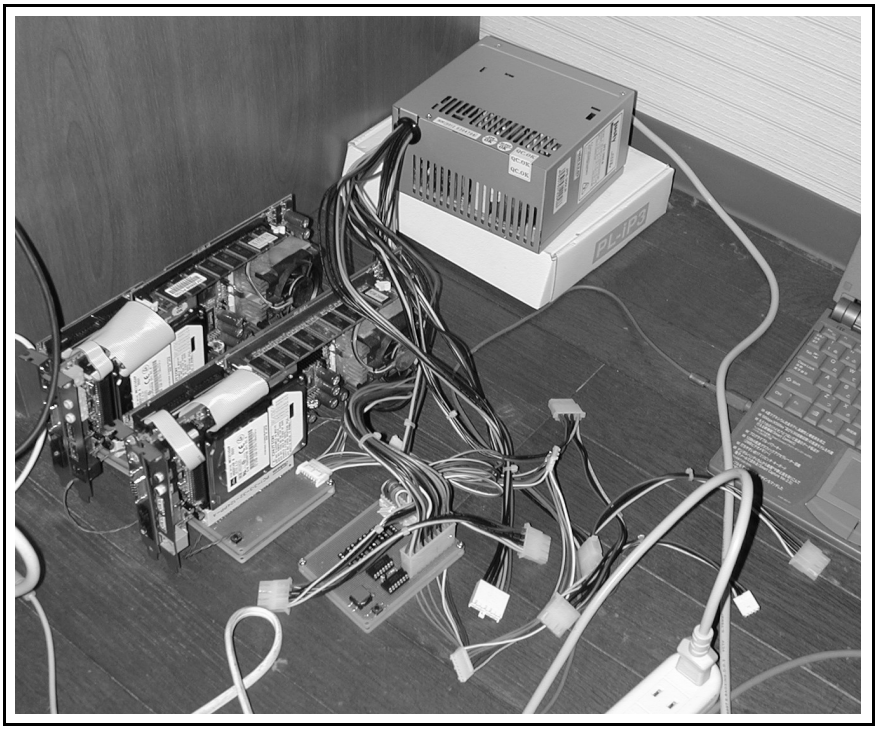


図 12 本来ならクラスタ動作の勇姿となるはずが…

しかしこの PC カード、800MHz までの Pentium III/Celeron に対応しているということですが 800MHz の PIII は既に販売されておらず、Celeron にしてもどれを乗せてよいやら悪いのやらさっぱり分からない…たぶん FSB 66MHz のなら大丈夫なのでしょうけど…という状態です。それにこのカード PC はコムサテライト 3 で購入していたんですが 12/2 に行ってみたらもう置いてませんでした。こんなことでクラスタユニットを増やせるのかな……

参考資料

窓の社での分散型コンピューティングの紹介

http://www.forest.impress.co.jp/article/2001/11/08/distributed.html

PCI のピンアサインに関する資料

トランジスタ技術 SPECIAL No. 65, PCI バスの基礎と応用　CQ 出版

TECH I (テックアイ) Vol. 3, PCI デバイス設計入門　CQ 出版

ATX 電源 (だけではないが) に関する資料

ATX/ATX12V Power Supply Design GuIde　Version 1.1

(http://www.formfactors.org/developer/specs/atx/ATX_ATX12V_PS_1_1.pdf)

ATX Specification

(http://www.formfactors.org/developer/specs/atx/atx2_03p1.pdf)

PS/2 キーボード・マウスインターフェース仕様

http://panda.cs.ndsu.nodak.edu/~achapwes/PICmicro/PS2/ps2.htm

http://panda.cs.ndsu.nodak.edu/~achapwes/PICmicro/mouse/mouse.html

74 シリーズのピンアサイン調査に用いた資料

Fairchild Semiconductor Products Home Page

(http://www.fairchildsemi.com/products/index.html)

CMOS アナログマルチプレクサ／デマルチプレクサ 74HC4051A のデータシートは

http://www.onsemi.com/pub/prod/0,1824,productsm_Taxonomy_MaxLevel=2_Subtype=Product,00.html より

http://www.onsemi.com/pub/Collateral/MC74HC4051A-D.PDF

PC－9801にVGAモニターを！

きよりん　kiyonari@cds.ne.jp

CPU の動作周波数が 2GHz を超えようとしている昨今ですが、メインに使用するパソコン以外に、いまだ"国民機"PC-9801 をなんらかの事情で使用されている方も多いと思います。私もそのひとりで、以前、トラ技の基板配布で作成した 9801 の C バスにインターフェース基板を刺す ROM ライターを使用しています。 また、今では 9801 でしか楽しめない旧作ゲームもあり、9801 を手放さずに愛用してきました。

しかし最近、メインの AT 互換機と 9801 とで共用させていたディスプレイである三菱 RD-17G が故障してしまい、修理するには修理サービス人件費のコストアップもあり、どうみても二~三万円はかかりそうでした。ところが、17 インチ CRT を使ったディスプレイは激しい価格競争を行っており、三万円から四万円を出せば RD-17G よりもずっときれいでシャープな表示の高性能ディスプレイが買えてしまい、これでは修理することはナンセンスになってしまいます。

ところが、ご存じのように PC-9801 シリーズの CRT への水平同期周波数は 24.8KHz で、RD-17G はこれに対応するモードを持っていました。しかし、現在、販売されているディスプレイでこの 24KHz モードを持っているものは皆無と言ってよい状態です。そこで、解決方法としては、以下の物が候補に挙がりました。

１)9801 専用のモニターを使う。 しかし、腐海部屋と化している我が家では 9801 専用にディスプレイの置き場所を確保することは不可能に近い状態です。

２)VGA の 31KHz の出力をサポートしている PC-9821 シリーズの中古品を購入する。秋葉原のジャンク屋などではペンティアム 120MHz 程度の PC-9821 が三、四千円程度で入手できますが、一部の機種を除いてゲームなどを楽しむ場合必須である FM 音源が無いので FM 音源ボードである PC-9801-86 を入手する必要があります。

３)PC-9801 の 24KHz 出力を 31KHz 出力に改造する。

今回はこの方法を実行してみることにしました。

9801 の CRT コントローラは水平同期出力の 24.8KHz を 21.0526MHz のクリスタルオシレーターから作り出しています。この 21.0526MHz のオシレーターを 26MHz あたりの適当な周波数のオシレーターに交換することで 31KHz を出力できるようになることは以前から知られており、また、改造方法も公開されてきました。しかし、この半端な周波数のオシレーターを入手することは困難で、秋葉原の店頭にも在庫はなく、発注するにしてもある程度の個数がまとまらないと受付をしてもらません。つまり、改造方法はわかっているが部品が入手できずに製作不可能だったわけです。ところが最近、アマチュアには大変ありがたい物が出回るようになりました。それは　EPSON　が発売している「プログラマブル水晶発振器」というもので、EEP-ROM を焼くように目的の周波数を指定してプログラムできるものです。(文末に資料参考)　つまり一個だけでもオリジナルな周波数の水晶発振器を手に入れることができるわけで、価格も作成手数料を含めて一個千円と実に入手しやすいものです。そこでこれを使って 9801 を改造することにしました。

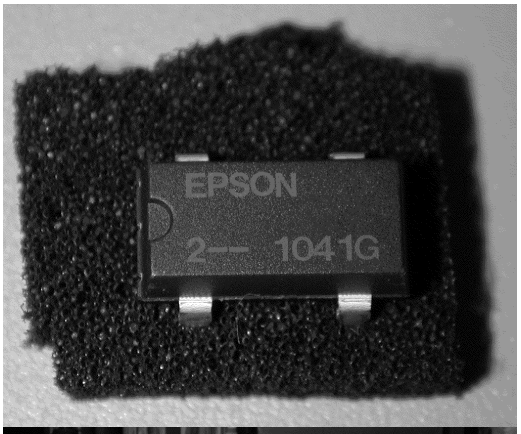
まず交換用のオシレーターを購入します。私の場合、秋葉原で製品の代理店になっている「田中無線電機」で購入しました。お店の発注用紙に希望の周波数、仕様を記入して店頭で申し込むと特にお店が忙しくない場合、30 分程度でてき上がります。

目的とする周波数ですが、私の場合、新しく購入して来たディスプレイが 31KHz からのマルチスキャンですので 31KHz より上の周波数が出ればよいわけで、31 ÷ 24.8 ＝ 1.25　つまり、21.0526MHz を 1.25 倍すればよいことになります。で、すこしディスプレイ側の余裕を見て 26.650MHz で註文することにしました。この周波数ですが、あまり周波数をあげても 9801 側が対応しきれず動作しない可能性があります。

発注した仕様は、

周波数	26.650MHz
形状	DC タイプ
一番ピン仕様	OE
動作電圧	5V　TTL

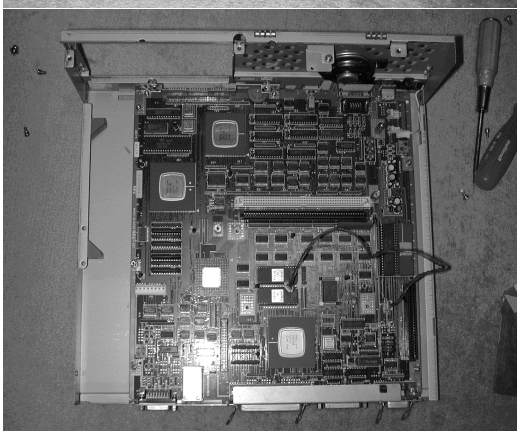
というものでした。



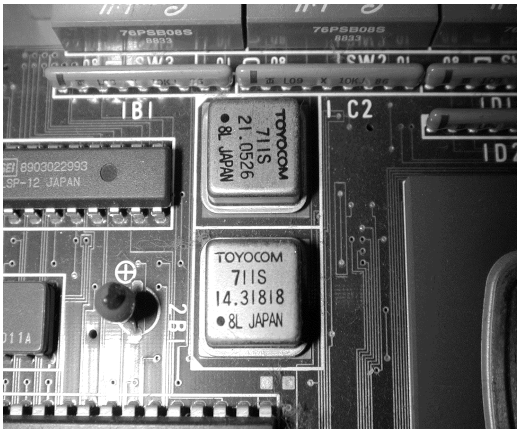
販売代理店を紹介しておきます。

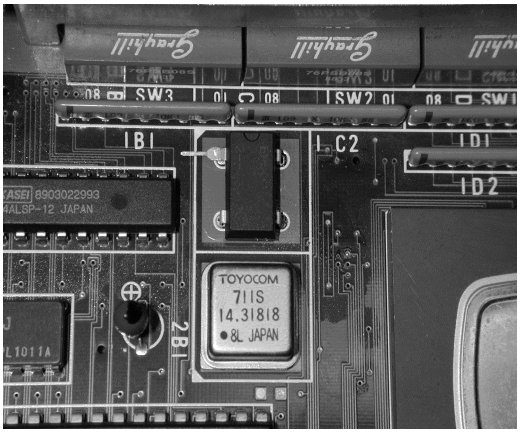
田中無線電機(株)
東京都千代田区外神田 3-13-7
(ラジオセンター 1F)
TEL　03-3253-3201
E-mail　parts@tanakamusen.co.jp

9801 を分解して目的のクリスタルオシレーターがあるメイン基板を取り出します。そして、21.0526MHz のクリスタルオシレーターを探します。作成例では PC-9801RA を使用しました。この機種ではオシレーターは FDD ドライブの下の部分にありました。

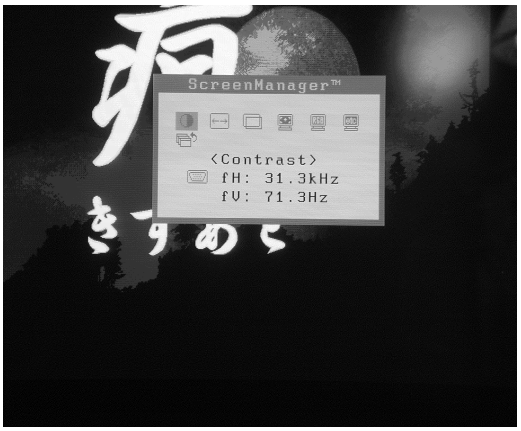


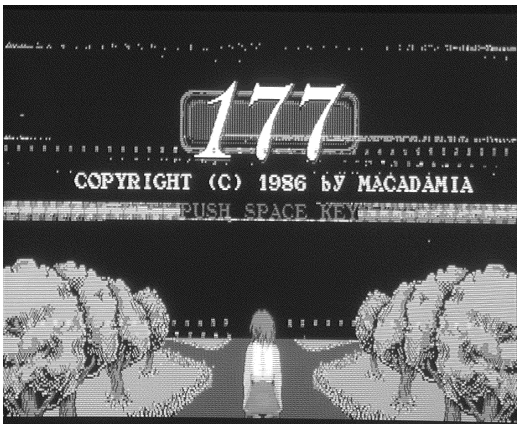
オシレーターを電動ハンダ吸い取り機などを使って取り外します。吸い取り機が無いときは 40~60W 位のハンダごてをピンに当てて加熱し、少しずつオシレーターを浮かせるようにして慎重に取り外します。このオシレーターの取り外しがうまくできるかどうかがこの改造の最大のポイントになります。取り外したらスルーホールの余計なハンダをきれいに取り除いておきます。細かなパターンが集中していますのでパターンを傷つけないように注意します。





新しいオシレーターを取り付けます。一番ピンの[OE]は元のオシレーターと同じように開放にしておいてかまいません。オシレーターを取り付けたら PC-9801 を元どおりに組み立てます。





VGA 対応のディスプレイに接続して動作試験を行います。モニターケーブルは PC-9801-VGA 対応のものか、コネクター変換アダプターを使います。

わたしの例では、テキスト表示は正常に行えるのですが、一部の、グラフィックスを使ったゲームで画面に乱れが生じました。これは、ゲームの中には CRT コントローラーの同期割り込みを使ったものが、周波数が変わることによりタイミングが変わるためにおこる物と思われます。

今回の改造はどうしても無理やり動かしているという印象が強いのですが、とりあえず最小の投資で 9801 をスクラップにせずにすみました。

SG-8002シリーズスペック一覧表

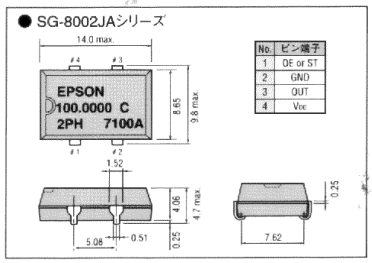
■仕　様（特性）

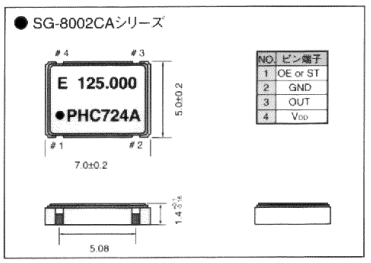
項目		記号	仕様 PT/ ST	仕様 PH/ SH	仕様 PC/ SC	条件
出力周波数範囲		f_0	1.0000MHz ～125.0000MHz		1.0000MHz ～106.2500MHz	
電源電圧	最大供給電圧	V_{DD}-GND	−0.5V～+7.0V			
	動作電圧	V_{DD}	5.0V±0.5V		3.3V±0.3V	3.0V±0.3V: f_0≦66.7MHz(PC/SC)
温度範囲	保存温度	T_{STG}	−55℃～+125℃			DB, DC, JA, CAパッケージ
			−55℃～+100℃			JCパッケージ
	動作温度	T_{OPR}	−20℃～+70℃(−40℃～85℃)		−40℃～+85℃	DB, DC, JA, CAパッケージ
			−20℃～+70℃			JCパッケージ
ハンダ付け条件		T_{SOL}	260℃以下×10秒以内			
周波数安定度		Δf / f_0	B：±50ppm、C：±100ppm、M：±100ppm(−40℃～85℃)			−20℃～+70℃(JCはC, B精度のみ)
消費電流		I_{op}	45mA max.		28mA max.	無負荷、最大出力周波数範囲
ディセーブル時電流		I_{OE}	30mA max.		16mA max.	OE=GND（PT,PH,PC）
スタンバイ時電流		I_{ST}	50 μA max.			ST=GND（ST,SH,SC）
デューティ		tw / t	−	40%～60%		C-MOS負荷:1/2V_{DD} レベル, 最大負荷時
			40%～60%	−		TTL負荷: 1.4Vレベル, 最大負荷時
"H" レベル出力電圧		V_{OH}	V_{DD} -0.4V min.			I_{OH}=−16mA（PT/ST,PH/SH）,−8mA（PC/SC）
"L" レベル出力電圧		V_{OL}	0.4V max.			I_{OL}=16mA（PT/ST,PH/SH）, 8mA（PC/SC）
出力負荷条件（TTL）		N	5TTL max.	−		最大出力周波数, 最大動作電圧範囲
出力負荷条件（C-MOS）		CL	15pF max.	25pF max.	15pF max.	
"H" レベル入力電圧		V_{IH}	2.0 V min.		0.7 × V_{DD} min.	OE端子,ST端子
"L" レベル入力電圧		V_{IL}	0.8 V max.		0.2 × V_{DD} max.	OE端子,ST端子
出力上昇時間		t_{TLH}	−	4ns max.		C-MOS負荷:20%→80%V_{DD}レベル
			4ns max.	−		TTL負荷:0.4V→2.4Vレベル
出力下降時間		t_{THL}	−	4ns max.		C-MOS負荷:80%→20%V_{DD}レベル
			4ns max.	−		TTL負荷:2.4V→0.4Vレベル
発振開始時間		t_{OSC}	10ms max.			最小値動作電圧のtを0とする
経時変化		f_a	±5 ppm / 年 max.			Ta=25℃,V_{DD}=5.0V/ 3.3V(PC/SC)
耐衝撃性		S.R.	±20ppm max.			硬木上75cm×3回または3000G ×0.3ms.×1 / 2 Sine Wave×3方向

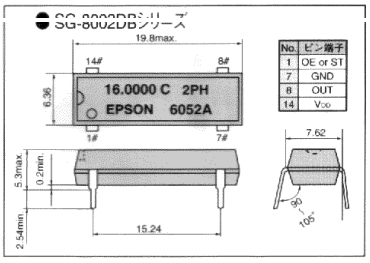
注意事項：お客様のお使いになる用途によっては使用できない場合がありますのでご了承ください
周波数によってはご利用できない場合がありますので前もってご確認ください
詳しいスペックはお問い合わせください

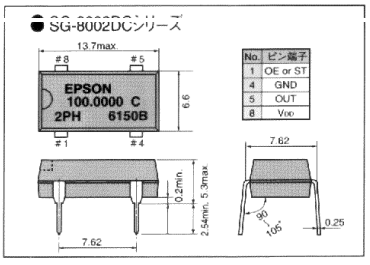
■外形寸法図

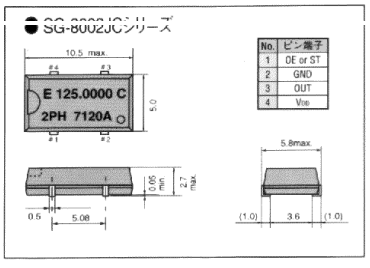
（単位：mm）











イーサネットボードの製作 (ハードウェア編)

本文・挿絵：ChaN

最近流行の Ethernet 応用の製作です。今回は実際に製作例を紹介するハードウェア編と、ハードウェアには依存しないプロトコル編に分けて解説します。

この手の製作モノは、流行り始めてからずいぶん経っているので、もはや目新しさもなくなってしまいました。ハードウェアの製作例は既に巷に氾濫しているので、改めて解説するまでもありません。しかし、肝心なプロトコルに関しては、詳しくまとまった解説がまだほとんどないのが現状です。したがって、今回はハードウェアに関しては製作の一例を取り上げるだけにして、プロトコルの動作の解説の方に重点を置いてみようと思います。

主要プロトコルの動作原理だけでも理解できると、ネットワーク全体のしくみが手に取るように分かるので、単に電子工作だけでなく、パソコンや WS、ネットワークを管理していくうえでも大きな力となります。

それでは、まずハードウェアの製作を軽く紹介しましょう。

ハードウェア

最近流行っている、PIC や H8、SH などごく小規模なワンチップマイコンを使って LAN に接続する Ethernet コントローラボードの製作です。

NIC チップ等の具体的な制御方法については、既にトラ技をはじめとする各種技術誌で詳しく解説されていますので、ここでは製作するボードの概要についてだけ解説します。その他必要な情報については、章末に示す各種文献を参照してください。

●主要部品

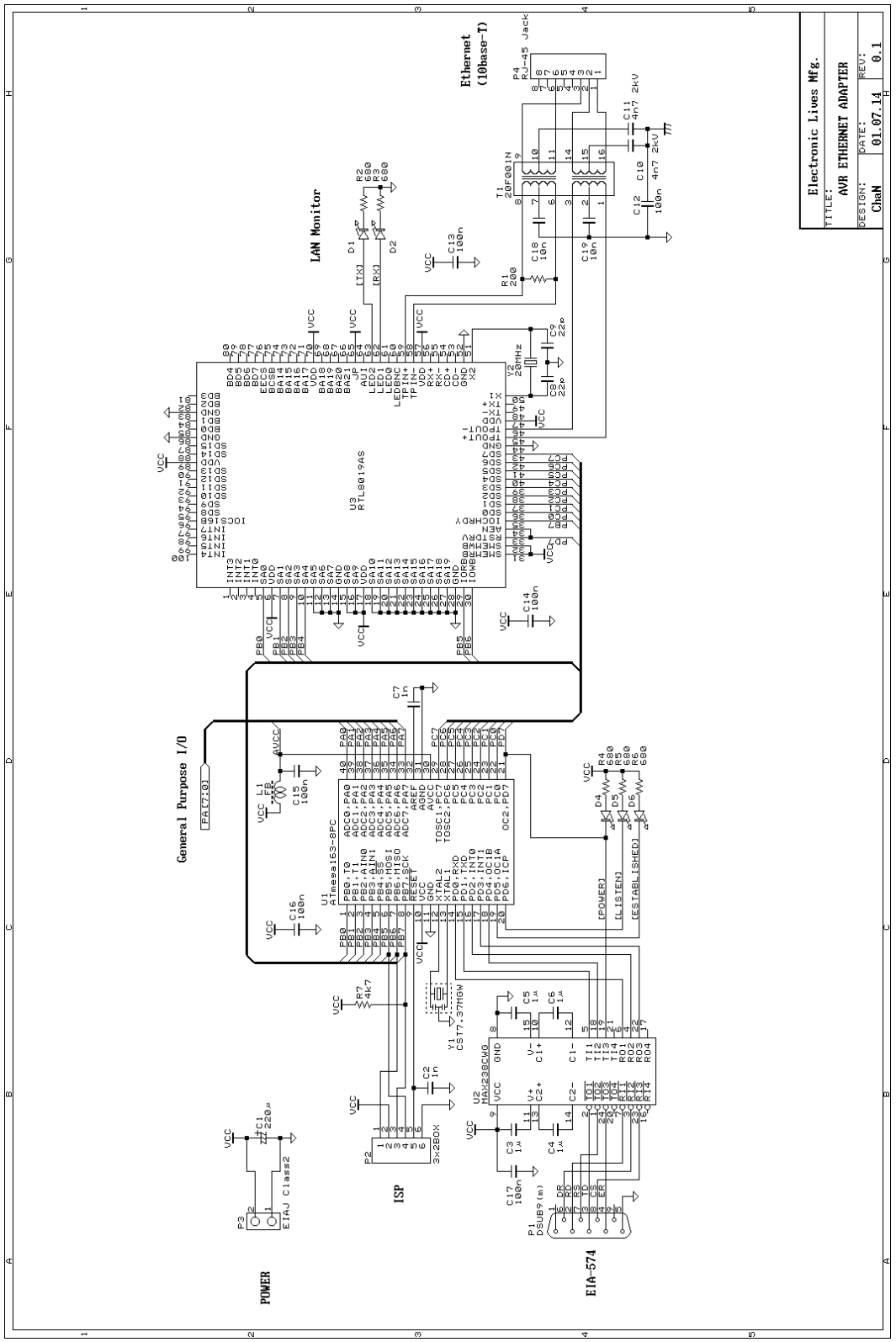
この手のモノは、秋月から出ているキットが有名で、誰もが知っていることでしょう。秋月のは、ホストコントローラになんと Microchip の PIC を使用しています。仕様的にはかなり割り切った不完全なインプリメントとなっているものの、ちゃんと TCP/IP が乗っています。あれは職人技です(^^;。

今回はプロトコルを実験してみるのが目的なので、PIC より使い勝手が良くて高性能な Atmel の AVR を使ってみます。使用するチップは、AVR ファミリの中でもハイエンドになる Mega シリーズから **ATmega163** を使用します。Mega シリーズは最近になって秋葉にも出回るようになってきました。

まぁ、各種プロトコルをフル実装したまともな Ethenet コントローラとして設計するには、メモリ空間の広いマイコン（最低でも H8 クラス）を使用するのが一般的ですが。

Ethernet コントローラ（NIC チップ）には、これまた有名な Realtek の **RTL8019AS** を使用します。蟹のロゴが特徴で、いわゆる「**蟹チップ**」と呼ばれているものです。これも秋葉では容易に入手できます。

ただ、RTL8019AS は主流と言うより、ISA が廃れてしまって、供給されている ISA 用チップセットが限られてしまっているからよく使われる…とも言えますが。組込用としてよく使われている NIC チップは、



これと Crystal Semiconductor(現在は Cirruslogic)の CS8900A くらいです。ISA が消えていく中、これらもいつまで供給され続けるかは不明です(^^;。当然ですが、PCI 用の NIC チップを普通のマイコンに接続するのはかなり困難で、自作には向いていません。

主要部品は以上です。下の図に今回製作した Ethernet コントローラの回路図を示します。

●ホストインターフェース

Ethernet コントローラには、リモート制御される上位のコントローラや、デバッグ用に端末を接続するためのシリアルポート(EIA-574)を設けてあります。シリアルフォーマットは任意ですが、LAN 側の速度とのバランスからしてなるべく速い方が良いので、最大 115.2kbps での通信に対応できるようにします。この速度まで誤差のないビットレートにするため、AVR のクロックは 7.37MHz としています。

また、モデムライクな使い方もできるようにするため、RI を除く制御信号も全て制御できるようにしました。

●NIC チップ

言わずと知れた RTL8019AS です。この IC は、ISA バス互換のバスインターフェースを持ち、単品で LAN カードを構成できます。また、NE2000 という有名な LAN カードと互換性のある内蔵レジスタを持っていて、これと制御方法が同じになっています。つまり、**ホスト側から見ると NE2000 カードそのもの**なのです。このため、使い方に関するノウハウの蓄積も多く、この辺のことも他のチップより多く使われる理由かも知れません。

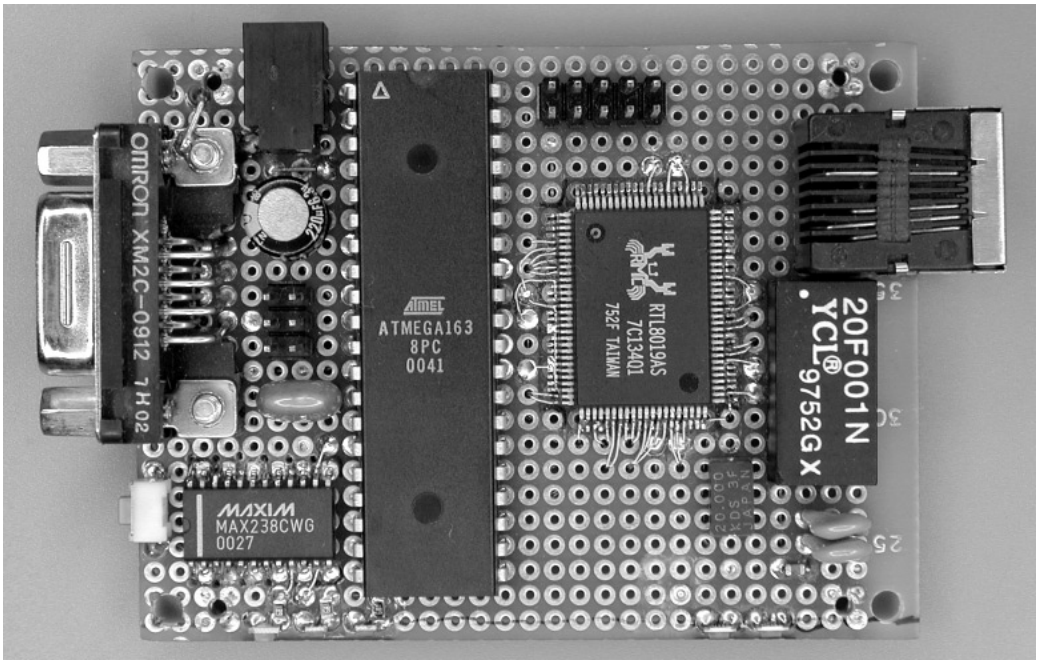
RTL8019AS のホスト I/F は ISA バス互換ですが、内蔵レジスタは I/O 空間(デフォルトでは 0x0300～0x031F)にマッピングされていて DMA も使わないので普通のマイコンのバスにも容易に直結可能です。また、今回使用した AVR のように、外部バスの無いマイコンでも I/O ポートでバスと制御信号を叩けば同様に制御できます。

RTL8019AS のバス幅は、16bit ですが、リセット時に IOCS16B ピンをオープンにしておけば自動的に 8bit モードでの動作となります。

RTL8019AS は、**自動コンフィグレーション用にシリアル EEPROM を接続**できるようになっていて、普通はこれによって I/O アドレスが設定されたり、物理アドレス(後述)をホストコントローラに示したりします。中でも重要なのは物理アドレスで、チップ初期化時にホストコントローラがこれを読み出して、受信フィルタの設定や送信パケットの送信元アドレスとするなど自ノードの物理アドレスとして使用されます。

物理アドレスをホストコントローラ側で正しく記憶しておけるのなら、EEPROM は不要です。今回は、AVR の内蔵 EEPROM に記憶して、**シリアル EEPROM は使用しません**。シリアル EEPROM を接

製作した Ethenet コントローラ



続しなかった場合、NIC チップの I/O アドレスはデフォルトの 0x0300～となります。LED 出力の定義もデフォルト (Tx,Rx,Col)となり、Link 表示が使えませんが、これは仕方ないでしょう。

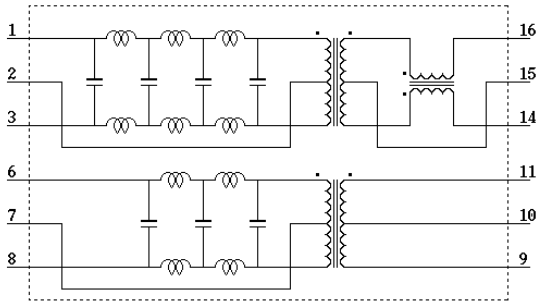
NIC チップに供給されるクロックには IEEE802.3 のスペックから、**±100ppm** が要求されます。セラミック発振子を使ってはいけません。水晶発振子でもちゃんと吟味して使わないと規格を外れてしまいます。クロック周波数(送信ビットレート)が規格を外れていると、長いパケットでデータエラーを起こす原因となります。

●パルストランス

Ethernet の良いところは、**絶縁型インターフェース**であるということです。これは耐ノイズ性や保安上の理由からも好ましいことです。実際、離れた場所に配置された何十台ものパソコン(普通はグランドレベルが保証されていない)が非絶縁インターフェースを介して接触していたりしたら、危険この上ありません。これほど手軽な絶縁型インターフェースはほかにないといえるでしょう。

同軸系メディアにしてもツイストペア系メディアにしても、絶縁はパルストランスによって実現されています。回路図ではパルストランスの中身が入出力対称に描かれていますが、実際にはただのパルストランスではなく、下の図に示すようにフィルタやコモンモードチョークが組み込まれています。したがって、極性はもちろんのこと、入出力方向にも注意が必要です。この図では、右側が伝送メディア側、上側が出力部です。

パルストランスの内部回路



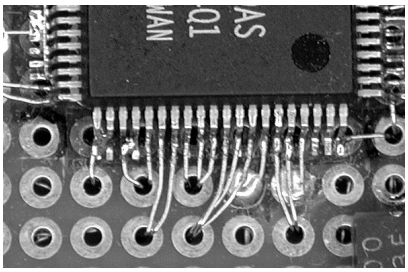
出力部に組み込まれたフィルタは、IEEE802.3 で規定される送信出力の高調波成分の制限をクリアするためのものです。トランスはコントローラからは矩形波で駆動されますが、高次のフィルタによって高調波が十分に減衰され、出力波形はほとんど正弦波となっています。入力部のフィルタは、受信信号に乗った高周波ノイズを取り除くためのものです。

●組み立て

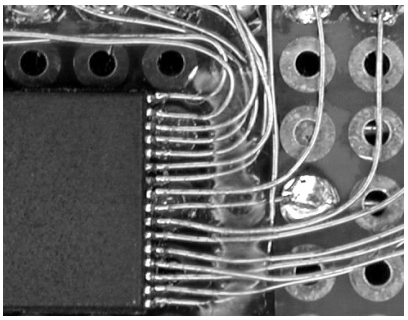
きわめて単純な回路なので、配線数も少なく特に難しいところはありません。しかし、手配線しにくい部品が多いので、マウントにちょっと苦労するかも知れません。主な部品とマウント方法を紹介しましょう。

QFP などについては、基板サイズにこだわらないなら各種変換基板を使うのがよいと思います。

◆QFP/TQFP：基板サイズ小型化のため、変換基板が使えず、直接配線しました。まず、基板にポリイミドテープを貼ってこの上に IC を接着します。そして、UEW でそれぞれのリードに直接半田付けしていくのです。この手は、0.5mm ピッチ程度までの配線に適用できます。難しそうですが、慣れればどうってことありません（↓）。

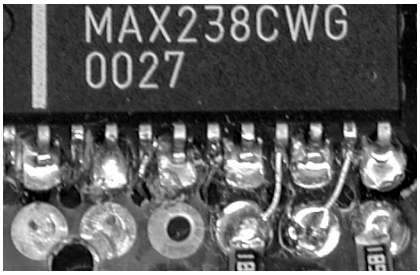


0.65mm



0.5mm

◆MFP：1.27mm ピッチなら穴のピッチに合うので、写真に示すようにランドを使って固定できます。ランドの間に来るリードと接触しないようにランドの一部を切ってから固定します（↓）。



◆RJ-45 ジャック：ピンの出方が変則なので、ピンが素直に刺さるようにピンバイスで基板に穴を開

けて、SuperX 等で基板に接着します。

ソフトウェア

ファームウェアの機能は、主に NIC チップを制御してパケットを送受信することと、ネットワークプロトコルの処理、それと対ホストインターフェース（モニタ機能）です。ここでは、あまり具体的な所には触れずに、概要を説明するにとどめておきます。基本的に、自分でアルゴリズムを考えてコーディングのできる人しか対象にしていませんので(^^;。

これらのプログラムは、アセンブラでコーディングしました。TCP/IP なんて本来アセンブラで書くようなシロモノではないのですが、リソースの限られたマイコンに効率よくインプリメントするにはアセンブラが最適です。まぁ、この辺はそれぞれの人の好みの問題ですが。普通は C でしょう。

●サポートプロトコル

今回は**各種プロトコルの実験**ということもあり、次に示すように必須とされるプロトコルに加え、よく使われる DHCP や DNS もインプリメントしてみます。

- Ethernet V2.0
- ARP
- ICMP(ping 応答)
- IP
- UDP
- TCP
- DHCP(クライアント 機能)
- DNS(クライアント 機能)

●プロトコル処理

普通のプロトコルスタックなら、必要なだけタスクを作ってアプリケーションからは独立して動作するものです。

今回使用する AVR の内蔵 RAM はたったの 1Kbyte です。これでは Ethernet で扱われるパケットサイズ（1500byte）にも足りませんし、タスクスタックすら確保することができません。それなので、この手のマイコンで TCP/IP をやる場合、正攻法ではダメで、かなり邪悪な手を使わなければ（時には規格をはしょって）なりません(^^;。

◆タスク割り

作れるタスクが限られているため、タスクの割り当てには少々工夫しなければなりません。

プロトコル処理のアクションは、主に**能動的動作**と**受動的動作**に分けられます。前者はアクションのトリガとなるイベントが自分側に起因するもので、上位からの指令によるパケット送信動作や、タイマによる状態遷移などです。後者は、ネットワークからのパケットの到着がトリガとなるアクションで、ARP や ping に対する応答や、各種受信データなどの処理です。

本機では、能動的動作と受動的動作にそれぞれタスクを割り当てるようにします。具体的には、能動的動作をメインルーチンにして、メインルーチンのイベント待ちループの中で受動的動作を呼び出して駆動することにします。これなら、タスク毎のスタックも必要ないので、限られた RAM でのインプリメントも比較的楽になります。

◆送受信バッファ

IP ミニマムとしては 576byte(+MAC ヘッダ)が要求されるので、パケット処理のワークエリアは約 600byte 必要になります。が、アプリケーションのワークエリアもまた必要なので、必要なパケットバッファを確保するのは困難です。このへんはもう適当なところで割り切るしかありません。

RTL8019AS には、外部からリモートアクセス可能な 16Kbyte のバッファ RAM が内蔵されています。これをうまく使うことによりメモリ不足を補うテクニックもあるようですが、今回はなるべく素直になるように、パケット全体をオンメモリ処理としています。

本機では、フォアグランド処理用とバックグランド処理用に 256byte づつ割り当てています。扱えるパケット長が短い場合、次のような弊害が発生するので、それぞれ対策を行っています。

BOOTP/DHCP：受信する IP パケット長が最大 512byte になることがあります。これらのパケットの送受信時は、不要なフィールドを飛ばしてメモリに取り込んで処理します。送信時は、NIC に渡すとき

Column

AVR ファミリの命令セットは C 言語でのコーディングを前提として決定されているので、C との親和性が大変良くなっています。

アトメルの調査によると、AVR ユーザの6 割は C でコーディングしているとのことです。ちなみに、アセンブリ言語は3 割で、残り1 割が BASIC やパスカルです。

に飛ばした部分をパディングします。
DNS : 受信パケットがバッファサイズを超えることがあるので、後ろの方の要らないフィールドを切って受信処理します。
TCP : ウィンドウサイズを受信バッファサイズ以下にします。

●実験に関して

プログラミング一般に言えることですが、動作を一つひとつ確認して理解しながら機能を追加していくと楽に進めていけます。

実際の開発をやるときは、デバッグ用のタスクを 1 個追加してしまうと良いです。これにより外部のデバッガが不要になります。

さすがに今回はこれ以上タスクを増やす余裕がありませんので、フォアグランドタスクにデバッグ機能を入れて共用にしています。

あと、ネットワークプロトコルのデバッグは、なるべく LAN の中で行うようにしましょう。

デバッグ中のプロトコルが変な動作をしてインターネットにゴミをばらまいたりすると、他のサイトの迷惑になることがあるので注意しなければなりません。

対サーバ通信の確認をするときなどは、LAN 上にサーバをでっち上げて使用するとよいでしょう。

デバッグモニタの動作例

●参考文献

(1) RTL8019AS , Realtek semi-conductor co., Ltd.
(2) DP83901A SNIC, National Semiconductor Corp.
(3) 10 Base T Low Pass Filter , YCL Electronics Co., Ltd.
(4) IEEE Std 802.3 2000 Edition, IEEE
(5) ATmega163, Atmel Corp.
(6) トラ技 1999.7 特集, インターネット時代のハード制御, CQ 出版

イーサネットボードの製作 (プロトコル編)

本文・挿絵：ChaN

Ethernetを利用した製作モノが流行っていて、電子工作もいよいよネットワーク時代に突入です。最近の32bitマイコン等の開発キットなら、プロトコルスタックも用意されているので、プロトコルの動作原理が分からなくても手軽にネットワークを活用できるレベルになっています。

しかし、自作精神でNICチップを買ってきてゼロから始めるとなると、各種プロトコルの動作原理を知り尽くしていないことには手も足も出ません。しかし、ほとんどの製作記事はページ数の制約のためかプロトコルに関しては漠然とした解説しかされていないので、それらを元にゼロから始めるのは難しいのが現状です。

ここではアプリケーションとハードウェアの間を補完する普段はブラックボックスとされているそれらプロトコル群について、『あとこれだけ知っていればとりあえず自作できる』という程度まで実際の例とともに解説します。私が実際にゼロからスタートするに当たり、足場を固めるためにRFCを読み漁ったり実際の動作を解析しながらまとめたものですので、これからプロトコルを書いてみようという方には少しは参考になるかもしれません。

NICチップ等ハードウェアの動作に関しては、それぞれ使用するデバイスによって異なってくるので、ここでは取り上げません。トラ技等の製作記事や関連デバイスのデータシートを参照した方が早いです。

Ethernetフレームフォーマット

実際にEthernet上を流れる最下位レベルのフレームフォーマットを右の図に示します。NICチップはハードウェア的にこのフレームフォーマットでのみ動作します。

送信の際、NICチップには宛先物理アドレスからデータ部までを渡せばよく、プリアンブルとFCSはNICチップにより付加されます。受信時は、宛先物理アドレスからFCSまでが受信データとして得られます。

NICチップ自体は受信データの先頭6バイトをパケットの宛先としてチェックするだけで、それ以外のフィールドは特に処理しません。キャリア停止で受信を終了してCRCレジスタの結果がエラー無しならそのパケットが受信バッファに取り込まれま

●DIX仕様(Ethernet V2.0) MACフレーム

プリアンブル (同期パターン)	64bit (有効フレーム長に含めない)
宛先物理アドレス	6byte (※)
送信元物理アドレス	6byte
フレームタイプ	2byte
データ部	46～1500byte
FCS	4byte

〈キャリア停止〉

す。

ここで説明する DIX 仕様(Ethernet V2.0) MAC フレームのほかに、これとは一部フィールドの意味が異なる IEEE 仕様(IEEE 802.3) MAC フレームもありますが、IP だけ使う分には特に必要ないのでここでは割愛します。詳しくは他の文献を参照してください。

◆プリアンブル：送信データ(NRZ)はマンチェスタコードに変換されてケーブルに送り出されます。プリアンブルは NIC チップにより付加される 10101010……10101011 のビット列で、レシーバ回路を同期させるためのフィールドです。有効なデータフィールドではないので、フレーム長には含めません。まず、1010…の繰り返しパターンで受信側のマンチェスタデコーダの PLL をビット同期させ、最後の 11 でフレームの開始を示し、そこから MAC フレームの受信が開始されます。

◆宛先物理アドレス: パケット先頭 48bit のフィールドで、この MAC フレームの送り先ノードの物理アドレスを示します。ユニキャストのほかマルチキャストの指定が可能。受信時は NIC チップがこのフィールドと自分のアドレスとを比較して、マッチした MAC フレームだけ取り込みます。

◆送信元物理アドレス：48bit のフィールドで、この MAC フレームの送り元ノードの物理アドレスをセットします。

◆フレームタイプ：データ部の内容を示します。主なものとしては、ARP パケット(0x0806)や IP パケット(0x0800)があり、実際に使われるのはこの2 つだけです。

IEEE 仕様 MAC フレームではこの部分がデータ長(～1500)となっていますが、DIX 仕様では 0x800(2048)以上の値を使うことにより互いを選り分けることができ、同じ Ethernet 上での共存を図っています。

同様に、このフィールドを 0x800 以上でかつほかと重複しない値で使用すれば、既存の IP ネットワークとは異なる独自仕様の通信システムを同じ Ethernet 上に共存させることもできます(普通やらないと思いますが)。

◆データ部：MAC フレームの長さは 64byte 以上 1518byte 以下と決められています。したがって、データ部の長さは 46byte から 1500byte までになります。下限が決められているのは、衝突検出が確実に働くようにするためです。必要なデータ長が 46byte 未満の場合は、適当にパディングしてデータ部を 46byte にする必要があります。

◆FCS: 送信時に NIC チップにより付加される 32bit CRC のフレームチェックサムです。生成多項式には AUTODIN Ⅱが使用されます。

●物理アドレスのフォーマット

物理アドレスは、**MAC アドレス、ハードウェアアドレス、ノード ID** などとも呼ばれている 48bit の数値で、Ethernet に接続されるそれぞれのノード(NIC)に与えられた固有のアドレスです。NIC チップはこの物理アドレスのみを認識して動作します。表記方法は MSB 側から順にバイト単位または中央で区切った 16 進数となります。NIC に貼られたラベルに見ることができるでしょう。

物理アドレスのタイプは大きく分けてユニキャストアドレスとマルチキャストアドレスがあります。前者は特定の一つのノードを示し、後者は複数のノードを示します。送信元アドレスは常にユニキャストでなければなりませんが、宛先アドレスにはマルチキャストも使用できます。ユニキャストアドレスは管理方法によってさらにユニバーサルアドレスとローカルアドレスに分けられます。

◆ユニバーサルアドレス

0	0	ベンダ ID (22bit)	ノード ID (24bit)

それぞれが唯一で重複のないアドレス。Ethernet の物理アドレス管理団体によってベンダに割り当てられるベンダ ID 部と、ベンダの管理するノード ID 部に分けられます。市販の NIC の殆どにユニバーサルアドレスが与えられています。

◆ローカルアドレス

0	1	

ローカル管理で自由に使ってよいアドレス。一部の妖しい市販 NIC でも使っていることがあります。このほかに先頭バイトが 0x02, 0x06, 0x0A, 0x0E のアドレスもローカルアドレスとして割り当てられているようです。

◆マルチキャストアドレス

1	

宛先アドレスとして、複数のノードを指定するアドレス。グループの指定方法はそれぞれの NIC によって異なります。

全てのビットが 1 のものは特に**ブロードキャストアドレス**といい、その MAC フレームが届く範囲の全てのノードを指定します。

Column

マルチバイト 値とバイト オーダー

Ethernet/ARP/IP/UDP/TCP など全般にわたり、基本的に**マルチバイト 値は Big Endian** で扱われます。本文中の説明も Big Endian をデフォルトとしているので、インテル系な人は十分な注意が必要です。次に主なパラメータフィールドと格納されるバイト順の例を挙げます

2byte 値： 1024 → 0x04, 0x00
4byte 値： 655360 → 0x00, 0x0A, 0x00, 0x00
IP アドレス： 64. 33. 57. 12 → 64, 33, 57, 12
物理アドレス： 00-33-10-84-06-C3 → 0x00, 0x33, 0x10, 0x84, 0x06, 0xC3

余談ですが、多くのパケットヘッダのフィールドにおいて、2byte 値はワード境界に、4byte 値はダブルワード境界にアライメントされるようになっています。これは、奇数アドレスにワードアクセスできないプロセッサが存在することから、それらへの配慮かと思います。

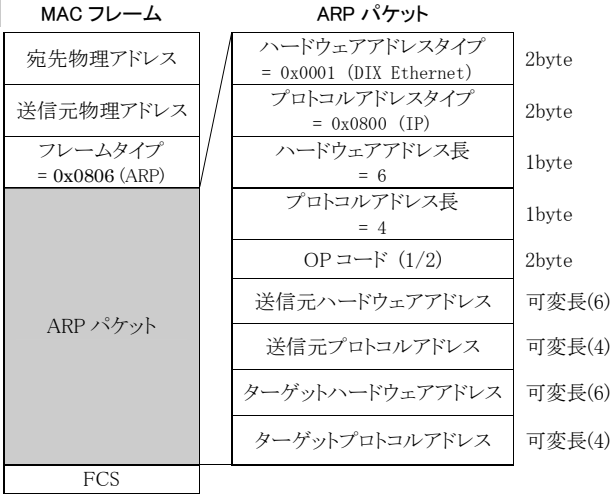
ARP(Address Resolution Protocol) RFC826

ARP は、Ethernet に IP パケットを通すための補助プロトコルで、直訳するとアドレス解決プロトコルとなります。ホスト機に実装された NIC 固有の 物理アドレスと、そのホストに割り当てられた IP アドレスとの間には何の関連性もありません。しかし、NIC 自体は物理アドレスでのみ動作するので、目的の IP アドレスのホストに IP パケット送るには、そのホストの NIC の物理アドレスを知らなければ送ることができません。この問い合わせを行うのが ARP です。

なお、相手の物理アドレスを知らなくても、宛先物理アドレスをブロードキャストにすれば IP パケットを送ることはできます。この場合、NIC によるフィルタリングが効かず全てのノードで受信されてしまいますが、宛先以外のホストでは IP アドレス不一致で捨てられます。しかし、関係ないホストに無用な負担をかけることになるので、特別な場合以外はやりません。

◆ハードウェアアドレスタイプ：DIX Ethernet では 0x0001。
◆プロトコルアドレスタイプ：IP では 0x0800。
◆ハードウェアアドレス長：DIX Ethernet では 6。
◆プロトコルアドレス長：IP では 4。
ハードウェアとプロトコルのタイプを特に指定するのは、ARP が Ethernet 以外のハードウェアや IP 以外のプロトコルも考慮しているため。
◆OP コード：ARP パケットの種類を示します。ARP リクエストは、1。ARP リプライは、2。

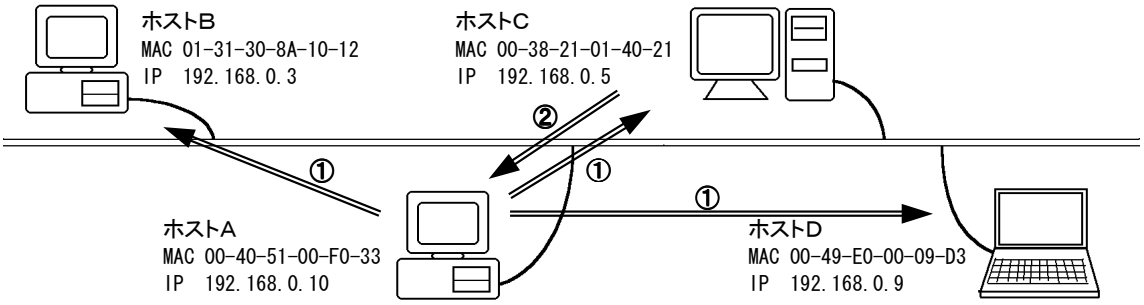
●ARP パケット のフォーマット



◆送信元ハードウェアアドレス：このパケットを送信したホストの物理アドレス。リクエスト時は自分の物理アドレスを入れ、リプライ時は回答（返送元の物理アドレス）が入ります。
◆送信元プロトコルアドレス：このパケットを送信したホストの IP アドレス。リクエスト時は自分の IP アドレスを入れます。
◆ターゲットハードウェアアドレス：このパケットの宛先の物理アドレス。リクエスト時は 0 で埋めます。
◆ターゲットプロトコルアドレス：このパケットの宛先の IP アドレス。リクエスト時は、調査対象の IP アドレスを入れます。

●ARP の動作例

ホストＡ（004051-00F033/192.168.0.10）が、ホストＣ（003821-014021/192.168.0.5）の物理アドレスを知りたい場合の例を示します。IP パケットをホストＣにユニキャストしようとしているホストＡは、まず宛先 IP アドレスに対応する物理アドレスが**ARP テーブル**にあるか調べて、無い場合は ARP によるアドレス解決を試みます。



①「この IP アドレスの人、物理アドレス教えてください」

アドレス解決をしたいホストＡは、そのネットワーク上の全てのノードに向けて ARP 要求パケットをブロードキャストします。

②「その IP アドレスは私のです。私の物理アドレスは xxx です」

ARP 要求パケットを受け取ったホストは、ターゲットプロトコルアドレスと自分の IP アドレスを比較します。そして、それが自分に対するものなら、その要求パケットを送ったホストに ARP 応答パケットで自分の物理アドレスを知らせます。

一般的に、要求パケットを送ったあと 100ms 以内に応答がなければそのアドレス解決は失敗とされて、IP パケットは送信されずに破棄されるようです。

ARP パケットの発生を極力減らすため、各ホストには IP アドレスと物理アドレスの対応を記録しておく ARP テーブルが設けられます。常に最新の対応になるように、ダイナミックエントリは一定時間（数分程度）で破棄されるようになっています。

	ARP 要求パケット①		ARP 応答パケット②
MAC ヘッダ	SRC = 00-40-51-00-F0-33 DST = FF-FF-FF-FF-FF-FF TYPE = 0x0806(ARP)		SRC = 00-38-21-01-40-21 DST = 00-40-51-00-F0-33 TYPE = 0x0806(ARP)
2byte	ハードウェアアドレスタイプ = 0x0001 (DIX Ethernet)		←
2byte	プロトコルアドレスタイプ = 0x0800 (IP)		←
1byte	ハードウェアアドレス長 = 6		←
1byte	プロトコルアドレス長 = 4		←
2byte	OP コード = 1 (リクエスト)		OP コード = 2 (リプライ)
6byte	送信元ハードウェアアドレス 00-40-51-00-F0-33		送信元ハードウェアアドレス **00-38-21-01-40-21**
4byte	送信元プロトコルアドレス 192.168.0.10		送信元プロトコルアドレス **192.168.0.5**
6byte	ターゲットハードウェアアドレス **空(0.0.0.0)**		ターゲットハードウェアアドレス 00-40-51-00-F0-33
4byte	ターゲットプロトコルアドレス **192.168.0.5**		ターゲットプロトコルアドレス 192.168.0.10

IP (Internet Protocol)　RFC791

IP は、その意味の示すとおりインターネットやイントラネットの中核をなすプロトコルです。IP のデータ伝送はパケット単位で行われ、個々のパケットは完全に独立したものとして伝送されます。機能的に IP はパケットを「ただひたすら送り先に届ける」と、それだけのことしかしません。飽くまで IP パケットを宛先へ届ける努力をするだけで、失われずに届くとか、化けずに届くとか、または送った順に届くとか、そういったことにはお構いなしです。したがって、IP レベルでは通信の信頼性があまりないので、エラー制御は IP の上位レイヤで持つ必要があります。

●IP パケット のフォーマット

次のページに IP パケットのフィールドフォーマットを示します。

◆バージョンとヘッダ長：バージョンはこの IP パケットのバージョンで、現在は4が使われています。長さはこの IP ヘッダの 4byte 単位の長さ。普通はオ

プション無しの 20byte なので、ヘッダ長の値は 5 となります。

◆**TOS** (Type of Service):ルータでこのパケットをどのように扱うかを指定するフラグです。普通は全てのビットをゼロ(つまり 全て通常)にします。

◆**トータル長**:この IP パケット(ヘッダ部+データ部)のバイト単位のサイズ。したがって、IP パケットのサイズは、最大 65535byte となります。

◆**ID 番号**:送信元で IP パケットが作成される毎に違う値がセットされます。分割された IP パケットはそれぞれが元と同じ ID 番号を持ちます。これは途中でフラグメント(分割)されて到着した IP パケットをデフラグメント(再結合)するための識別用に使われます。

◆**フラグ**と**フラグメントオフセット** :

0	DF	MF	フラグメントオフセット(13bit)

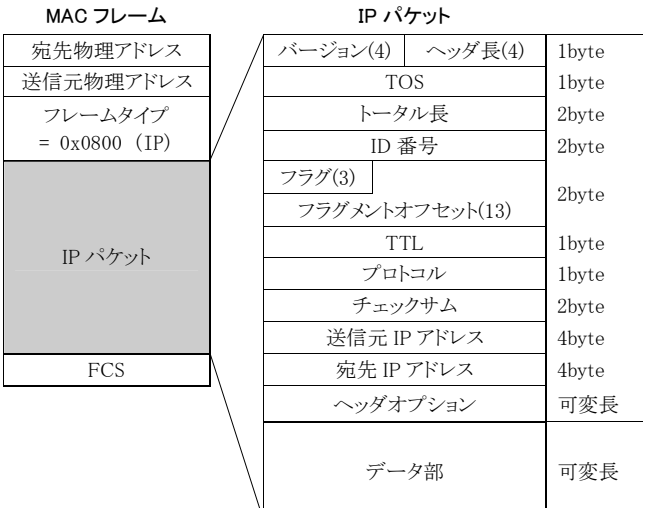
IP パケットは経路の MTU (Maximum Transfer Unit) の制限で分割される場合があります (IP フラグメントという)。例えば、Ethernet の MTU は 1500byte なので、これを越えるサイズの IP パケットはそのままでは Ethernet を通れません。そのような場合、そのパケットはいくつかの適当なサイズに分割されて送出されます (途中に MTU 値の小さい経路があれば、ルータによって分割される)。この分割・再結合は、IP の持つ機能で、IP の上位層は IP パケットがどのように伝送されてきたかは関知しません。

分割の際、それぞれの IP ヘッダのこのフィールドに分割情報が書き込まれます。オフセットは、そのデータが元の IP データ部のどの位置にあったかを 8byte 単位で指定します。つまり分割は 8byte 単位で行われます。**MF**(More flagments)フラグはさらに分割パケットが続くことを示します。**DF**(Don't flagment)フラグの立った IP パケットは分割されることはありません。しかし、分割しないと通過できない場合、その IP パケットは破棄されます。なお、IP パケットの経路は最低でも 576byte のパケットを分割することなしに通せることになっています。

◆**TTL** (Time to Live) : この IP パケットの秒単位の寿命。ルータを通過する際にかかった秒数だけ減じられ、0 になったらそのパケットは破棄されます。これにより宛先不明パケットがネットワークを無限にさまようのを防ぎます。ルータの通過時間が 1 秒に満たなくても少なくとも-1 されるので、**TTL 値は通過できるルータの数を示す**といえます。当然ですが、TTL が変わればチェックサムも再計算されます。

◆**プロトコル**:上位プロトコルを示します。つまり、データ部が何のプロトコルのパケットかを示します。代表的なものとして、ICMP(1)、TCP(6)、UDP(17)などがあります。 RFC1700

●IP パケット のフォーマット RFC791



◆**チェックサム**:適用範囲は IP ヘッダのみです。データ部は含まないので、データ部のエラーを検出することはできません。計算方法はコラムを参照してください。

◆**送信元 IP アドレス**:このパケットを送信したホストの IP アドレス。

◆**宛先 IP アドレス**:このパケットの宛先ホストの IP アドレス。ユニキャストのほか、マルチキャストやブロードキャストも可能です。

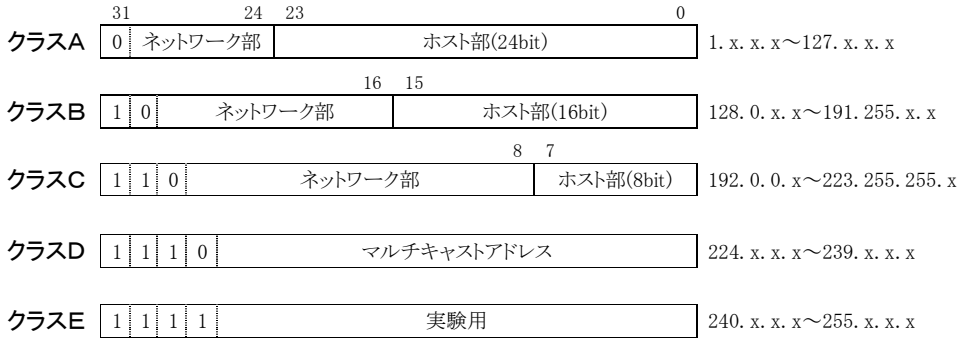
◆**ヘッダオプション** :ルーティングやセキュリティ等に関する付加情報を乗せるフィールドで、フィールド長はヘッダサイズ-20 となります。

●IP アドレスのフォーマット RFC796

IP アドレスは 32bit の数値で、192.168.1.8 などのように上位側からバイト単位でピリオドで区切った 10 進数で表記されます。IP アドレスはネットワークの規模や用途に応じて次のページに示すようにいくつかのクラスに分類されます。

クラスAアドレスは、ホスト数約 1677 万の巨大ネットワークに、クラスCアドレスはホスト数 254(各フィールドで全て 0 または 1 のアドレスは使わない)までの小規模ネットワークに与えられます。クラス B はその中間です。IP アドレスを取得する際は、1 社に一つのネットワークアドレスが割り当てられ(クラスC では例外もあるようだ)、ホストアドレスはそのネットワークの管理機構によってネットワーク内の各ホストに割り当てられます。

ＩＰアドレスのクラスとビットフィールド



クラスDアドレスは、特に特定のゲートウェイを指定するためのマルチキャストアドレスとして用いられています。また、クラスEアドレスは、実験用で一般には用いられません。

●ネットマスクとサブネットマスク RFC950

ネットマスクとは、IPアドレスからネットワークアドレスを抽出するためのビットマスクのことです。IPアドレスとネットマスクの論理和をとることにより、そのIPアドレスのネットワーク部が得られます。従ってネットマスクはIPアドレスのクラスによって決まります。たとえば、クラスCアドレスのネットマスク値は、255. 255. 255. 0 となります。

与えられた一つのネットワークアドレスを複数のネットワークに分けて使いたいということがあります。その場合、クラスで決まるネットワーク部を拡大して、ホスト部の一部をネットワーク部に取り込んで使用します。たとえば、クラスBアドレスのネットワーク部を 6bit 拡大すると、ホスト数 1022 のネットワークを 62 個作ることができるのです。ホスト部からネットワーク部に取り込まれた部分をサブネットアドレスといい、それも含んだネットマスクを特に**サブネットマスク**と言います。いま取り上げた例で言うと、サブネットマスクの値は、255. 255. 252. 0 となります。

●ネットワーク外との通信（ルータ越え）

ネットワーク外のホストへ IP パケットを送るにはどうすればよいのでしょうか。MACフレームは同一ネットワーク内にしか届かない(厳密には少し違いますが)ので、物理アドレスで宛先ホストの NIC を指定することはできません。そこで、送信時に宛先アドレスのネットワーク部と自分のネットワーク部とを比較(ここでネットマスクを使います)します。同じ場合は、ARP で相手の物理アドレスを調べて直接そこへ送りますが、違う場合は宛先がネットワーク外と判断して **IP パケットをルータの物理アドレスへ送る**のです。LAN 側から IP パケットを受信したルータは、宛先 IP アドレスが他のネットワークならそれを WAN 側に向かって送り出し、パケットはインターネットを巡り巡って宛先ホストに到達することになります。同様に、ネットワーク外からのパケットは WAN 側からルータに到着し、宛先がネットワーク内ならルータは LAN 側の宛先ホストに向けて転送します。

●特別な IP アドレス

宛先アドレスとして使用可能なもののうち、ホストに割り当てられるもののほかに特別な意味を持つアドレスがあります。よく使われそうな物をいくつか取り上げてみます。

255. 255. 255. 255	一般的な IP ブロードキャスト。ただし、そのネットワーク内のみ有効。
0. 0. 0. 0	このネットワークのこのホスト。つまり、自分自身。
ホスト部が全て 1	ネットワーク部で示すネットワーク内へのブロードキャスト。
ネットワーク部が全て 0	このネットワークのホスト部で示すホスト。

●ローカル用の IP アドレス RFC1918

次のネットワークアドレスは、プライベートインターネットで使えるアドレスとして予約しています。要はグローバル空間では使われることのないアドレスで、LAN などの閉鎖空間で使うことが推奨されているものです。

10. x. x. x～10. x. x. x	クラスA　1 個。
172. 16. x. x～172. 31. x. x	クラスB　16 個。
192. 168. 0. x～192. 168. 255. x	クラスC　256 個。

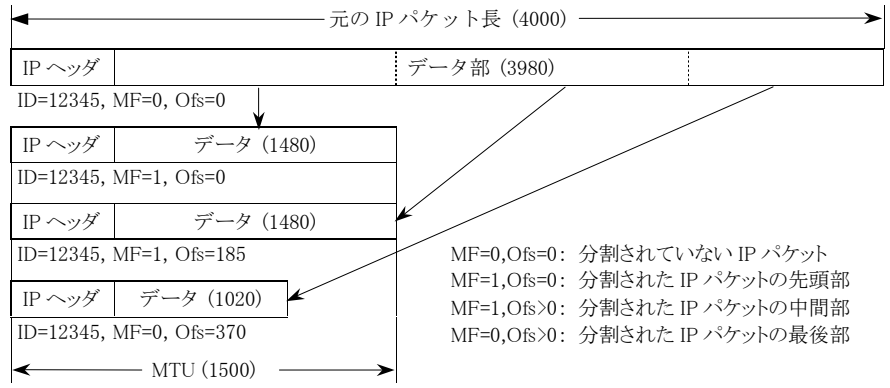
ところで、192. 168. . . . と聞いてピンとくる方が多

いと思います。ルータの持つ DHCP サーバのデフォルト設定では、この規約に則って LAN 側にアドレス割り当てるようになっているからです。

下の図に 4000byte の IP パケットを Ethernet で伝送しようとするときの例を示します。Ethernet の MTU は 1500byte なので、この IP パケットは 3 分割されて伝送されます。

●IP フラグメント の例

IP フラグメント の例



Column

チェックサム

IP、UDP、ICMP、TCP パケットはそれぞれエラー検出のためのチェックサムを持っています。計算方法は、チェックサム適用範囲のデータをワード単位で加算して、その結果の1 の補数をサムの値とします。サム作成の際、サムフィールドの値はゼロとして計算します。したがって、エラーチェックするときは、適用範囲に対してサムも含めて加算して最後に 1 の補数をとった結果がゼロならエラー無しということになります。

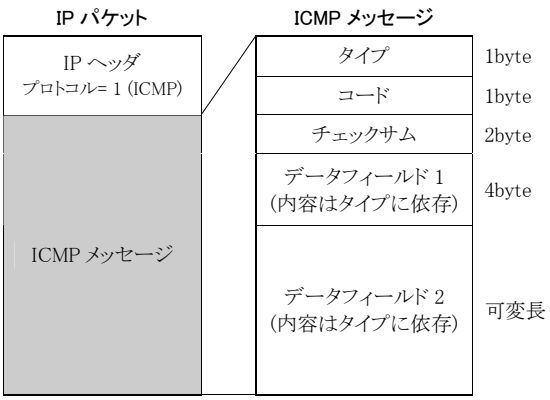
サムワード計算の際のキャリーは無視しないで、ワードの最下位ビットに戻します(つまり、キャリーが出たらさらに+1 する)。チェックサムの適用範囲が奇数バイトの場合は、最終バイトをワードの上位側に詰めて(例えば、0x34 ならば 0x3400 で)加算します。

ICMP (Internet Control Message Protocol) RFC792

ICMP は主に、IP パケットの伝送に何らかの問題が発生した場合、そのことを IP パケットの送り元に知らせるためのプロトコルです。つまり、ICMP はネットワークのトラブル解決を助ける IP サポートプロトコルとしての役割を持ちます。

◆**タイプ**:その ICMP メッセージのメッセージタイプを示します。
◆**コード**:タイプを補足するフィールドで、その ICMP メッセージの発生原因を示します。
◆**チェックサム**：メッセージ全体に適用。計算方法は IP ヘッダのサムと同じ。

●ICMP パケット のフォーマット



●主なICMPメッセージ RFC792 and follows

タイプ	コード	概要
Destination Unreachable (3)	原因	様々な理由でのメッセージ不達による IP パケットの破棄。コードはその原因を示す。
Time Exceeded (11)	原因	TTL がゼロになったり、デフラグメントタイムアウト等、時間切れによる IP パケットの破棄。コードはその原因を示す。
Parameter Problem (12)	0	ヘッダパラメータ異常による IP パケットの破棄。
Source Quench (4)	0	トラフィック低減要求。ゲートウェイの先の経路の速度不足等によるバッファオーバーフローによる IP パケットの破棄。
Redirect (5)	原因	指定した別のゲートウェイを使うように促す。IP パケットは転送される。
Echo (8) Echo Reply (0)	0 0	エコー要求/応答。Echo メッセージを受信したホストは、送信元に Echo Reply メッセージを返す。主に ping コマンドで使われる。

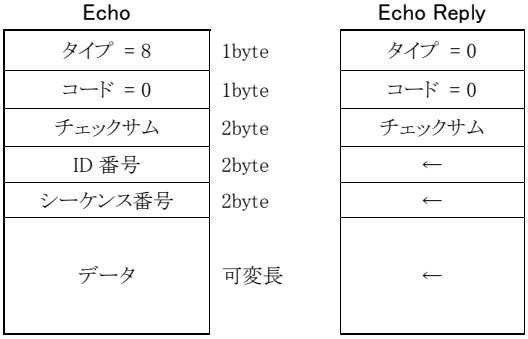
※IP パケットを破棄した場合でも、ICMP メッセージを発生するかどうかはそのゲートウェイ(または宛先ホスト)のインプリメント次第です。
また、フラグメントされた IP パケットの先頭部以外や、ICMP メッセージのエラーについては新たな ICMP メッセージを発生しません。

●ping における Echo/Echo Reply メッセージ

ホスト間で IP パケットが正常に伝送できるかどうか調べるコマンドとして有名なものに、ping があります。ping コマンドは、宛先ホストに ICMP メッセージの Echo を送信し、戻ってくる Echo Reply メッセージを調べます。

Echo メッセージを受信したホストは、ID 番号、シーケンス番号、データについては受信したものをそのまま返します(だから Echo メッセージと言うのでしょうけど)。

Echo メッセージではメッセージの内容自体に意味はありませんが、ID 番号とシーケンス番号はどの Echo メッセージに対する Echo Reply かを識別するため、ping を打つ側でメッセージ毎にユニークな値が付けられます。



●ルート 検索

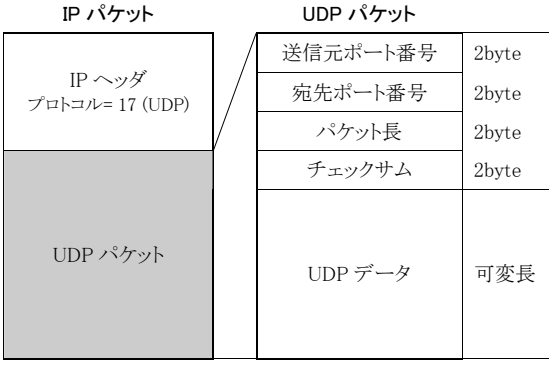
IP パケットがどんなルートを通って目的のホストに到達しているのかを調べる traceroute コマンドがあります。これは、TTL タイムアウトを利用しています。まず、ping と同様に目的ホストに向けて Echo メッセージを発信しますが、それを IP ヘッダの TTL 値を 1 から順に増やしながら繰り返していきます。TTL 値が 3 なら、3 番目、10 なら 10 番目のルータで IP パケットが破棄され、そのルータから Time Exceeded メッセージが返ります。これにより通過するルータを順に調べていけます。

UDP (User Datagram Protocol) RFC768

IP は一般のアプリケーションからは直接制御しないのが普通です。そこでアプリケーションから IP パケットをほぼ直接扱えるようにするためのプロトコルが UDP です。通信開始・終了の際には通信相手との接続・切断処理は必要なく、相手が受信待機状態ならいきなりパケットを送信することができます。これをほかのことに例えると、対象ホストの捕手(アプリケーション)のグローブ(開いている UDP ポート番号)に向かってボール(UDP パケット)を投げるだけといったイメージです。

UDP の信頼性は IP そのものなので、パケット損失

●UDP パケット のフォーマット



や到着順序の狂いなどのエラー制御はUDPを使用するアプリケーション側で考慮しなければなりません。しかし、1往復で済むような通信ではTCPに比べて効率が良いので、この利点を生かせるアプリケーション(BOOTPやDNSなど)で使われています。コネクションの概念がないので、IPブロードキャストを使って一発で1:Nの通信を行うことも可能です。

また、UDPは実装に必要なリソース(メモリ・CPUパワー等)がTCPより少なくて済むので、単純な組み込み用途やブート時の一時的用途でもよく使われるようです。

◆送信元ポート番号、宛先ポート番号：送り元と宛先のUDPポート番号。ポート番号とは、個々のパケットとアプリケーションとの関連付けのための識別番号です。送信元ポート番号にはそのパケットを送ったアプリケーションが使用したポート番号を入れます。相手のホストに到達したパケットは、宛先ポート番号の示すUDPポートに到着します。もしもそのポートを開いているアプリケーションがなければそのUDPパケットは破棄されます。

◆パケット長：このUDPパケットのbyte単位のサイズ(ヘッダ(8)＋データ(可変長))。

◆チェックサム：計算方法はIPヘッダのサムと同じですが、**疑似IPヘッダ**、UDPヘッダ、UDPデータに対して適用されるので、データ部のエラーも検出できます。また、UDPのチェックサムはオプションなので、省略することもできます。チェックサムを省略する場合、このフィールドに0x0000を設定します。受信側ではチェックサムの値が0x0000の場合、計算せずにエラー無しとします。有効な値を設定する場合、サムが0x0000になったら代わりに0xffffを設定します(計算上の問題はない)。

疑似IPヘッダとは実際に存在するフィールドではなく、サム計算に含めるための次のようなIPヘッダのサブセットのような構造体です。

疑似ヘッダ

送信元IPアドレス	4byte
宛先IPアドレス	4byte
0	1byte
プロトコル（UDP:17, TCP:6）	1byte
パケット長（ヘッダ+データ）	2byte

これにより誤配信パケットも確実に排除できるようになります。また、このことからUDPやTCPはIP上で使うことが前提で、それらは不可分の存在であることが分かります（だからTCP/IPなどと言うのですけど）。

●Well Known ポート について RFC1700

多くのサーバアプリケーションは、**特定のポート**をパッシブオープン(LISTEN)してクライアントからのサービス要求を待っています。たとえば、HTTPサーバならTCPポート番号80、telnetサーバならTCPポート番号23、DNSサーバならUDPポート番号53といった感じです。それらのポート番号を**Well Knownポート**といい、サーバアプリケーションを指定するための公的な窓口となっています。クライアントアプリケーションは、デフォルトではそれらのポートに接続します。たとえば、wwwブラウザは対象ホストのTCPポート番号80に対して接続して、HTTP手順でデータを引き出します。接続するポート番号は、http://www.who.com:8000/などとオーバーライドすることもできます(webサーバがそのポートで稼働していなければrejectされるだけですが)。

ソケット

送信元または宛先それぞれのIPアドレスとポート番号の組み合わせをソケットと言い、「IPアドレス：ポート番号」のように表記されます。

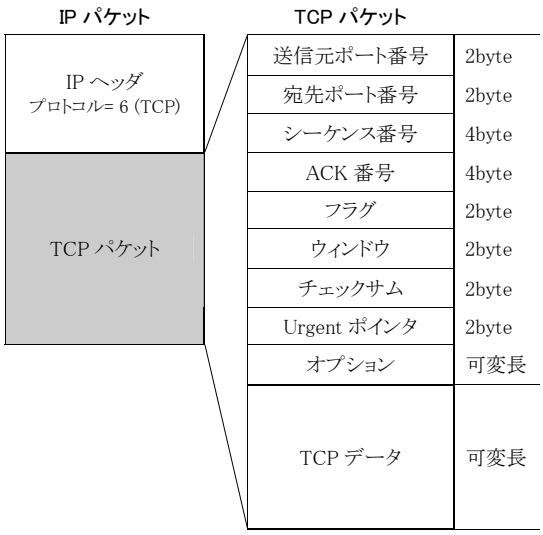
例 → 64.33.57.12:1050

TCP（Transmission Control Protocol）RFC793

TCPもUDP同様に、IPの上位に来るトランスポート層プロトコルです。UDPとの大きな違いは、コネクション型・ストリーミング型であるということです。ストリーミング型とは、UDPのようなパケットの概念のない(区切りが無く連続した)データストリームを扱えることを示します。つまり、TCPは通信相手との間に品質の保証された論理的な通信回線(コネクションという)を提供します。コネクション型は通信の開始と終了時に「接続」「切断」といった手続きを踏みます。TCPの場合、コネクションの開設(open)・切断(close)の処理がこれに相当します。

TCP上のアプリケーションは、通信エラー対策やデータのブロッキングといった処理から解放され、本来の処理に専念できます。このため、TCPはインターネット(イントラネット)を介してアプリケーション同士を結ぶ標準プロトコルといった存在になっています。

●TCP パケット のフォーマット



※TCP パケットのことを特に TCP セグメントと言います。それぞれの TCP パケットは互いに独立したものではなく、連続したデータストリームの一部であるからです。

◆送信元・宛先ポート番号：UDP におけるポート番号に同じですが、TCP ではコネクションの識別のために両端の IP アドレスとポート番号（ソケット）の組が重要な意味を持っています（後述）。

◆シーケンス番号：TCP では転送されるデータバイトそれぞれと一部制御ビット(SYN, FIN)に 32bit のシーケンス番号をふることによってセグメント欠損の検出や順序制御を実現しています。この値は、データ部の先頭バイトまたは、SYN, FIN ビットに割り当てられたシーケンス番号を示します。

例えば、ある TCP セグメントのシーケンス番号が 100 で、データ部が 100byte なら、その TCP データはそれぞれ 100～199 のシーケンス番号を持つことになります。当然ですが、同一セグメント中の SYN、FIN、データはそれぞれ排他です。SYN も FIN もデータも含まれないセグメントではシーケンス番号は特に意味を持ちません。

シーケンス番号は、2**32-1 を超えたら 0 にラップアラウンドするので、前後関係の比較には注意が必要です。

◆ACK 番号：ACK ビットが立っているとき有効なフィールド。相手のシーケンス番号に対応し、この直前のシーケンス番号までのデータは受け取っていることを示します。つまり、次に送ってほしいデータのシーケンス番号を示します。

◆フラグ：

データオフセット (4bit)	未使用(0) (6bit)	URG	ACK	PSH	RST	SYN	FIN

フラグは 16bit のフィールドで、その TCP セグメントの意味を示す重要なものです。

データオフセット：データ部の開始位置、つまり、この TCP ヘッダのサイズ(20～64byte)を 4byte 単位で示します。

URG (Urgent)：緊急に処理してほしいデータがあり、Urgent ポインタが有効なことを示しています。

ACK（Acknowledgment）：ACK 番号フィールドが有効なことを示しています。相手からの SYN を受け取った後は常に立っています。

PSH (Push)：転送データの区切り(上位から write ファンクションで一度に渡されたデータの最後)など、ここまでのデータをバッファリングしたままにせず、すぐに相手の上位レイヤに渡してほしい場合にセットします。

RST（Reset）：致命的エラー発生時や緊急時などに強制的にコネクションを切断したい場合に使用します。このビットの立ったセグメントを受信した TCB は即時破棄されコネクションは切断されます。

SYN（Synchronization）：コネクション開設時に使われるビットで、こちらのシーケンス番号の初期値（0とは限らない）を相手に知らせることを示します。SYN ビットはそれに対する ACK を必要とするので、シーケンス番号を持ちます。したがって、最初のデータは、SYN のシーケンス番号＋1 から始まります。双方が自分の送った SYN に対する ACK を確認した時点でコネクションが確立します。

FIN（Final）：コネクション切断時に使われるビットで、こちらからのデータ送信が終了し、これからコネクションを切断したいことを相手に知らせるビットです。このビットもそれに対する ACK を必要とするので、シーケンス番号を持ちます。

コネクション開設から切断までの一連のシーケンス番号の最初が SYN で、中間がデータ、最後が FIN になると言えます。

◆ウィンドウ：自分のデータ受信バッファの空き（受信ウィンドウ）サイズを示すフィールド。つまり、こちらからの ACK の確認なしに一方的に送ってよいデータバイト数を相手に示します。受信バッファ分のデータを送り出す時間がホスト間のラウンドトリップ（IP パケットの往復）時間より長く、受信側が滞りなくデータを引き取れば、伝送路の帯域を目いっぱい使って途切れることなくデータを転送できることになります。

また、送信ウィンドウ（ACK 待ちキューの空き）と

いうのもありますが、これは送信側とその上位レイヤとの間でローカルに完結するもので、プロトコル上には出てきません。

◆チェックサム：計算方法は UDP と同じ。UDP では省略可能でしたが、TCP ではサムは必須です。詳しくは UDP の解説を参照してください。

◆Urgent ポインタ ：URG ビットが立っているときに有効な値。緊急に処理してほしいデータをデータ部の先頭からこの値で示すバイト数割り込ませてあることを示す。緊急データに対して特別な処理をするわけではなく、上位アプリケーションにシグナルを送って緊急データがあることを知らせます(緊急データは通常のデータとは別チャネルのデータとして扱われます)。その手のデータを送るには、この機能を使うよりもう1 本コネクションを開設することが多いようですが。

◆オプション ：可変長フィールドで、サイズはヘッダサイズ-20 となります。RFC793 では、自分の MSS 値(処理可能最大セグメントサイズ)を相手に知らせるためのオプションが定義されています。オプションは SYN セグメントで使用可能です。

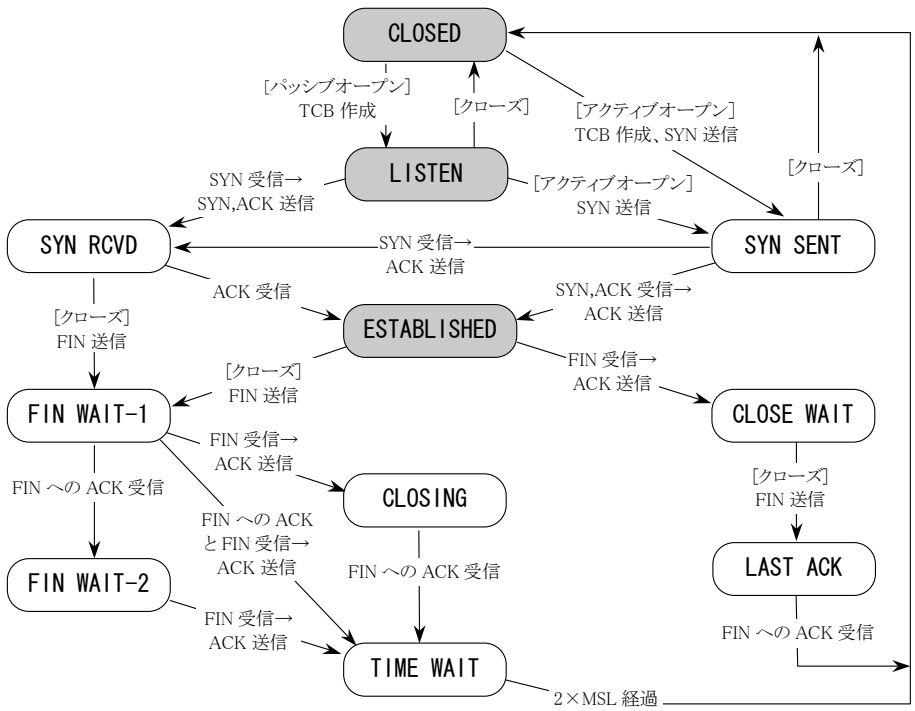
◆TCP データ ：可変長フィールド。伝送するデータです。このフィールドの先頭データがシーケンス番号の示す値のシーケンス番号を持ちます。

●TCP のステート ダイアグラム

TCP では、接続および切断の際に双方で確実なハンドシェークを行うため、それぞれの **TCB** に次の図のようにステートが定義されています。ステートダイアグラムは一見複雑そうですが、このうち定常状態における ステート は、 CLOSED 、 LISTEN 、 ESTABLISHED の 3 つだけで、それ以外は接続・切断時に通過する過渡的なステートです。

ここで、TCB (Transmission Control Block)とは、コネクション毎に作成されてその状態（ステート、シーケンス番号、各種バッファ等）を保持している構造体のことです。それぞれのコネクションは、コネクション両端のソケットの組で識別されます。どちらかが違えばそれらのコネクションは別物です。ソケットプログラミングをやっている方にとっては常識的なことなのでたやすく理解できると思いますが。これにより、サーバは一つのポートで複数のクライアントに同時にサービスできますし、http クライアントが web ページを開くときなど異なるポートで次々とコネクションを開設して複数のコンテンツを並行して転送することもできるのです。

TCP のステート ダイアグラム



◆**CLOSED** ：TCB が無い状態。それなので厳密な意味でのステートではありません。

◆**LISTEN** ：上位からの open() (パッシブオープン要求)などにより作成された TCB が接続要求を待っている状態。この時点の TCB は相手側のソケット情報が null で、コネクションとしてはまだ無効です。接続要求(SYN の立った TCP セグメント)が来たら、相手からの SYN に対する ACK と自分からの SYN を返して SYN RCVD に移行します。

相手側ソケットが確定したら有効なコネクションとして元の TCB から分離されます。以降、そのコネクションのセグメントだけがその TCB で処理されます。

◆**SYN RCVD** ：相手からの接続要求(SYN)に ACK を返したあと、同時に送った SYN に対する ACK を待っている状態。SYN に対する ACK を確認したら、ESTABLISHED に移行します。上位にはシグナル等で接続した旨知らせます。

◆**SYN SENT** ：上位からの open() (アクティブオープン要求)などにより TCB が作成され、接続先に SYN を送って相手からの SYN,ACK を待っている状態。相手からの SYN,ACK を確認したら、SYN に対する ACK を返して ESTABLISHED に移行します。上位にはシグナル等で接続した旨知らせます。

◆**ESTABLISHED** ：コネクションが確立している状態。上位側からの read()や write()でデータの送受信ができます。こちらから切断を要求する場合は、FIN を送信して FIN WAIT-1 に移行します。相手から先に FIN が来たらそれに対する ACK を返して CLOSE WAIT に移行します。

◆**FIN WAIT-1** ：上位からの close()などでこの状態に移行します。こちらからコネクションの切断を要求するするため、FIN を送信して相手の応答を待っている状態。相手からのデータはまだ受信できます。FIN に対する ACK が来たら FIN WAIT-2 に移行します。相手からの FIN が来たらそれに対する ACK を返して CLOSING に移行します。両方が同じセグメントで来たら ACK を返して直接 TIME WAIT に移行します。

◆**FIN WAIT-2** ：こちらからの切断シーケンスは済み、相手からの切断要求を待っているハーフクローズ状態。相手からのデータはまだ受信できます。相手からの FIN が来たらそれに対する ACK を返して TIME WIAT に移行します。

◆**CLOSING** ：こちらから FIN を送った後、相手からも FIN が来ている状態。FIN に対する ACK が来たら、それに対する ACK を返して TIME WAIT に移行します。

◆**TIME WAIT** ：コネクションの切断が完了した状態。そのコネクションの TCP セグメントがまだネット上を迷走している可能性があるため、その間同じコネクションが使われないように TCB を凍結しておく期間。通常、セグメント寿命×2 (60 秒程度)ですが、最大コネクション数の乏しい環境では TCB 不足を防ぐためもっと短い時間で済ませることもあるようです。待ち時間が過ぎたら、その TCB を破棄します。

◆**CLOSE WIAT** ：相手からの切断シーケンスは済んだが、こちらからはまだ FIN を出していないハーフクローズ状態。上位にはシグナル等で相手からの切断要求を知らせます。こちらからの送信はできます。切断準備ができたら(上位が close()するなど)、FIN を送って LAST ACK に移行します。

◆**LAST ACK** ：相手からの切断シーケンスが済み、こちらからの FIN に対する ACK を待っている状態。FIN に対する ACK が来たらコネクションの切断は完了、TCB を破棄します。

※ホストに到着した TCP パケットは、各 TCB に配信されて処理されますが、それに対して具体的にどのような手順で処理をするべきか RFC793 の最後の方でその例が詳しく解説されています。これは TCP/IP のインプリメントの際にはかなり参考になるので必読です。

●TCP の動作例（接続からデータ転送まで）

次のページの図に接続時のシーケンスを示します。図から TCP 接続時の**3 ウェイハンドシェーク**の様子がよく分かるでしょう。

①TCP A が SYN ビットを立てて TCP B に能動的に接続要求を出します。シーケンス番号フィールドには自分のシーケンス番号の初期値(この例では 100)を入れます。

②TCP A からの SYN(接続要求)を受け、これを受理した TCP B は、ACK=101 を返して SYN(100)を受け取ったことを TCP A に示します。同時に SYN も立てて自分のシーケンス番号の初期値(300)を TCP B に伝えます。

③TCP A は TCP B からの ACK を受け、接続が確立します。同時に ACK=301 を返して SYN(300)を受け取ったことを TCP B に示します。TCP B も TCP A からの ACK を受け取ったら接続が確立します。

④コネクションが確立するとデータを転送できるようになります。

TCP A は、上位から送信データ(この例では 100byte)が渡されると、シーケンス番号 101～200 のデータを入れたセグメントを TCP B へ送信します。相手の

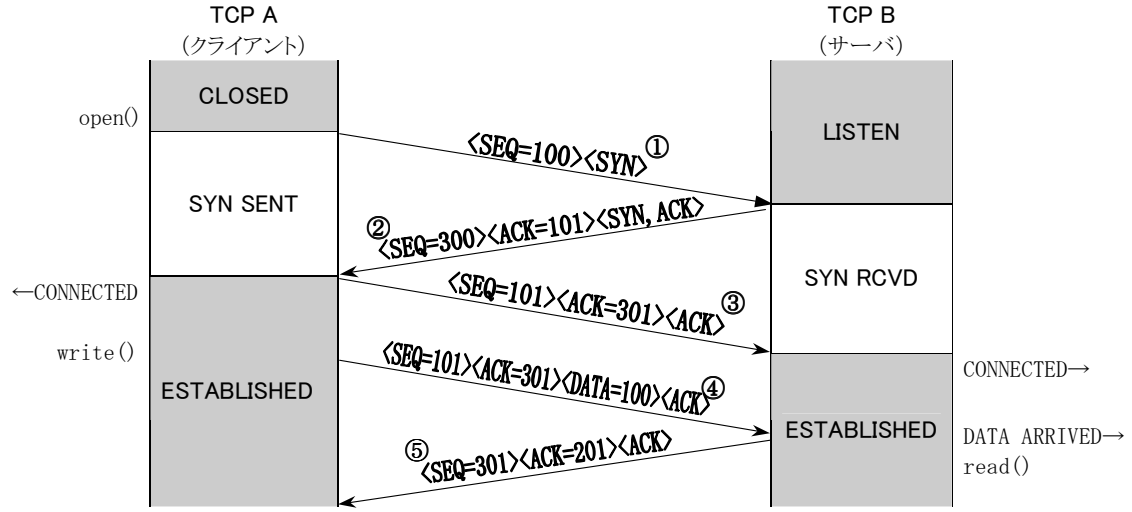
MSS を越えるサイズの送信データがある場合は、当然複数のセグメントに分割されて送られます。

シーケンス番号が③と同じなのは、③にはシーケンス番号を持つもの(データや SYN, FIN ビット)が含まれないからです。

⑤TCP B は TCP A からのデータを引き取ると ACK=201 で応答して、シーケンス番号 200 までのデータの受信が完了して 201 からのデータを要求していることを TCP A に示します。

※TCP は双方が対称であるため、一方から開始される接続のほか、両方から同時に接続を開始するという動作も考慮されています。そのため、SYN SENT から SYN RCVD へのパスが定義されています。

一般的な接続シーケンスとデータ転送



※この例ではサーバが接続要求を待っている一般的な例だが、クライアントとサーバの関係はそう決まっているわけではない。

●実際の TCP 接続時のパケット ダンプ (http サーバへの接続例)

```
Packet.1 ①
0000  00 00 02 00  58 A7 00 90  99 42 1D 11  08 00 45 00  ....X....B....E.  [TCP/IP]
0010  00 30 95 18  40 00 80 06  E2 55 C0 A8  01 03 C0 A8  .0..@....U......  SRC=192.168.1.3:1030, DST=192.168.1.6:80
0020  01 06 04 06  00 50 00 2D  61 73 00 00  00 00 70 02  .....P.-as....p.  SEQ=2974067, ACK=0, CTL=SYN, WIN=8192
0030  20 00 79 CF  00 00 02 04  05 B4 01 01  04 02        .y...........     Option: MSS=1460

Packet.2 ②
0000  00 90 99 42  1D 11 00 00  02 00 58 A7  08 00 45 00  ...B......X...E.  [TCP/IP]
0010  00 30 32 00  40 00 80 06  45 6E C0 A8  01 06 C0 A8  .02.@...En......  SRC=192.168.1.6:80, DST=192.168.1.3:1030
0020  01 03 00 50  04 06 00 05  53 7E 00 2D  61 74 70 12  ...P....S~.-atp.  SEQ=349054, ACK=2974068, CTL=SYN,ACK, WIN=8760
0030  22 38 24 03  00 00 02 04  05 B4 01 01  04 02        "8$...........    Option: MSS=1460

Packet.3 ③
0000  00 00 02 00  58 A7 00 90  99 42 1D 11  08 00 45 00  ....X....B....E.  [TCP/IP]
0010  00 28 96 18  40 00 80 06  E1 5D C0 A8  01 03 C0 A8  .(..@....]......  SRC=192.168.1.3:1030, DST=192.168.1.6:80
0020  01 06 04 06  00 50 00 2D  61 74 00 05  53 7F 50 10  .....P.-at..S.P.  SEQ=2974068, ACK=349055, CTL=ACK, WIN=8760
0030  22 38 50 C7  00 00 20 20  20 20 20 20              "8P... ....       ※下線部は MAC パッド(ごみ)

Packet.4 ④
0000  00 00 02 00  58 A7 00 90  99 42 1D 11  08 00 45 00  ....X....B....E.  [TCP/IP]
0010  01 1D 97 18  40 00 80 06  DF 68 C0 A8  01 03 C0 A8  ....@....h......  SRC=192.168.1.3:1030, DST=192.168.1.6:80
0020  01 06 04 06  00 50 00 2D  61 74 00 05  53 7F 50 18  .....P.-at..S.P.  SEQ=2974068, ACK=349055, CTL=ACK, WIN=8760
0030  22 38 B0 B3  00 00 47 45  54 20 2F 69  6E 64 65 78  "8....GET /index  DATA=245bytes "GET /index.html HTTP/1.0.....
0040  2E 68 74 6D  6C 20 48 54  54 50 2F 31  2E 30 0D 0A  .html HTTP/1.0..
                        ---- 中略 ----
0110  72 73 65 74  3A 20 53 68  69 66 74 5F  4A 49 53 2C  rset: Shift_JIS,
0120  2A 2C 75 74  66 2D 38 0D  0A 0D 0A                 *,utf-8....       ※枠内は MAC/IP/TCP 各ヘッダ
```

●TCP の動作例（一般的なクローズ）

①TCP A からコネクションをクローズする場合を示します。上位からクローズ要求があると、TCP A は FIN ビットを立てたセグメントを TCP B に送信します。

②TCP B は、TCP A からの FIN を受けるとそれに対して ACK を返して、さらに上位にシグナル等で TCP A からの送信が終了したことを知らせます。この時点ではまだ TCP B からデータを送ることができます（ハーフクローズ）。上位は送るデータが無ければ、TCP にクローズ要求を出します。

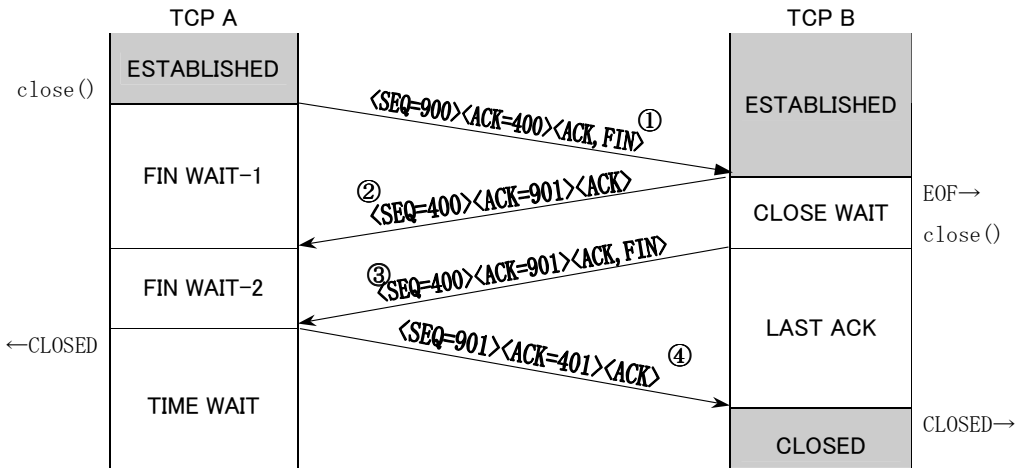
③TCP B は、上位からのクローズ要求が来たら TCP A に FIN を送ります。TCP B がクローズ要求を即時受け入れ可能な状態なら、②と③を同じセグメントで返すこともあります。

④双方のクローズシーケンスが済むとコネクションのクローズが完了しますが、最初に FIN を出した方は一定時間 TCB を保持しておく必要があります。

※TCP は双方が対称であるため、一方から開始される切断のほか、両方から同時に切断を開始するという動作も考慮されています。そのため、CLOSING ステートが定義されています。

※通常の手順で切断した場合は一定時間 TCB を保持しておく必要がありますが、リソースの乏しい環境では、短時間に接続・切断を繰り返すとリソース不足で新たな TCB が作成できない事態が発生します。このような場合は、TIME WAIT を介さないリセットによる切断(ハードクローズと言う)の方が良いでしょう。ただし、ハードクローズは相手の同意のない強制的な切断なので相手から送信中のデータが失われる場合があります。

一般的な切断シーケンス



●TCP の動作例（エラーリカバリー）

LAN のような近距離ならセグメントが失われたり到着順が狂ったりデータが化けたりすることは殆ど無いのですが、インターネットではそういったことがよく起きます。TCP ではタイムアウト再送やシーケンス番号の確認でこれらのエラーを回避して信頼性の高いコネクションを提供しています。

◆送信側 TCP は、送信したセグメントを ACK 待ちキューに一時保存して、同時に再送タイマをスタートします。

◆ACK が帰ってきたら、その ACK 番号直前までのシーケンス番号のデータを持つセグメントをキューから削除します。

◆タイムアウトしたセグメントは再送されます。再送間隔は一定ではなく、1, 2, 4, 8...といった指数関数的に間隔を広げていくようにします。このようにシーケンス番号を持つもの(データと SYN、FIN)は再送によって到達が保証されます。つまり、伝送制御(到達確認と再送処理)はそのデータの送信側の主導の下に行われます。受信側が能動的にデータを要求するということはありません。

◆受信側 TCP では、現在の ACK 番号以外のセグメントが到着しても受け付けず（セグメントが ACK 位置にまたがっている場合、その中の新しい分だけ引き取る）、現在の ACK 番号を返します。これによりデータストリームの順序が保証されます。

●TCP の動作例（フロー制御）

TCP セグメントに含まれるウィンドウサイズは、その TCP の受信バッファの空きを示しています。正確には、その ACK 番号から数えた受信可能なデータバイト数を示します。たとえば、〈ACK=300〉〈Win=200〉なら、シーケンス番号 499 までは一方的に送ってよいという意味になります。

図ではデータを 300byte 送った時点でウィンドウが閉じてそれ以上送れない状態になります。その後、TCP B の上位がデータを読み出してバッファに新たな空きができた時点で、改めて〈Win=200〉を TCP A に送っています。これを受けて TCP A は続きのデータを送り始めます。

ところで、ウィンドウを開くセグメントが通信エラーで失われたらどうなるでしょうか。ウィンドウは永久に閉じたままになってデータ転送が止まってしまいます。このため、送信側はウィンドウが閉じたまま一定時間(2 分とされています)経過したら続きのデータを送ってみることになっています。まだウィンドウが閉じていれば〈ACK=500〉〈Win=0〉が帰ってきますし、既に空いていて受信できるサイズなら受信されて次の ACK 番号とウィンドウ値が帰ってきます。

※ここで示した以外にもいろいろなエラーリカバリーの例が RFC793 では解説されています。TCP の動作をより詳しく理解するには一度は目を通しておいてください。

●Silly Window Syndrome（SWS） RFC813

ウィンドウ制御で陥りやすい良くない状況を SWS（おバカなウィンドウ制御？）といいます。典型的な例は、データが無用に細かいセグメントに分割されるという状態です。たとえば、極端な例ですが、受信バッファがいっぱいの状態で上位が 1byte づつデータを引き取るような場合です。すると、そのたびに、

① 〈Win=1〉が送信側に返る
② 送信側が〈Data:1〉を送る
③ 〈Win=0〉が送信側に返る

と、1 バイト転送当たり 3 つの IP パケットが流れることになります。そのような状態はネットワークの利用効率を悪くするので避けなければなりません。

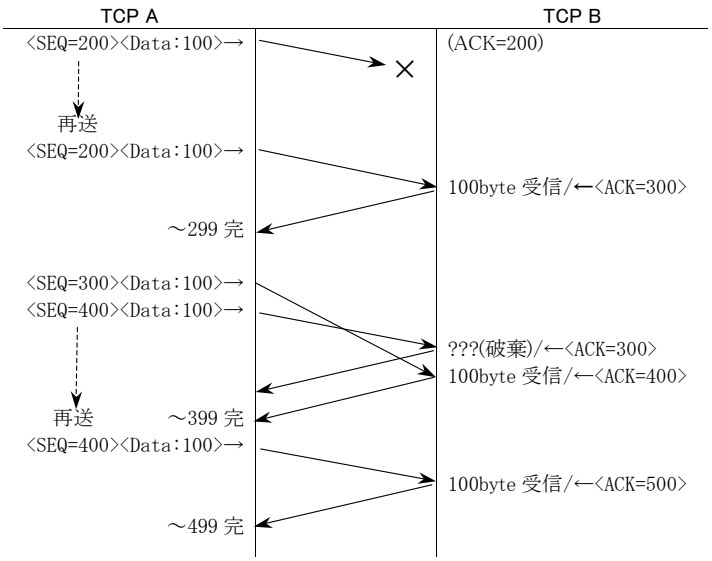
SWS を避けるには上位側がデータをまとめて引き取るようにすればよいのですが、これは TCP で解決すべき問題です。TCP での対策は次のような方法が推奨されています。

◆受信側では、受信バッファが少し空いてもすぐにウィンドウ値を返さず、空きが 50%以上になるか数 100ms 経過してからウィンドウ値を返すようにする。
◆送信側では、一旦ウィンドウが閉じたら初期値の 25%以上に開くまで送信を控える。

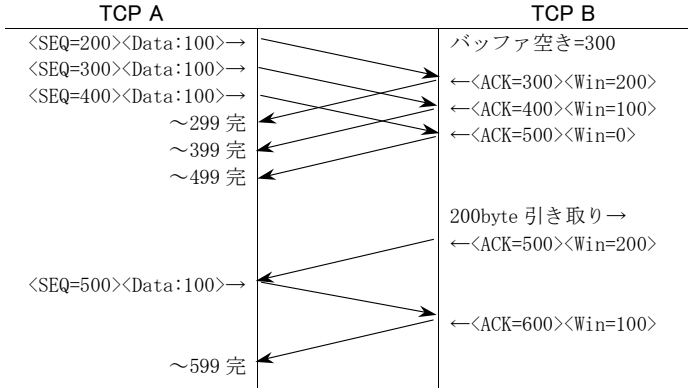
●キープアライブ（Keep-alive）

いわゆる生存確認です。コネクションが確立したあと、相手のクラッシュや経路の障害等で片割れのコ

エラー制御（パケット抜けや順序制御）の例



フロー制御の例



ネクションになる場合があります。相手からセグメントが届かない状態が長く続いたとしても、相手が死んだのかそれとも単にデータ転送が無いだけなのか分からないので、放っておいても自然には復旧しません。

そこで、セグメントが流れない状態が一定時間続いたら、キープアライブパケットという不正なシーケンス番号のセグメントを送ってみます。相手が生きていれば正しい ACK 番号の応答があるはずです。応答のない状態が続いたら相手が死んでいると判断してコネクションをリセットします。

キープアライブは、元々TCP の規格には無かったものですが、現在は多くのインプリメントにおいて標準的に備えられています。

DHCP (Dynamic Host Configuration Protocol)

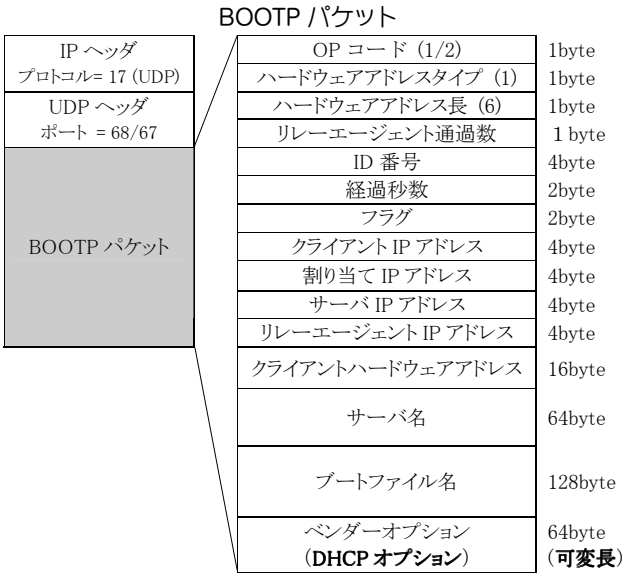
RFC2131

DHCP は、ネットワーク上での動作に必要な情報（IP アドレス、ネットマスク、ゲートウェイ、その他多数）を各ホスト与えるためのプロトコルです。動作の基本は、これからネットワークに参加するホストが DHCP サーバにサービス要求を出して、サーバがそれに応じて IP アドレス等のコンフィグレーションパラメータをホストに割り当てるというものです。DHCP サーバはその目的上常に稼働していなければなりません。このため、ルータなどにインプリメントされることが多いようです。

DHCP は、その前身の **BOOTP**（Bootstrap Protocol）**を拡張**する形で実現しているので、まずは BOOTP から説明しなければなりません。**BOOTP は、UDP 上で動作するプロトコル**です。**サーバは、UDP ポート番号 67** をオープンして待機しています。**クライアントは、UDP ポート番号 68** を使用してサーバと通信します。

●BOOTP パケット　RFC1542

右の図に BOOTP メッセージのパケットを示します。



◆OP コード　：この BOOTP パケット（または DHCP メッセージ）の種類を示します。
BOOTREQUEST(1)：クライアントからサーバへの IP アドレス割り当て要求。※DHCP では、クライアントの発するメッセージ。
BOOTREPLY(2)：サーバからクライアントへの IP アドレス割り当て応答。※DHCP では、サーバの発するメッセージ。
◆ハードウェアアドレスタイプ　：Ethernet では 1 を使用します。
◆ハードウェアアドレス長　：Ethernet では 6 を使用します。
◆リレーエージェント通過数　：リレーエージェントが使うフィールドです。メッセージ生成時に 0 で初期化します。リレーエージェントを通過する度に+1 され、16 に達したらそのパケットは捨てられます。

リレーエージェントとは、BOOTP サーバがルータ越えの場所にあるとき、BOOTP パケットを中継するために設けられるホストのこと。IP ブロードキャストを使う BOOTP パケットは通常の IP ルーティングができないためです。
◆トランザクション ID　：それぞれの BOOTP リクエスト/リプライを識別する番号。サーバは応答するパケットの ID 値でリプライします。
◆経過秒数　：クライアントがブート開始してからの経過秒数。タイムアウト等でリトライするときは、トランザクション開始からの経過秒数をセットする。※定義が不明確。
◆フラグ　：BROADCAST フラグ(MSB)のみが使用さ

れます。通常、サーバは与えるIPアドレスを宛先にしてリプライを返しますが、クライアントがインプリメント上、自分のIPアドレスが確定するまでユニキャストIPパケットを受信できない場合、サーバにブロードキャストでリプライを返すように要求するフラグ。

◆クライアントIPアドレス：クライアントが以前割り当てられたIPアドレスを覚えているとき、これを優先的に割り当ててもらうため、サーバに知らせるもの。※定義が不明確のため、BOOTPでは0.0.0.0をセットすべきとなっています。

◆割り当てIPアドレス：クライアントは、0.0.0.0をセット。サーバは割り当てるIPアドレスをここにセットして割り当てるクライアントにリプライを返します。DHCPにおいてもこのフィールドが割り当てIPアドレスの格納フィールドとして使われます。

◆サーバIPアドレス：リレーエージェントが使うフィールド。クライアントは0.0.0.0をセット、サーバからの戻り値は無意味。

◆リレーエージェントIPアドレス：リレーエージェントが使うフィールド。クライアントは、0.0.0.0をセットすること。

◆クライアントハードウェアアドレス：クライアントが自分の物理アドレスをセット。残りの10バイトは0でパディングします。このフィールドはサーバが個々のクライアントを識別するために使用します。

◆サーバ名：クライアントは、0をセット。BOOTPサーバがサーバ名を返し、残りは0でパディング。

◆ブートファイル名：クライアントは、0をセット。BOOTPサーバがブートファイル名を返し、残りは0でパディング。

◆ベンダーオプション：ベンダ依存のオプションを送るためのフィールド。"magic cookie"という4バイトのベンダ識別コードと、それに続くオプション形式のパラメータとなります。残りは0でパディング。オプションを使わないときはmagic cookieの値を0x63825363として、オプションは無し(Endコードのみ)とします。

BOOTPでは実際には殆ど使われていなかったので、**DHCPではこの部分を拡張して主要なパラメータを記述**するようになりました。さらにフィールドサイズは64バイト以上の可変長へと変更されています。

●BOOTPの動作

BOOTPの動作はとても単純で、1往復の通信でコンフィグレーション動作が完了します。

①BOOTPクライアントがリクエストメッセージをブロードキャストしてサービスを要求します。

宛先ソケットは、255.255.255.255:67としてブロードキャストします。送信元ソケットはIPアドレスがまだ確定していないので、0.0.0.0:68とします。BOOTPパケットは所定の値で埋めます。

②BOOTPサーバは、リプライメッセージでIPアドレスをクライアントに知らせます。

それぞれのクライアントは物理アドレスで識別され、それぞれにユニークなIPアドレスが割り当てられます。クライアントは、受信したリプライメッセージから割り当てIPアドレスを取り出して動作を開始します。

■DHCPにおける拡張点

BOOTPは、元々ディスク無しのX端末にIPアドレスを割り当てたりブートファイルを指定したりするためのプロトコルです。クライアントはIPアドレスしか得ることができませんし、その割り当ては永久的で、限られたIPアドレスを動的に管理することができません。

DHCPではIPアドレスの割り当て以外にもサブネットマスクやゲートウェイ、DNSサーバ、各種タイマー値などネットワークでの動作に必要なさまざまな情報も同時に得られるようになりました。

IPアドレスも期限付きの割り当てが可能になっています。このため、DHCPではメッセージのタイプが増え、クライアント側にはやや複雑なステートが定義されています。

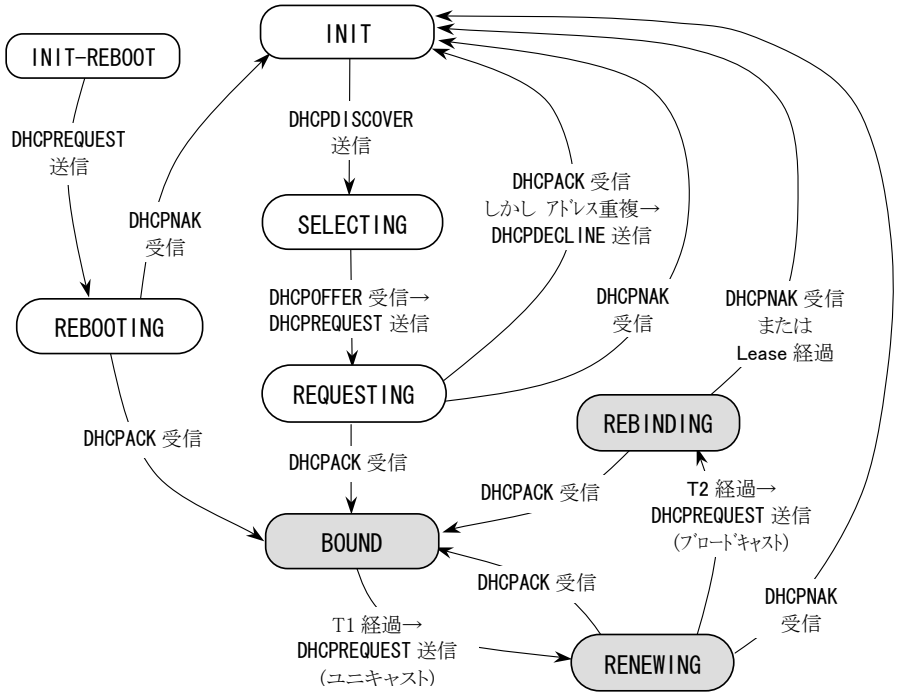
●DHCPクライアントのステート

クライアント側には次のページで示すように明確なステートが定義されています。

サーバへメッセージを送信するタイミングはステート遷移のときだけですが、サーバからの応答がない場合などは適宜リトライを行う必要もあります。

割り当てられたIPアドレスが有効なのは、灰色のステートの間だけで、それ以外の状態ではそのクライアントのIPアドレスは無効となり、DHCP以外のプロトコルを停止しなければなりません。

DHCP クライアント のステート ダイアグラム



◆**INIT**:これから IPアドレスを取得しようとしている初期状態。DHCP を開始するには SELECTING に移行します。

◆**SELECTING**：サービス可能な DHCP サーバを探すため DHCPDISCOVER をブロードキャストしてサーバからの DHCPOFFER (サービス提供の申し出) を待っている状態。サーバが複数ある場合は複数の応答があります。

一定時間待ってサーバからの申し出が集まったら、その中から適当な条件を示したサーバを選びます(普通は最初の応答か前回使ったサーバでしょうけど)。使うサーバを決めたら REQUESTING に移行します。

◆**REQUESTING**:DHCPREQUEST でサーバにアドレス割り当て要求を出して、返事を待っている状態。このときのメッセージには選んだサーバの ID と要求する IP アドレスを入れてブロードキャストします。選んだサーバの IDをメッセージに入れるのは、要求を出すサーバを特定するためです。同時に、割り当てを要求する IPアドレスもメッセージに含めなければなりません。

クライアントに IP アドレスが割り当てられると、サーバから DHCPACK が帰ります。DHCPACK を受信したら、与えられた IPアドレスの重複を ARP でチェックします。重複が無ければ BOUND へ移行してネットワークの動作を開始します。

アドレス重複チェックを行うのは、DHCP 管理外や、誤って IPアドレスが設定されているホストとの衝突を避けるためです。もしもその IPアドレスの重複を検出した場合は、DHCPDECLINE で重複の発生をサーバに知らせて INIT に戻ります。

サーバは DHCPOFFER の時点では空いているアドレスの一つをクライアントに示すのみで、その時点ではそのアドレスを予約するわけではありません。したがって、要求を出したときにはそのアドレスが空いているとは限りませんので、DHCPNAK で要求が拒否されるかも知れません。DHCPNAK を受信した場合は黙って INIT へ戻ります。

◆**BOUND**：IP アドレスが貸与された通常の動作状態。殆どの間はこの状態にあります。この間に受信した DHCP メッセージは全て破棄します。

このステートに移行するのは DHCPACK を受けたことによりますが、そのとき与えられる **Lease, T1, T2** で 3 つのタイマをスタートします。それぞれの時間の比は Lease > T2 > T1(一般的に 8:7:4)となります。Lease は、その IPアドレスが貸与される時間で、新た

な DHCPACK を受けないままタイムアウトするとその IP アドレスが無効になり、ネットワークを停止して INIT ステートに戻らなければなりません。T1 は貸与期間の更新を開始する時間で、タイムアウトしたら RENEWING に移行します。

◆**RENEWING**：貸与期間の更新のため、その IP アドレスを与えたサーバに DHCPREQUEST を**ユニキャスト**してサーバからの応答を待ちます。DHCPACK を受信したら再び BOUND に移行します。DHCPNAK を受信したら、ネットワークを停止して INIT に移行しなければなりません。

このステートと次の REBINDING ステートでは、誰かが処理を待っているといったような時間的制約がないので、リトライ間隔は最短 60 秒とされています。何も受信しないまま T2 がタイムアウトしたら、REBINDING に移行します。

◆**REBINDING**：現在の IP アドレスを与えたサーバからの応答が無いため、ブロードキャストで DHCPREQUEST を出して、現在の IP アドレスを扱える別のサーバに貸与期間の延長を求めている状態。DHCPACK を受信したらそのサーバ ID を記憶して再び BOUND に移行します。DHCPNAK を受信したり、何も受信しないまま Lease がタイムアウトしたら、ネットワークを停止して INIT に移行しなければなりません。

◆**INIT-REBOOT**：以前 IP アドレスを割り当てられていたクライアントがリブートした状態。以前の IP アドレスが分かる場合、それがまだ有効かどうかを確認するため、いきなり DHCPREQUEST を送ることができます。

◆**REBOOTING** ：DHCPREQUEST を送って、IP アドレスが有効かどうかサーバからの返事を待っている状態。DHCPNAK が帰ったり、何回かのリトライで応答がない場合、INIT に移行して最初からやり直します。それ以外の挙動は REQUESTING ステートと同じです。

●DHCP メッセージ

BOOTP のメッセージは、リクエストとリプライだけでしたが、DHCP ではクライアントとサーバがより綿密なハンドシェークを行うためメッセージの種類が大幅に増やされています。DHCP メッセージの種別は DHCP オプションの一つとしてオプションフィールドに格納されます。OP コードフィールドは BOOTP との互換性維持のためか、クライアントが発するときの BOOTREQUEST(1)とサーバが発するときの BOOTREPLY(2)だけとなっています。

◆**DHCPDISCOVER**：クライアントが DHCP サーバを探すため SELECTING ステートに移行する際にブロードキャストするメッセージ。

◆**DHCPOFFER**：DHCPDISCOVER を受けて、サービスの提供が可能なサーバはこのメッセージでクライアントにサービス条件（割り当て IP アドレスや貸与時間等）を提示します。

◆**DHCPREQUEST**：クライアントがサーバにサービスを要求するメッセージ。DHCPOFFER でサービス提供を申し出てきたサーバに正式にサービスを要求するとき、IP アドレス貸与期間の更新を要求するとき、またはクライアントがリブートした際に前の IP アドレスがまだ使えるか問い合わせるときに使用します。

◆**DHCPACK**：クライアントからの DHCPREQUEST に対して、IP アドレスを割り当てるメッセージ。クライアントに割り当てる IP アドレスと、その他のコンフィグレーションパラメータを含みます。

◆**DHCPNAK**：クライアントからの DHCPREQUEST を拒否するメッセージ。これを受けたクライアントはネットワークを停止して INIT ステートからやり直さなければなりません。

◆**DHCPRELEASE**：クライアントがシャットダウンするときなど、割り当てられている IP アドレスを解放することをサーバに知らせるメッセージ。

◆**DHCPDECLINE**：DHCPACK で割り当てられた IP アドレスが他のホストと重複しているなどで使用不可能な場合、それをサーバに知らせるためのメッセージ。

◆**DHCPINFORM** ：DHCP 以外の手段で既に IP アドレスを設定されているクライアントが IP アドレス以外のネットワーク情報を要求するためのメッセージ。

●DHCP オプション RFC2132

クライアントへの割り当て IP アドレスを除く多くの情報は、オプションとして授受されます。それぞれのオプションは、次のようなフォーマットとなっていて、これらは必要なだけ連ねてオプションフィールドに格納されます。次に主なオプションを示します。

DHCP オプションのフォーマット

コード	データ長	データ
(1byte)	(1byte)	(可変長)

コード： オプションの種類を示す。
データ長： データ部の長さをバイト単位で示す。
データ： コードで示されるオプションの値。

◆53：メッセージタイプ

53	1	タイプ

その DHCP メッセージのタイプを示す必須オプションです。これが無いとただの BOOTP パケットになってしまいます。

値	タイプ
1	DHCPDISCOVER
2	DHCPOFFER
3	DHCPREQUEST
4	DHCPDECLINE
5	DHCPACK
6	DHCPNAK
7	DHCPRELEASE
8	DHCPINFORM

◆1：サブネットマスク

1	4	サブネットマスク

サーバの返す情報で、このネットワークで使用するサブネットマスク。

◆3：ルータ

3	n	ルータ、ルータ....

サーバの返す情報で、ルータの IP アドレス。複数入る(推奨順)こともあるので、n は 4 の倍数になります。

◆6：DNS サーバ

6	n	DNS、DNS....

サーバの返す情報で、DNS サーバの IP アドレス。複数入る(推奨順)こともあるので、n は 4 の倍数になります。

◆51： IP アドレス貸与(Lease)時間

51	4	Lease 時間[秒]

クライアントに IP アドレスが貸与される時間。サーバは DHCPOFFER で提供する値を示し、DHCPACK で確定値として指定します。クライアントが DHCPDISCOVER や DHCPREQUEST に入れたときは、サーバに要求する貸与時間を意味します。

◆58：リニューアル(T1)時間

◆59：リバインド(T2)時間

58	4	T1 時間[秒]

59	4	T2 時間[秒]

Lease 時間の延長のため RENEWING ステートに移行する時間と、それがうまくいかないとき REBINDING ステートに移行する時間。サーバが DHCPACK で確定値として指定します。

◆54：サーバ ID

54	4	サーバ ID

サーバからのメッセージに含まれ、個々のサーバを識別する ID（サーバの IP アドレス）が入ります。REQUESTING ステートにおいては要求メッセージに選んだサーバの ID を入れることによりリクエストを出すサーバを特定します。

◆61：クライアント ID

61	n	クライアント ID

個々の DHCP クライアントを識別するユニークな値。サーバが内部で持っている「クライアント－アドレス対応リスト」の識別用として使用します。通常は、そのクライアントのハードウェアタイプ(1)+ 物理アドレスの 7byte 長が使用されます。

BOOTP フィールドにもクライアントを識別できるハードウェアアドレスがあるので、このオプションは必須ではありません。

◆55：パラメータ要求リスト

55	n	コード、コード、コード ....

クライアントがサーバに要求するパラメータ(サブネットマスクやルータ等)を挙列したリスト。DHCPDISCOVER や DHCPREQUEST に入れてサーバに要求します。

◆50：リクエストする IP アドレス

50	4	IP アドレス

クライアントが特定の IP アドレス(以前使っていたものなど)を優先して割り当ててもらいたいとき DHCPDISCOVER に含めることができます。また、SELECTING ステートにおいてアドレス割り当て要求する DHCPREQUEST メッセージには、要求する IP アドレスを必ず含めなければなりません。

◆56：メッセージ文字列

56	n	"メッセージ"

サーバからの DHCPNAK やクライアントからの DHCPDECLINE などの否定応答メッセージに、その原因を示すメッセージ文字列を入れることができます。

◆52：オプションオーバーロード

52	1	0-3

BOOTP パケットのサーバ名とブートファイル名のフィールドもオプションフィールドとして使うとき、これを行うことを示すフラグです。

◆0：パッド

0

パディングとして使うオプションです。

◆255：終了

255

オプションフィールドの終了を示します。

●ステート別オプション使用基準

ステートによって推奨・必須または禁止されているオプションがあります(RFC2131)。それほど複雑なものではないので、これらを簡単にまとめておきます。

	メッセージの種類	DHCPINFORM	DHCPDISCOVER	DHCPREQUEST (1)	DHCPREQUEST (2)	DHCPDECLINE	DHCPRELEASE
	ユニキャスト／ブロードキャスト	BC	BC	BC	UC(9)/BC	UC	BC
BOOTP フィールド	クライアント IP アドレス	MUST	0	0	MAY	0	MUST
	割り当て IP アドレス	0	0	0	0	0	0
	サーバ IP アドレス	0	0	0	0	0	0
	リレーエージェント IP アドレス	0	0	0	0	0	0
	クライアントハードウェアアドレス	MUST	MUST	MUST	MUST	MUST	MUST
	サーバ名	0	0	0	0	0	0
	ブートファイル名	0	0	0	0	0	0
DHCP オプション	要求 IP アドレス	MUST NOT	MAY(3)	MUST(4)	MUST(7)NOT	MUST(8)	MUST NOT
	貸与(Lease)時間	MUST NOT	MAY(5)	MAY(5)	MAY(5)	MUST NOT	MUST NOT
	クライアント ID	MAY	MAY	MAY	MAY	MAY	MAY
	サーバ ID	MUST NOT	MUST NOT	MUST(6)	MUST NOT	MUST(6)	MUST(6)
	パラメータ要求リスト	MAY	MAY	MAY	MAY	MUST NOT	MUST NOT
	最大メッセージサイズ	MAY	MAY	MAY	MAY	MSUT NOT	MUST NOT
	メッセージ文字列	SHOULD NOT	SHOULD NOT	SHOULD NOT	SHLULD NOT	SHOULD	SHOULD
	その他	MAY	MAY	MAY	MAY	MUST NOT	MUST NOT

(1) REQUESTING ステートでの。
(2) REBOOTING、RENEWING、REBINDING ステートでの。
(3) 使っていた IP アドレスを優先的に割り当ててもらいたいとき。
(4) 割り当てを要求する IP アドレス。
(5) 要求する貸与時間。
(6) ブロードキャストにおいてサーバを特定する必要があるため。
(7) REBOOTING では MUST。
(8) 問題のあった IP アドレス。
(9) RENEWING ステートにおいては UC。

RFC の読み方

まず、英語を覚えてください(汗)。いや、冗談ではなくて、RFC は英文なので、それが読めないと話になりません。でも、心配することはありません。英語といっても技術系の文章なので決まったパターンの言い回しが多く、慣れるのはそれほど難しいことではないと思います。日頃半導体のデータシートを読んだりしているのと大して変わりません。それでも多くの日本人にとって英文を読むのは骨が折れるものですが…。どっかに全 RFC 日本語訳ってのはないものですかねぇ。(あったら教えてください:-)

このプロジェクトに当たって参照した RFC 等各種文献を最後に示しておきます。どれも重要な物ばかりなので、プロトコルを書くときは必ず必要になります。

用語とインプリメントの基準

RFC を読んでいくと、MUST とか SHOULD といった単語が頻繁に出てきます。これらの単語は実際のインプリメントの際の判断基準になるので、RFC を参照するときは常に留意しておく必要があります。

MUST = ～しなければならない（必須）
SHOULD = ～すべきである（しなくても違反ではない）
MAY = ～してもよい（任意）
MAY NOT = ～しなくてもよい（任意）
SHOULD NOT = ～すべきでない（しても違反ではない）
MUST NOT = ～してはならない（禁止）

●実際の DHCP 動作時のパケット ダンプ

Pkt n-	Time (h:m:s)	Dest. MAC	Src. MAC	Network	Description
1	08:13:50.67...	FFFFFF-FFFFFF	0090CC-A0599A	IP: 0.0.0.0 => 255.255.255.255 (328)	UDP: Length= 308, Port (68 => 67) DHCP: Discover
2	08:13:50.68...	0090CC-A0599A	00000E-8B5B0D	IP: 192.168.1.1 => 192.168.1.3 (576)	UDP: Length= 556, Port (67 => 68) DHCP: Offer
3	08:13:50.68...	FFFFFF-FFFFFF	0090CC-A0599A	IP: 0.0.0.0 => 255.255.255.255 (328)	UDP: Length= 308, Port (68 => 67) DHCP: Request
4	08:13:50.69...	0090CC-A0599A	00000E-8B5B0D	IP: 192.168.1.1 => 192.168.1.3 (576)	UDP: Length= 556, Port (67 => 68) DHCP: Ack

Packet.3 DHCPREQUEST

0000	FF FF FF FF FF FF 00 90 CC A0 59 9A 08 00 45 00	[MAC]
0010	01 48 01 00 00 00 80 11 38 A6 00 00 00 00 FF FF	SRC=0090CC-A0599A, DST=FFFFFF-FFFFFF, TYPE=0x800(IP)
0020	FF FF 00 44 00 43 01 34 8F 16 01 01 06 00 82 02	[UDP/IP]
0030	82 02 00 00 00 00 00 00 00 00 00 00 00 00 00 00	SRC=0.0.0.0:68, DST=255.255.255.255:67
0040	00 00 00 00 00 00 00 90 CC A0 59 9A 00 00 00 00	[BOOTP]
0050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	OP=1, HTYPE=1, HLEN=6, HOPS=0, ID=0x82028202, SECS=0, B=0
	---- 中略 ----	CIADDR=null, YIADDR=null, SIADDR=null, GIADDR=null
0110	00 00 00 00 00 00 63 82 53 63 35 01 03 3D 07 01	CHADDR=0090CCA0599A
0120	00 90 CC A0 59 9A 32 04 C0 A8 01 03 36 04 C0 A8	[BOOTP.OPTION(DHCP)]
0130	01 01 0C 08 43 68 6F 62 69 74 73 00 37 08 01 03	Magiccookie=0x63825363, MessageType=3(DHCPREQUEST)
0140	06 0F 2C 2E 2F 39 FF 00 00 00 00 00 00 00 00 00	CrientID=0x010090CCA0599A, RequestedIP=192.168.1.3
0150	00 00 00 00 00 00	ServerID=192.168.1.1, HostName="Chobits"
※枠内は MAC/IP/UDP ヘッダ		ParameterRequestList=1,3,6,15,44,46,47,57

Packet.4 DHCPACK

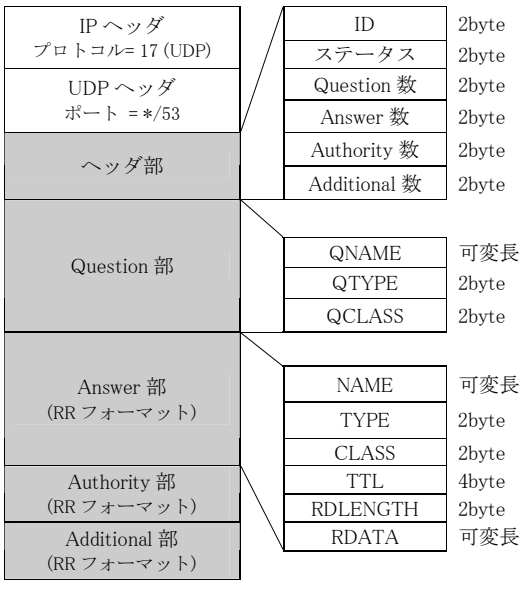
0000	00 90 CC A0 59 9A 00 00 0E 8B 5B 0D 08 00 45 00	[MAC]
0010	02 40 52 29 40 00 20 11 83 2F C0 A8 01 01 C0 A8	SRC=00000E-8B5B0D, DST=0090CC-A0599A, TYPE=0x800(IP)
0020	01 03 00 43 00 44 02 2C 76 B6 02 01 06 00 82 02	[UDP/IP]
0030	82 02 00 00 00 00 C0 A8 01 03 C0 A8 01 03 00 00	SRC=192.168.1.1:67, DST=192.168.1.3:68
0040	00 00 00 00 00 00 00 90 CC A0 59 9A 00 00 00 00	[BOOTP]
0050	00 00 00 00 00 00 FF 00 00 00 00 00 00 00 00 00	OP=2, HTYPE=1, HLEN=6, HOPS=0, ID=0x82028202, SECS=0
	---- 中略 ----	CIADDR=192.168.1.3, YIADDR=192.168.1.3, SIADDR=null, GIADDR=null
0110	00 00 00 00 00 00 63 82 53 63 35 01 05 36 04 C0	CHADDR=0090CCA0599A
0120	A8 01 01 33 04 00 01 51 80 3A 04 00 00 A8 C0 3B	[BOOTP.OPTION(DHCP)]
0130	04 00 01 27 50 01 04 FF FF FF 00 03 04 C0 A8 01	Magiccookie=0x63825363, MessageType=5(DHCPACK)
0140	01 06 04 C0 A8 01 01 0C 08 43 68 6F 62 69 74 73	ServerID=192.168.1.1, Lease=24hrs, T1=12hrs, T2=21
0150	00 3D 07 01 00 90 CC A0 59 9A 34 01 03 FF 00 00	SubnetMask=255.255.255.0, Router=192.168.1.1, DNS=192.168.1.1
		HostName="Chobits", CrientID=0x010090CCA0599A, OptionOverHead=3

DNS (Domain Name System) RFC1035

DNS とは、"www.who.com" といった階層化されたドメイン名から IP アドレスを得るためのシステムの総称です。DNS は世界中の DNS サーバ同士の連携によってとてもうまく実現されています。DNS 全体の説明はほかに譲り、ここでは、**DNS クライアントがサーバにアクセスしてドメイン名から IP アドレスを引くことに絞って説明**します。

まず、DNS クライアントは、「"www.who.com"のアドレスを教えてください」といったリクエストメッセージを DNS サーバに送ります。DNS サーバはドメイン名から IP アドレスを検索して、「"www.who.com" のアドレスは n.n.n.n です」と、応答メッセージを返します。このように DNS サーバとの通信は 1 往復で済むため、**効率の良い UDP が使われます** (TCP も定義されているが、使われていない)。**DNS サーバのポート番号は 53** で、クライアント側は任意です。

●DNS メッセージ



DNSメッセージは図に示すように内容別に5つのセクションに分けられます。

●ヘッダ部

このDNSメッセージの種類と、続くセクションの数を指定します。

◆ID ： クライアントが設定する送信毎に違う識別番号。サーバはこの値を応答メッセージにコピーしてクライアントに応答します。クライアントはこの値で送ったどのリクエストに対する回答かを識別します。

◆ステータス ：

QR	OPCODE (4bit)	AA	TC	RD	RA	予約 (3bit)	RCODE(4bit)

各種フラグや命令コード、リザルトコードなどが含まれる16bitのフィールドです。

QR（Query/Response）ビット ： このメッセージが質問(0)か回答(1)かを示します。

OPCODE：質問メッセージで、その種類(0:順引き、1:逆引き、2:サーバ状態要求）を示します。

AA（Authoritative Answer）ビット ： 回答メッセージで、サーバは質問されたドメイン名を管理していることを示します。

TC（TrunCation）ビット ： 回答メッセージで、メッセージサイズ制限のため、一部のレコードが削られた(1)ことを示すビットです。

RD（Recursion Desired）ビット ： 質問メッセージで、サーバに再帰検索（階層化されたドメイン名をホストに到達するまで検索すること）を要求(1)するビットです。

RA（Recursion Available）ビット： 回答メッセージで、サーバが再帰検索機能を持っていた場合にセットされます。

RCODE： 回答メッセージで、その結果（0:正常終了、1～5:各種エラー）を示します。

◆Question/Answer/Authority/Additional 数 ：それぞれのセクション内のレコード数(1 個とは限らない)。各レコードが可変長で長さを示す情報が無いので、これを元に Question 部から順に辿っていかないと目的のデータが得られません。

●Question 部

サーバへの質問内容を格納するセクションです。

◆QNAME ：質問するドメイン名。可変長フィールドですが、長さを得るにはこのフィールドを解析しなければなりません。ドメイン名はドット境界で**ラベル**という単位に分解されて格納されます。それぞれのラベルのフォーマットは、<ラベル長>,<ラベル文字列>となります。例として、"www.who.co.jp"を格納した例を示します。名前フィールドは長さ 0 のラベルまたはリンクで終了します。

0020: 03 77(w) 77(w) 77(w) 03 77(w) 68(h) 6F(o) 02 63(c) 6F(o) 02 6A(j) 70(p) 00

ラベル圧縮 ：DNSメッセージ長は512byte以下と決まっています。しかし、長いドメイン名が多く含まれるとその制限を越えてしまうことがあるので、ラベルの圧縮が定義されています。ドメイン名は後ろが同じ文字列が多いので、いわゆる「以下同文」という方法で圧縮を行うのです。例えば、上の名前を含むメッセージに"mail.who.co.jp"を加えるとすると、"who.co.jp"の部分が共通なので、

0050: 04 6D(m) 61(a) 69(i) 6C(l) C0 24 ←ｵﾌｾｯﾄ 0024 へリンク

のように圧縮できます。ラベル長の上位2bitが11なら、続く14bitでリンクオフセットとします。オフセットはメッセージの先頭(ヘッダ部)が基準になります。

DNSメッセージのサイズ制限
ラベル長：63byte以下
名前長：255byte以下
メッセージ長：512byte以下

◆QTYPE ：この質問レコードのタイプを示します。通常は、ホスト名(1)です。

◆QCLASS ：この質問レコードのクラスを示します。通常は、インターネット(1)です。

●Answer 部

サーバからの回答です。質問したドメイン名に複数のアドレスやエイリアス名がある場合、複数のレコードが入ることもあります。格納形式はRR(Resource Record)フォーマットというDNSで使われる共通のフォーマットで、TYPEとCLASSの値によってNAMEとRDATAの示す意味が変わってきます。

◆NAME ：このレコードの名前を示します。フォーマットはQNAMEと同じです。

◆TYPE ：このレコードのタイプを示します。通常の回答では、ホスト名(1)です。

◆CLASS ：このレコードのクラスを示します。通常の回答では、インターネット(1)です。

◆TTL ：このレコードの情報の有効時間[秒]。

◆RDLENGTH ：RDATAの長さをバイト単位で示します。

◆**RDATA** ：このレコードで示す名前に関するデータです。TYPE と CLASS の値が 1 の場合は、IP アドレスとなります。

●**Authrity/Additional 部**

そのドメイン名を管理している DNS サーバに関する情報などが入ります。DNS クライアントではこれらのセクションは無視してかまいません。

●**実際の DNS Query/Response の動作例**

Pkt n~	Time (h:m:s)	Dest. MAC	Src. MAC	Network	Description
1	04:27:02.21...	00A0DE-0BE388	009099-334E96	IP: 192.168.0.2 => 192.168.0.1 (58)	UDP: Length= 38, Port (1162 => 53) DNS: Query: Ne
2	04:27:02.21...	009099-334E96	00A0DE-0BE388	IP: 192.168.0.1 => 192.168.0.2 (151)	UDP: Length= 131, Port (53 => 1162) DNS: Response
3	04:27:02.21...	00A0DE-0BE388	009099-334E96	IP: 192.168.0.2 => 64.33.57.12 (48)	TCP: Port (1163 => 80) Data (SN 18030287, ACK 0, WIN
4	04:27:02.43	009099-334E96	00A0DE-0BE388	IP: 64.33.57.12 => 192.168.0.2 (44)	TCP: Port (80 => 1163) Data (SN 1201937630, ACK 1803

```
Packet.1 DNS Query                                                    [UDP/IP]
0000  00 A0 DE 0B  E3 88 00 90  99 33 4E 96  08 00 45 00  ........3N...E.  SRC=192.168.0.2:1162, DST=192.168.0.1:53
0010  00 3A 1C 05  00 00 80 11  9D 5A C0 A8  00 02 C0 A8  .:.......Z......  [DNS.HEADER]
0020  00 01 04 8A  00 35 00 26  29 7E 00 01  01 00 00 01  .....5.&)~......  ID=1, QR=0(Query), OPCODE=0(Normal), RD=1
0030  00 00 00 00  00 00 08 65  6C 6D 2D 63  68 61 6E 03  .......elm-chan.  nQuery=1, nAnswer=0, nAuthority=0, nAdditional=0
0040  6F 72 67 00  00 01 00 01                          org.....          [DNS.QUESTION 1]
※枠内は MAC/IP/UDP 各ヘッダ。下線はヘッダ部と質問部                              QNAME="elm-chan.org", QTYPE=1, QCLASS=1

Packet.2 DNS Response                                                 [UDP/IP]
0000  00 90 99 33  4E 96 00 A0  DE 0B E3 88  08 00 45 00  ...3N.........E.  SRC=192.168.0.1:53, DST=192.168.0.2:1162
0010  00 97 15 80  00 00 40 11  E3 82 C0 A8  00 01 C0 A8  ......@.........  [DNS.HEADER]
0020  00 02 00 35  04 8A 00 83  E0 22 00 01  81 80 00 01  ...5.....".....  ID=1, QR=1(Response), AA=0, TC=0, RA=1
0030  00 01 00 02  00 02 08 65  6C 6D 2D 63  68 61 6E 03  .......elm-chan.  RCODE=0(NoError)
0040  6F 72 67 00  00 01 00 01  C0 0C 00 01  00 01 00 00  org.............  nQuery=1, nAnswer=1, nAuthority=2, nAdditional=2
0050  2A 39 00 04  40 21 39 0C  C0 0C 00 02  00 01 00 00  *9..@!9.........  [DNS.QUESTION 1]
0060  2A 39 00 0F  02 6E 73 06  68 6F 73 74  34 75 03 6E  *9...ns.host4u.n  QNAME="elm-chan.org", QTYPE=1, QCLASS=1
0070  65 74 00 C0  0C 00 02 00  01 00 00 2A  39 00 06 03  et.........*9...  [DNS.ANSWER 1]
0080  6E 73 32 C0  3D C0 3A 00  01 00 01 00  00 2A 39 00  ns2.=.:......*9.  NAME="elm-chan.org", TYPE=1, CLASS=1
0090  04 D1 96 80  1E C0 55 00  01 00 01 00  00 2A 39 00  ......U......*9.  TTL=10809, RDATA=64.33.57.12
00A0  04 D1 96 81  03   ※下線はヘッダ部、質問部と回答部                    .....             --- 以下略 ---
```

参考文献

(1) RFC768: User Datagram Protocol (**UDP**)
(2) RFC791: Internet Protocol (**IP**)
(3) RFC792: Internet Control Message Protocol (**ICMP**)
(4) RFC793: Transmission Control Protocol (**TCP**)
(5) RFC821: Simple Mail Transfer Protocol (**SMTP**)
(6) RFC826: Address Resolution Protocol (**ARP**)
(7) RFC854: Telnet Protocol Specification (**TELNET**)
(8) RFC855-861: (**TELNET オプション**)
(9) RFC950: Internet Standard Subnetting Procedure (**サブネット**)
(10) RFC959: File Transfer Protocol (**FTP**)
(11) RFC1034: Domain Names - Concepts and Facilities (**DNS 概要**)
(12) RFC1035: Domain Names - Impremention and Specifications (**DNS 詳細**)
(13) RFC1350: The TFTP Protocol (**TFTP**)
(14) RFC1542: Bootstrap Protocol (**BOOTP**)
(15) RFC1661: The Point-to-Point Protocol (**PPP**)
(16) RFC1700: Assigned Numbers (**各種割り当て番号**)
(17) RFC1918: Address Allocation for Private Internets (**ローカル IP アドレス**)
(18) RFC1945: Hypertext Transfer Protocol (**HTTP**)
(19) RFC2131: Dynamic Host Configuration Protocol (**DHCP**)
(20) RFC2132: DHCP Options and BOOTP Vendor Extensions (**DHCP オプション**)
(21) 山居正幸, 「イーサネットと TCP/IP」, OpenDesign No.3, CQ 出版, 1994

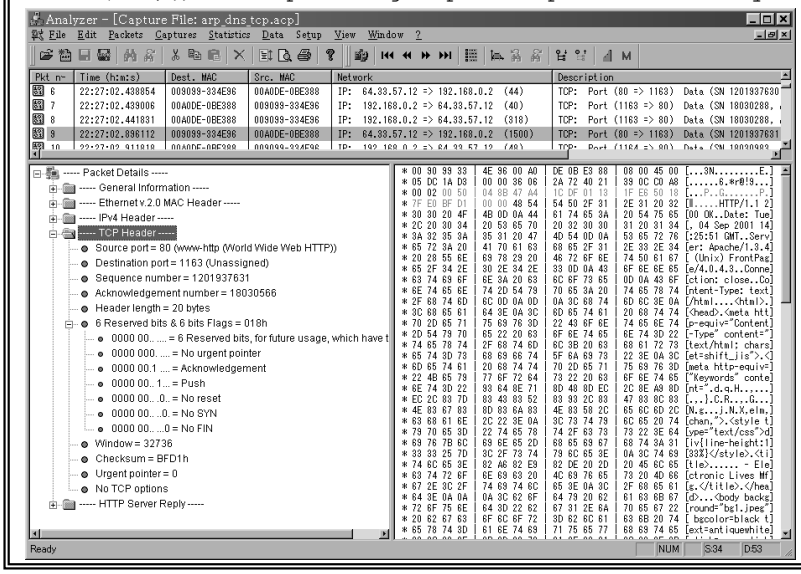
Column

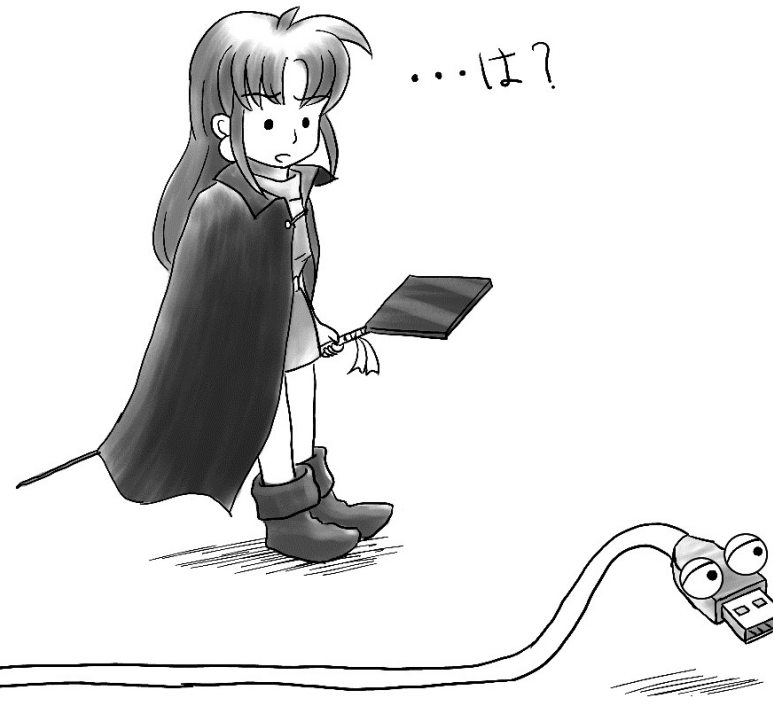
ネットワークアナライザ

プロトコルの動作のデバッグには、LANアナライザが不可欠となります。LANアナライザとは、ネットワーク上を流れるパケットを収集して通信を解析する測定器のことです。これがあるのと無いのとではデバッグ効率がまるで違ってきます。

LANアナライザとしては、専用機からパソコン上で動作するソフトまでピンキリですが、通常は後者で十分でしょう。ここではフリーで公開されている LAN アナライザを取り上げてみます。このアナライザは単体のアプリケーションではなく、汎用のパケットキャプチャドライバと、そのクライアントとしてのアナライザから成ります。

キャプチャドライバは、http://netgroup-serv.polito.it/WinPcap/ で、
アナライザ本体は、http://netgroup-serv.polito.it/analyzer/ で配布されています。





はじめに

PC の周辺機器のインターフェイスとして USB はすっかり定着しました。トランジスタ技術などの技術誌でも USB インターフェイスを使用したデバイスの設計手法の記事が普通にみられるようになり、秋月通商のような秋葉原の一般の電子部品店でも USB スレーブコントローラを入手できるようになっています。

USB を使うメリットはデータ転送速度の高さと PC との親和性にあり、多くの汎用周辺機器が USB スレーブデバイスとして発売されています。そのため、それらの豊富な USB 機器をマイコンにつなげることができれば、USB という1つのポートに様々な PC の汎用デバイスをつなげて使用することができますので、ホストの機能を実装することもかなりのメリットがあります。

そこで、本稿では技術誌においてあまり紹介されていない USB のホスト側の詳細な動作を解説し、実際に小規模マイコンの1つである AVR マイコンを使って USB ホストコントローラを制御し、USB ホストシステムを実現する方法について解説します。

USBとは

USB(Universal Serial Bus)は差動シリアルバスの一種で、現行のリビジョン 1.1 では 1.5Mbps(LS デバイス)と 12Mbps(FS デバイス)、そして最新のリビジョン 2.0 では 480Mbps(HS デバイス)の速度を出すことのできるバスです。USB は構造が簡単なため低コストで実現することができます。さらにプラグアンドプレイが十分に考慮されており、バスにデバイスを差し込むだけで、すぐにホスト側はそのデバイスを使用することができるようになっており、ユーザーには大変使いやすいものとなっています。

Windows において USB は、Windows98 以降で標準サポートされ、そのプラグアンドプレイ機能や低消費電力機能と組み合わされ、標準的な WindowsPC でのインターフェイスとなっており、レガシーデバイスであるパラレルポートやシリアルポートと置き換わりつつあります。

従来、外部デバイス制御用に一般的に使われてきた RS-232C と比べると、USB を Windows で使うにはデバイスドライバを作成しなければならないことや、プラグアンドプレイなどの機能をサポートするために、いくらかのネゴシエーションのためのプロトコルが実装されているため、少々取っつきにくい部分があります。しかし、USB スレーブデバイスを作る分には、難しい部分は USB コントローラがある程度処理してくれますし、プロトコルで面倒な部分はホスト側となる PC が行ってくれますので、実際には決まり事さえ覚えてしまえばさほど難しいものではありません。

しかし、逆に USB で設計が難しいのはホスト側で

す。基本的に USB はホスト主導で制御されるバスですので、接続されるデバイスやデータ転送シーケンスの管理などはホストがやらなければなりません。さらに、接続の際のバスの設定、デバイスの判別もホストがすべて行う必要がありますので、ホストはデバイス情報に関するデータベースの管理も行う必要があります。要は、スレーブ側の処理が簡単な分、ホストは処理が複雑になっているのです。以上のようなことを踏まえつつ、まずは USB を制御するために最低限必要な解説を手短に行っていきます。

転送方法の種類

USB では転送方法として次の4種類が定義されています。これらは、通常は用途によって使い分けられています。

コントロール転送

コントロール転送は、基本的な USB デバイスの制御を行う標準リクエストのやりとりに一般的に使用されます。また、1 回の転送で読み出し、書き込みが自由に行え、デバイスの応答も取得できる転送であるため、デバイス独自のコマンドをユーザーが適当に定義して、ターゲットとなるデバイスを細かく制御するのに使うことができます。ただし、1回の転送での手順が複雑であるために、あまりスピードが出ないので、大量のデータの送受信には使用されません。

バルク転送

この転送方法は、大量のデータを送受信し、かつ送受信データが保証されなければならない転送に使用されます。ただし、データを保証するために、送受信エラーが発生したときには再送処理も行わねばならないため、転送に遅延が発生することがあります。このため、遅延が問題とならない転送で、送られるデータに十分な信頼性が必要とされる転送に使用されます。ただし、この転送は LS モードでは使用できません。

インタラプト転送

この転送方法は、キーボードやマウスなどデータ転送量の少ないデバイスで、かつデータを送る頻度が低いものに使用されます。インタラプト転送と言っても割り込みとしてデバイス主導でデータを送るのではなく、あくまでホスト側が定期的に送信、受信要求を出して転送されますので、正確には間欠転送と言った方がよいでしょう。

アイソクロナス転送

バルク転送と同じように大量のデータの送受信を行うのに使用されますが、データ転送のリアルタイム性が優先されます。これはバルク転送と異なり、パケットにエラーが発生しても再送処理が行われないことで実現されますが、その代わりデータの送受信結果が保証されません。ただし、再送処理がないだけで各パケットに対する CRC チェックは入りますので、正常に転送されたパケットのデータの信頼性は保証されます。このため、オーディオデータの送受信、ビデオキャプチャなどのリアルタイム性の重要な機器でのデータ転送に使用されます。なお、この転送は LS モードでは使用できません。

パケット

USB では**表1**に挙げるようなパケットの組み合わせで、前述のような4種類の転送が制御されます。USB2.0 ではさらに多くのパケットが定義されていますが、ここでは USB1.1 で定義されるパケットについてのみ挙げています。

各転送方式での制御

USB では各デバイスに固有のアドレスがホストによって振られ、ホストがデバイスと通信を行いたい場合には、デバイスの**アドレス**と、送りたいパケットを受け取るデバイス内の**エンドポイント**と言われる内部ポートアドレスを指定して通信を行います。アドレスは 0 ～ 127、エンドポイントは 0 ～ 15 の値を取りますが、アドレス 0 はデフォルトの初期化用アドレスとして予約されていますのでデバイスには割り当てられません。またエンドポイント 0 は、ホストとデバイス間のネゴシエーションなどに使用するコントロール転送を行うための専用のエンドポイントとしてのみ使用されます。先に解説した4つの転送方式におけるパケットの組み合わせを**図1**に示します。基本的にデータ転送ステージでは DATA0/1 のデータパケットが交互に送受信されますが、アイソクロナス転送のみが同一の DATA0 パケットを送られます。このとき、パケットは 1ms ごとに周期的、規則的に送られ、そのパケットのサイズは後述のエンドポイントデスクリプタの設定で決まります。

表1　パケットの種類

名称	意味
SOF	フレームのはじまりを示します。通常1msごとに定期的に発行されます。
SETUP	コントロール転送のリクエストを発行します
OUT	ホスト→デバイスへの転送を開始します。
IN	デバイス→ホストへの転送を開始します。
DATA0	偶数パケットIDをもつデータパケット
DATA1	奇数パケットIDをもつデータパケット
ACK	転送が成功したことを相手に通知する
NAK	転送がエラー発生などで失敗したことを相手に通知する（リトライ要求的な意味）
STALL	転送が失敗し、これ以上継続できなくなったことを相手に通知する（アボート要求的な意味）
PRE	LSモードの通信を開始します。

USB機器のネゴシエーション

USB 機器はホストに接続されると**図2**のような手順でネゴシエーションされて、バスの中に組み入れられます。この際のアドレスは、最初デフォルトアドレス 0 とエンドポイント 0 が使用されます。設定のやりとりが完了すると、今後通信に使用するアドレスをホストがデバイスに対して指示し、さらに細かな設定のやりとりを行った後、ホストとスレーブの設定を調停してコンフィグレーションを確定させます。

このようなやりとりに使用される情報がデスクリプタと呼ばれるもので、代表的になものが4つあります。1つが**デバイスデスクリプタ**というもので、接続されるデバイスのおおまかな素性、メーカー ID、プロダクト ID が入っています。Windows では、ここで設定されている ID を使って、どのような機器が接続されたかを判断し、内蔵しているデバイスデーターベースから適合するドライバを探し出して、その組み込みを行います。

2つめが**コンフィグレーションデスクリプタ**と言われるもので、後述のインターフェイス全体をまとめる1つの構成要素であるインターフェイスデスクリプタ数や、消費電力情報などが入ります。

3つめが**インターフェイスデスクリプタ**というもので、デバイス内のインターフェイスにあるエンドポイント数やデバイスタイプなどの基本的な設定が記述されています。例えば1つのデバイスに2つの機器をもち、それぞれを USB で制御可能である場合、それら2つに対してそれぞれ1つづつ、計2つのインターフェイスデスクリプタが存在します。

4つめが**エンドポイントデスクリプタ**と言われるもので、デバイス内に存在するエンドポイントの設定が記述され、その内容は該当のエンドポイントのサポートする転送種類（コントロール転送、バルク転送など）や最大パケットサイズ、インタラプト転送やアイソクロナス転送における転送間隔などです。

これらの4つのデスクリプタをやりとりすることで、ホストとスレーブの通信が確立されます。

図2　USBデバイスの初期化手順

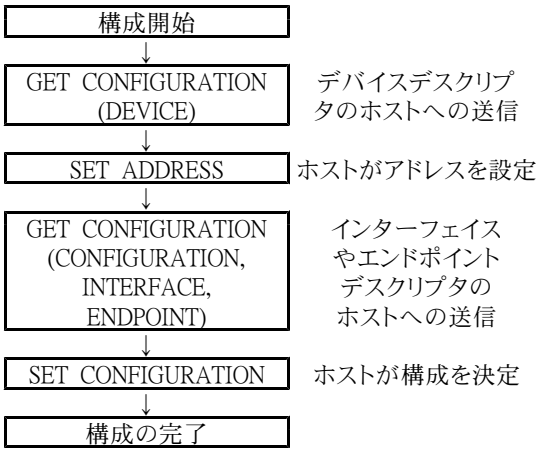
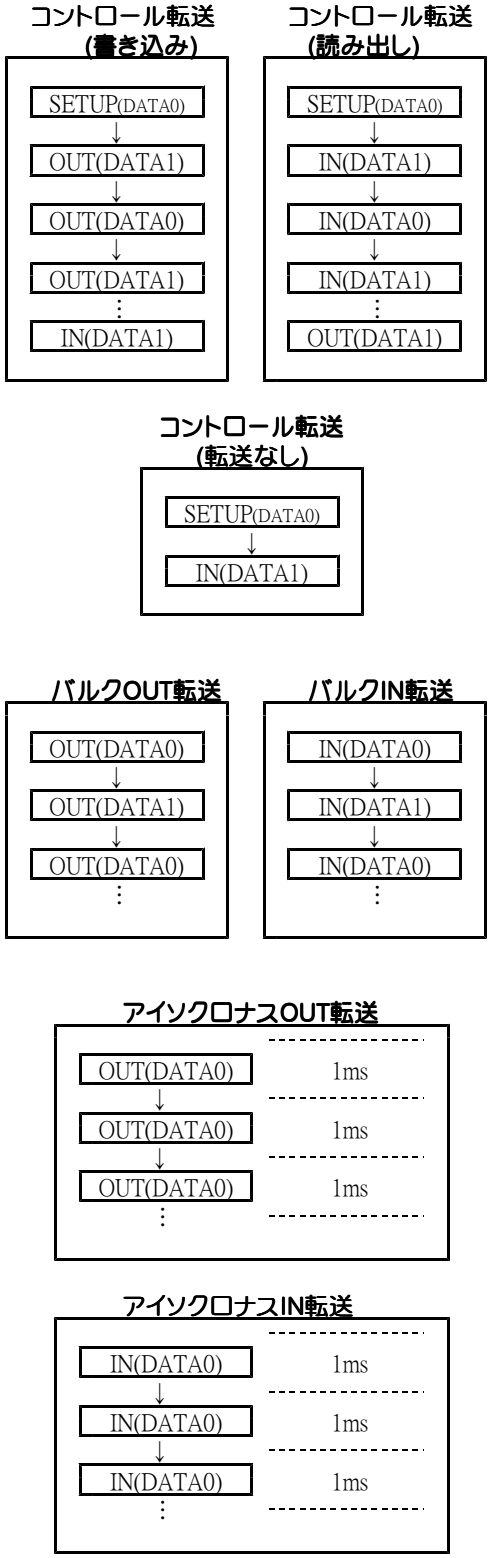


図1　転送方式によるパケット送信の違い



USBホストのしくみ

さて、簡単に USB スレーブデバイス側の動作を解説したところで、ホストの動作を解説していきます。USB はスレーブデバイスの開発は結構需要があるので、市販の解説書も数多く出回っているのですが、ホスト側は Windows が行うぐらいですので、あまりホスト側の動作をきちんと記述している文献は見かけることがありません。そのため、ここではなるべく丁寧に、特にハブに関する動作について解説していきたいと思います。

USBの接続形態

USB ホストがスレーブデバイスを認識する手順は USB の接続形態をうまく使ったものになっています。**図3**に USB デバイスの基本的な接続形態を示します。これをみるとわかるように、USB では物理的に1つのバス線に対して1つのデバイスしか繋がりません。

SCSI バスなどでは1つの電線に対して複数のデバイスの電線が繋がっていましたが、USB ではそのようなことはなく、あくまで1対1で繋がっていることが重要です。さらに、位置的に下流のハブから上位のバスに還流するような流れは存在せず、常に下流に流れるような構成となっています。

これらのことが前提となって、デバイスの検出などが正常にできるように USB のプロトコルは規定されています。

デバイスの挿抜検出

USB ホストと USB スレーブデバイスの一般的なインターフェイス回路は**図4**のようになっています。USB クライアントが USB ホストのポートに接続されると、まず VBUS と GND が接続され、そのあとにデータ線である D+と D-が接続されるようなコネクタ構造となっています。

図4を見ればわかるように、USB ホストでは D+と D-はそれぞれプルダウンされていますので、デバイス未接続時にはどちらもLレベルとなっています。最初に説明したように USB は差動シリアルバスであるため、そのデータ線 D+と D-は差動信号線、つまり普通は D+=NOT D-という状態でデータが送られます。そのため、このようにどちらも同じレベルになっているのは特殊な状態で、実はこれは USB のバスリセット状態(SE0)と同じ状態になっています。

そして、USB スレーブデバイス側では D+または D-の信号線がプルアップされており、USB ホストとクライアントを接続すると、D+か D-のどちらかが H レベルになります。このときの D+= D-= L → D+≠ D-の状態遷移をホスト側が検出することで、USB ホストは簡単にデバイスの挿抜を検出することができるようになっています。

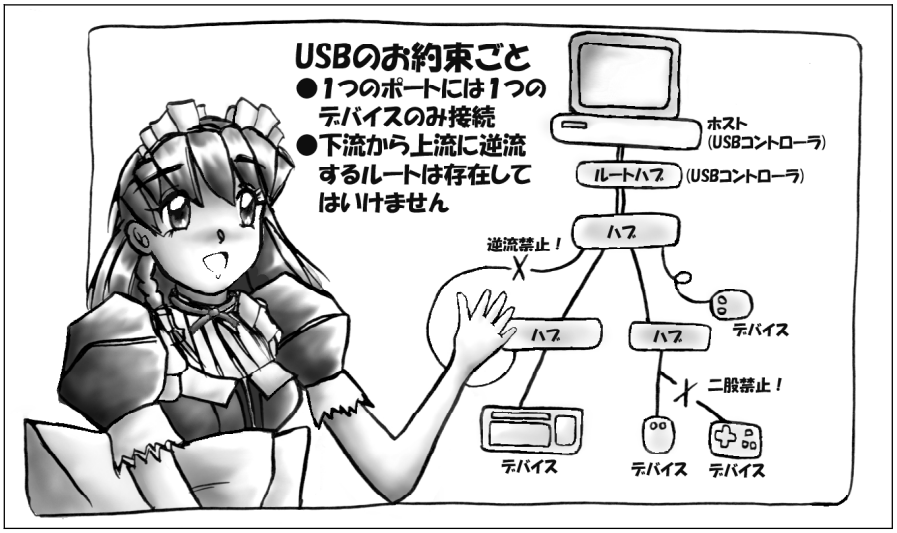
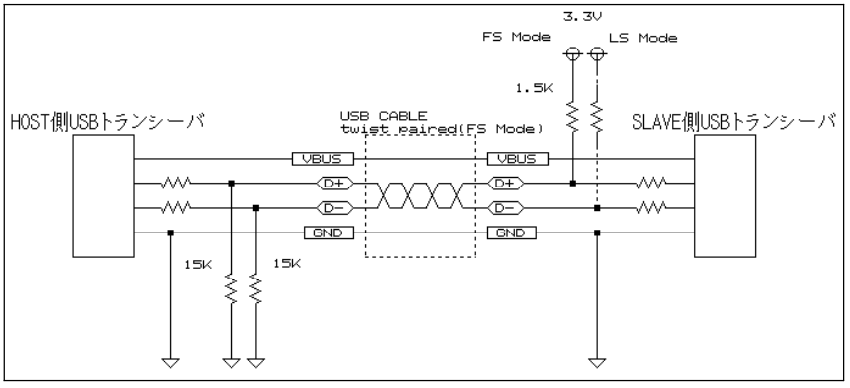


図3
USBの接続形態

図4
USBインターフェイス回路



速度の異なるデバイスの検出

USB スレーブデバイスのうち、FS デバイスと LS デバイスではプルアップされている信号線が異なります。前者は D+が、後者は D-がプルアップされており、USB ホストとクライアントを接続されたときにどちらが H レベルになっているかをみて、接続されているデバイスの速度を認識します。

なお、LS デバイスと FS デバイスはプルアップされている信号線が異なるだけでなく、転送されるデータの論理も逆になっています。このため、スレーブデバイス側で LS/FS を切り替えるときには、プルアップする信号線を変えるだけでなく、データ転送時の極性も変更する必要があります。

デバイスの設定

デバイスが接続されると、一度バスをリセットしたあとに実際の通信を開始します。まず SOF パケットを 1ms 間隔で定期的に発行するようにします。そのあと、最初の USB 機器のネゴシエーションのところで解説したように、いくつかのデスクリプタのパラメータの整合を行いアドレスの設定を行います。

コマンドの構造は**表2**のようになっており、それぞれ**表6**のように設定した標準コマンドを一般的に USB スレーブデバイスに対して、コントロール転送の SETUP トランザクションのときに送信します。

GET CONFIGURATION(CONFIGURATION)の場合、実際に取得しなければならないデータ数は取得した CONFIGURATION の中のデータ数のメンバをみないとわかりませんので、最初にホストはそれなりに大きいサイズでデータを取得し、足りなければさらに大きいデータサイズで受信リクエストします。このため、CONFIGURATION は1度だけで終わる場合があります。なお、GET CONFIGURATION では wValue の上位 8 ビットにデスクリプタタイプ(DEVICE/CONFIGURATION など)を、下位 8 ビットに取得したい CONFIGURATION のインデックス値を設定します。

この際取得される CONFIGURATION のインターフェイスデスクリプタ**(表4)**にはデバイスの素性が詳しく書かれています。たとえば、HID デバイスクラスのキーボードやマウスであれば、そのデバイスクラスとデバイスプロトコルが書かれていますので、GET CONFIGURATION(DEVICE)**(表3)**で取得されるベンダ ID やプロダクト ID と共にデバイスを判別する情報となります。

さらに、もう一つ取得されるエンドポイントデスクリプタ**(表5)**で、各エンドポイントの転送方法、間隔、ペイロードサイズを取得できますので、これらの値をみてクライアントが送受信可能な転送パラメータにホスト側の通信パラメータを調整します。

最後にSET CONFIGURATION を実行した段階で USB スレーブデバイスは構成され、実際に使用できるようになります。

表2　標準リクエストパケットの構造

メンバ名	内容
bRequestType	Bit7 ： **データ転送方向** ホスト→デバイス、デバイス→ホスト
	Bit6～5 ： **タイプ** 標準リクエスト、クラス、ベンダ、予約
	Bit4～0 ： **受信側の種類** デバイス、インターフェイス エンドポイント、その他、予約
bRequest	リクエスト
wValue	bRequestの定める値
wIndex	bRequestの定める値
wLength	データ転送時の転送バイト数

表3　デバイスデスクリプタの構造

メンバ名	内容
bLength	デスクリプタのサイズ(0x12)
bDescriptor	デスクリプタのサイズ(0x01)
bcdUSB	USB使用リリース番号
bDeviceClass	クラスコード(なし、ベンダ、予約)
bDeviceSubClass	サブクラスコード
bDeviceProtocol	プロトコルコード
bMaxPacketSize0	エンドポイント0の最大パケットサイズ
idVendor	ベンダID(USB IF割り当て)
idProduct	プロダクトID
bcdDevice	デバイスのリリース番号
iManufacturer	製造者名へのストリングデスクリプタのインデックス
iProduct	製品名同上
iSerialNumber	シリアル番号同上
bNumConfigurations	構成可能数

表4　インターフェイスデスクリプタの構造

メンバ名	内容
bLength	デスクリプタのサイズ(0x09)
bDescriptor	デスクリプタのタイプ(0x04)
bInterfaceNumber	このインターフェイスのインデックス
bAlternateSetting	SetInterfaceの代替設定値
bNumEndpoints	エンドポイント数(エンドポイント0を除く)
bInterfaceClass	クラスコード
bInterfaceSubClass	サブクラスコード
bInterfaceProtocol	プロトコルコード
iInterface	ストリングデスクリプタへのインデックス

表5　エンドポイントデスクリプタの構造

メンバ名	内容
bLength	デスクリプタのサイズ(0x07)
bDescriptor	デスクリプタの種類(0x05)
bEndpointAddress	エンドポイントのアドレスと転送方向
bmAttributes	転送種類(コントロール、バルク、インタラプト、アイソクロナス)
wMaxPacketSize	最大パケットサイズ
bInterval	ポーリング間隔(ms)　アイソクロナスでは常に1。

表6　コマンド送出手順(一般的な手順)

順番	bRequest	アドレス	wValue	bmRequestType	wIndex	wLength
1	GET CONFIGURATION	0	DEVICE とインデックス	0x80	1	取得データサイズ
2	SET ADDRESS	0	設定するアドレス	0x00	0	0
3	GET CONFIGURATION	*1	CONFIGURATION とインデックス	0x80	0	取得データサイズ
	GET CONFIGURATION (取得数が足りないとき)	*1	CONFIGURATION とインデックス	0x80	0	取得データサイズ
4	SET CONFIGURATION	*1	設定する CONFIGURATION	0x00	0	0

(*1)はSET ADDRESSで設定したアドレス

USBハブ

前述のように USB の1つのポートには物理的に1つのデバイスしかつながりません。このため、複数のデバイスを接続するには**USBハブ**を接続する必要があります。USB ハブは**図3**を見ればわかるように、ホストのルートハブにぶら下がる形態で入ります。このうち、上位のポートに繋がるポートをアップストリームポートといい、ぶら下がるデバイスが接続されるポートをダウンストリームポートと呼びます。このようにホストに 2 つ以上のデバイスを接続する場合には、USB スレーブデバイスの代わりに USB ハブを接続してポートを増やすわけです。さらに多くのデバイスを繋ぐときには、ハブのダウンストリームポートにさらにハブを繋いでポートを増設するというように、ツリー状にバスを拡張することができます。

USBハブのしくみ

USB ハブは Ethernet のハブと形態が似ているので、まったく同じように単に物理的にポートを増設するものと考えられがちですが、USB のハブはちょっと様子が異なります。それは、ハブ自身が USB スレーブデバイスの一種であり、単に繋ぐだけでダウンストリーム側のポートがすぐに使えるわけではないことです。このため、ハブの下にぶら下がる USB デバイスを有効にするには、USB デバイスであるハブに対して、ダウンストリームポートに接続されているデバイスを有効にするための特定の手順に従って初期化処理を行わなければなりません。

ハブの制御

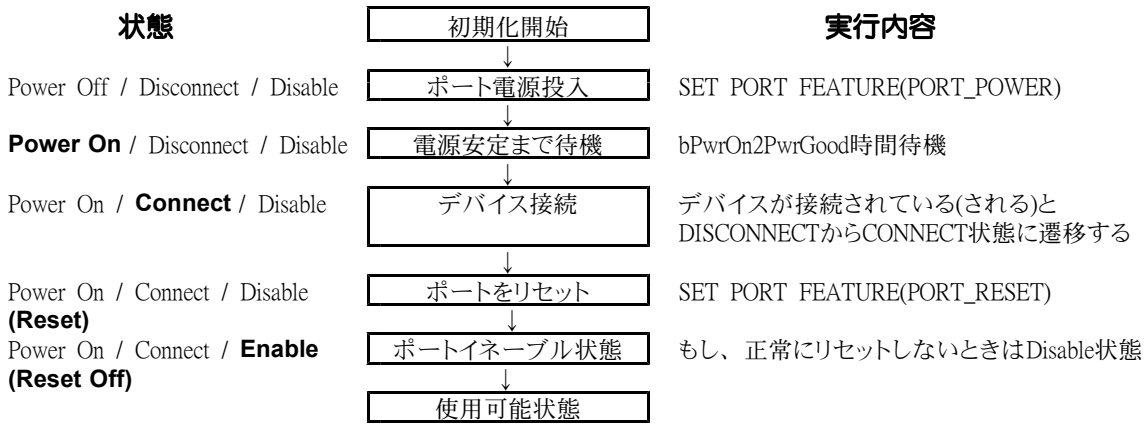
まず、USB のルートにハブが接続されると、通常の USB スレーブデバイスと同様の手順でアドレスが確定されハブが構成されます。

そして、構成を完了したあと、ホストはハブ特有のコマンドを使って、ハブ自身の構成や素性を検出してあげます。ハブで使用するコマンドの一覧を**表8**に示します。

最初に標準リクエスト(GET DESCRIPTOR)を使って取得した、デバイスデスクリプタ(**表4**)のデバイスクラスとインターフェイスデスクリプタに、ハブの場合はそれぞれのデバイスクラス、インターフェイスクラスのところにデバイス番号 9(HUB)が入っています。ここで、接続されたものがハブであることを認識し、デバイスを構成したあと、GET HUB DESCRIPTOR を実行してハブのデスクリプタ(**表7**)を取得します。

この内容のうち最低限必要なのは、ハブに存在するポートの数(bNbrPorts)とポート電源を ON してから有効になるまでの時間(bPwrOn2PwrGood)です。これらの値を覚えておき、後に続くハブの各ダウンストリームポートの初期化を行います。

図5　ハブのダウンストリームポートの初期化手順



ハブを経由したデバイスの検出と制御

図5にハブに接続されるデバイスの状態遷移を示します。基本的にハブのアップストリームポートが上位ポートと接続された直後は、ダウンストリームポートの状態は、状態遷移表での一番上に当たる POWER OFF、かつ DISCONNECT(切断)となっています。つまり、最初の状態ではダウンストリームポートに対して USB ケーブルでデバイスが繋がっていても、USB バス全体としてはなにも意味を持ちません。

この状態からポートの電源を入れてあげます。ポートの電源の投入などのポート状態の設定の変更には SET PORT FEATURE コマンドを使って PORT_POWER ステータスを設定します。その後、電源が安定化するまで待機します。この待機時間は先の bPwrOn2PwrGood パラメータで指定されている時間分を待機しなければなりません。ちなみに筆者の手元のハブでは 100ms 程度に設定されていましたので、結構な時間がかかります。このため、必ず指定された時間分はウェイトを入れる必要があります。

ここでステートは POWER ON かつ DISCONNECT 状態となっています。もし、ここで既にデバイスが接続されていれば CONNECT(接続)状態になり、ポートは DISABLE 状態になります。接続されていなければ DISCONNECT 状態のままとなります。このような指定のポートの状態は GET PORT STATUS(表8、9)を発行することでわかりますので、ここで実際にポートにデバイスが接続されているかをホストが確認できます。

ダウンストリームポートが CONNECT 状態になっていれば、ポートに対してリセット SET PORT FEATURE(PORT_RESET) を行います。これを行うとハブは指定のダウンストリームポートに対してバスリセットを行います。正常にリセットが行われれば、そのポートは ENABLE 状態となり、そこに繋がったデバイスと通信ができるようになります。

あとは、普通のポートに繋がっているデバイスと同様に、デバイスデスクリプタを取得し、アドレスを設定し、インターフェイスデスクリプタやエンドポイントデスクリプタのネゴシエーションを行ったあと、コンフィグレーションを構成します。なお、バスパワードハブでは1ポート当たり 100mA までの電流しか許可されませんので、このとき取得したコンフィグレーションデスクリプタをみて、接続されたデバイスが 100mA 以下の消費電流の機器かどうかをチェックし、それ以上であれば接続を拒否するようにしなければいけません。

このような一連の1つのポートに対する操作を GET HUB DESCRIPTOR で取得した bNbrPorts 分行います。各ポートのデバイス初期化は、1つ1つのダウンストリームポートを順番に Enable 状態にして行っていくため、デフォルトアドレス 0 で通信しても必ず USB バスの中では新しく接続されたデバイスは1つしか存在しない状態となっています。このため、同時に複数のスレーブデバイスが挿入されても競合することなく初期化されるわけです。

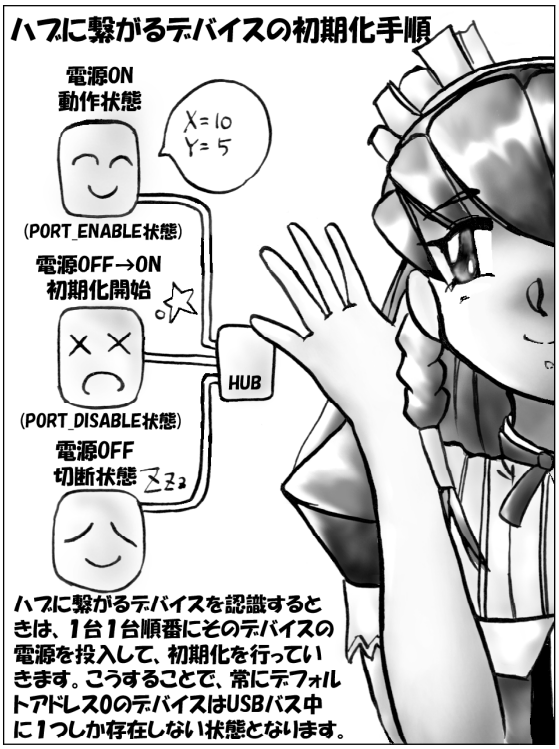


表7 ハブデスクリプタの構造

Offset	メンバ名	意味
0	bDescLength	デスクリプタ長
1	bDescripterType	29h固定
2	bNbrPorts	ハブの持つポートの数
3	wHubCharacteristics	ポートの属性 D0~1: ポートの電源ONの制御 D2: Hubが復号デバイスかどうか (0:複合デバイスでない) D3~4: 過電流保護モード D5~15: 予約
5	bPwrOn2PwrGood	ポートをPOWER ONしてから有効になるまでの時間(2ms単位)
6	bHubContrCurrent	コントローラの必要な最大の電流値(mA)
7	DeviceRemovable	ポートのデバイスがRemovableかどうか Bit0: 予約 Bit1: Port1, Bit2: Port2 .... (0:Removable, 1: No-Removable)
…	PortPwrCtrlMask	USB 1.0との互換用

表8　ハブを操作するコマンド群

Request	bmRequestType	bRequest	wValue	wIndex	wLength	データ
ClearHubFeature	00100000B	CLEAR_FEATURE	機能選択	0	0	なし
ClearPortFeature	00100011B	CLEAR_FEATURE	機能選択	ポート	0	なし
GetBusStatus	10100011B	GET_STATE	0	ポート	1	ポートごとのバスステータス
GetHubDescriptor	10100000B	GET_DESCRIPTOR	デスクリプタタイプとインデックス	0または言語ID	デスクリプタ長	デスクリプタ
GetHubStatus	10100000B	GET_STATUS	0	0	4	ハブ状態と状態変更インジケータ
GetPortStatus	10100011B	GET_STATUS	0	ポート	4	ポート状態と状態変更インジケータ
SetHubDescriptor	00100000B	SET_DESCRIPTOR	デスクリプタタイプとインデックス	0または言語ID	デスクリプタ長	デスクリプタ
SetHubFeature	00100000B	SET_FEATURE	機能選択	0	0	なし
SetPortFeature	00100011B	SET_FEATURE	機能選択	ポート	0	なし

表9　GetPortStatusで得られるポート状態と状態変更インジケータ

Bit	PORT_STATUS_BITS	Bit	PORT_STATUS_CHANGE_BITS
0	接続(0:接続無し,1:接続有り)	0	接続(0:変更無し,1:変更有り)
1	使用可(0:Disabled,1:Enabled)	1	使用可・不可(0:変更無し,1:変更有り)
2	サスペンド (0:No Suspend, 1: Suspended)	2	サスペンド状態(0:変更無し,1:変更有り)
3	過電流(0:過電流ポートなし,1:あり)	3	過電流(0:変化なし,1:変化あり)
4	リセット(0:ノーマル,1:リセット状態)	4	リセット状態(0:変化なし,1:変化あり)
5~7	リザーブ	5~15	リザーブ
8	電源(0:OFF,1:ON)		
9	LSデバイス接続(0:FS,1:LS)		
10~15	リザーブ		

図6　ハブのアップストリームポートとダウンストリームポートの通信

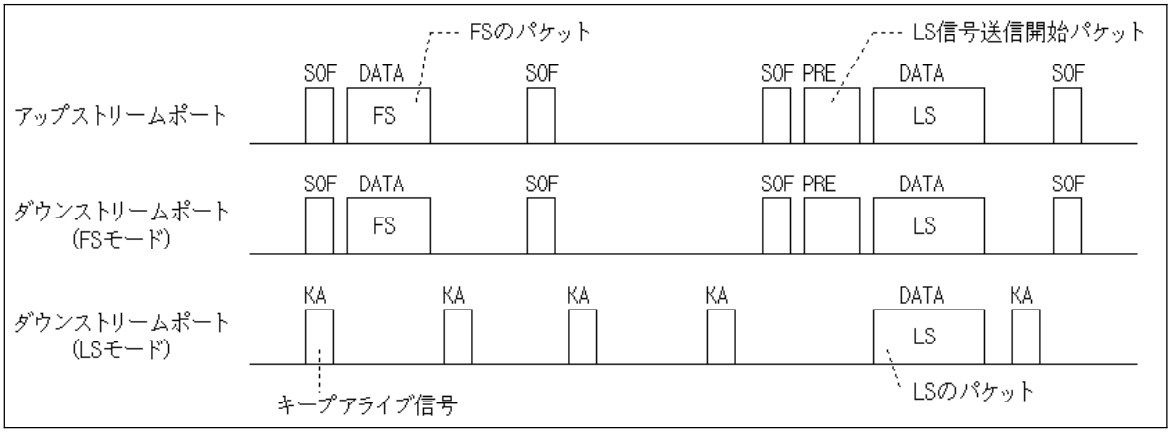
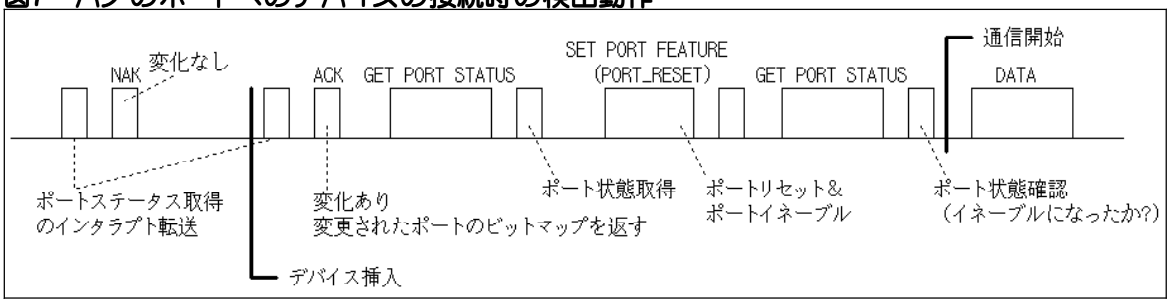


図7　ハブのポートへのデバイスの接続時の検出動作



ハブのダウンストリームポートに接続されたLSデバイスとの通信

FS モードで上位ポートが接続されているハブで、そのダウンストリームポートに LS モードのデバイスが接続された場合、そのデバイスとホストの間の通信は少し事情が変わってきます。このような場合、ホストの定期的に発行する SOF パケットはハブを透過せず、ハブから直接デバイスに対してキープアライブ(KeepAlive)信号が発行され、デバイスがサスペンドしないようにします。

実際にホスト側からハブにぶら下がっている LS デバイスに対してコマンドを発行する場合には、必ず PRE(PREAMBLE)パケットを送ってから、実際に送信するパケットを送らなければなりません。この PRE パケットの後に送信されるパケットが、LS パケットとしてデバイスに送られることになります(**図6**)。

このため、ポートを ENABLE 状態にするときには、必ず GET PORT STATUS を発行したときに、そのポートに接続されているデバイスが FS デバイスか LS デバイスかを検出しなければなりません。そして、該当のデバイスが LS デバイスであるときには、必ず通信の際に PRE パケットを一緒に送るように設計する必要があります。

ハブに繋がるデバイスの挿抜の検出

ホストのポートがデバイスの挿抜を検出する場合は、D+と D-の信号線の状態を見さえすればよかったのですが、ホストのルートハブのバスとは物理的に切り離されている USB ハブのダウンストリームポートにぶら下がるデバイスでは、このようにはいきません。ではどのように検出するのかというと、インタラプト転送により定期的にハブのダウンストリームポートの変更状態を示すデータを取得する処理を行って、ソフトウェア的にこれを検出します(**図7**)。このインタラプト転送に使用するエンドポイントの情報は、最初のネゴシエーション時に取得されるエンドポイントデスクリプタで得られますので、通信はその設定に従い行われます。

ハブは自分のダウンストリームポートの状態に変化がなければ、このホストのポート状態データ受信要求に NAK を返します。もし変更があれば、ポート状態情報を返します(**表10**)。ホストはそれをみて、もし変更されたポートがホスト内の情報において、デバイス未接続状態であれば接続処理を開始し、既にデバイスが接続されている状態であれば、そのポートのデバイス接続情報のクリア処理を行います。

正常にポート状態更新処理を行ったあと、必ずそのポートの更新情報ビットを ClearPortFeature コマンドを使って落とします。これを落とさないと更新ビットが立ちっぱなしになりますので、ポート情報を更新したあともインタラプト転送での状態検出で NAK が返らない状態に陥ります。このとき基本的には、更新ビットすべて(C_PORT_*)を ClearPortFeature を使って落とす必要があります。なお、最初のパワーオン時の全体構成検出のときにも更新ビットが立ってしまいますので、ポートの状態を変更したときには、必ず該当する更新ビットをクリアせねばなりません。

このようにハブに繋がるデバイスの検出は、ソフトウェア的に行われますので、ハブを繋ぐほどその状態を把握するためのパケットが USB バスの帯域を使っていくことになりますので、あまり多くのハブを繋ぐべきではないことになります。

表10　ハブの状態変更ビット(15ポート時)

MSB　　LSB	MSB	LSB
PORT15～8	PORT7～1	HUB

Bit0はハブの状態に変化があったときに立つ。
Bit1～15は、順にPORT1,2,3...と割り当てられ、変更があったポートのビットが立つ。

HIDデバイス

今回試作する USB ホストシステムでは HID デバイスをサポートします。この HID デバイスは Human Interface Device の略で、キーボードやマウスなどの一般的なコンピュータの標準的入出力装置のことです。USB 規格では、このような一般的なデバイスの入出力プロトコルを厳密に決め、HID デバイスクラスを持つ USB デバイスでは、様々なメーカーの少々仕様の異なるデバイスでも統一的に扱えるようにしています。

レポートデスクリプタ

レポートデスクリプタは前述のようなデバイスメーカー間の細かい違いのある製品の出力するデータが「どのような形式で出力されるか」を記述したフォーマットデータです。これを取得するには、GET REPORT コマンドを使用します。一例として、**表11**に筆者が持っていた USB キーボードのレポートデスクリプタを、**表12**にその実際のデータ出力形式を示します。

このデスクリプタは基本的にビットの下位から、かつデータパケットのオフセット 0 から記述されます。1 つのブロックは基本的に**表14**の項目で 1 セットとなります。このうち、パラメータを省略すると、そのパラメータは 0 扱いとなります。

例えば、サンプルの**表11**の USB キーボードの場合、最初の**機能キーパラメータ①**はキーボードのシフトキーなどの定義となっています。このアイテム設定では、キーコードページで、論理値が 0 ～ 1 ですから、8ビットの値1つにそれぞれ E0 ～ E7 の値が割り当てられていることを示しています。つまり、オフセット 0 のビット 1 が 1 であるとき、USB キーコード E1 が ON であることを示します。

この次にダミーデータ(Constant Input パラメータ）として 8 ビットの固定データが1つ入ることを示しています。つまりオフセット1はダミーデータです。

次の**一般キーデータ②**は実際に押されているキーの情報です。キーコードで、1つのデータが 8 ビットで 6 バイトあり、範囲は 0 ～ 255 で入力のときに配列で使用されることを示しています。

最後は**LED関係出力データ③**です。UsagePage は LED となっていますので、キーのインジケータ LED

を示します。下位 5 ビットが LED のデータで、1 つのデータサイズは 1 ビット(LogicalMaximum=1)となっていますので、ビット 0 ～ 4 は LED のページの 1 ～ 5 の LED に当たることを示します。

最後は残り3ビットがダミーデータであることを示します。前項のように 8 ビットに満たない値を設定したときは、必ず次のデータが 1 バイトをまたがないように余ったときはパディングビットで埋める必要があります。

表11　レポートデスクリプタの例 (USBキーボード)

05 01	usage page (Generic desktop)
09 06	usage(key board)
A1 01	application
05 07	usage page(keycode)　**機能キー①**
19 E0	usage min e0
29 E7	usage max e7
15 00	logical min 0
25 01	logical max 1
75 01	report size 1　1bit
95 08	report count 8 1bit×8=1byte
81 02	input (variable absolute)
75 08	report size 8　8bit　**固定値**
95 01	report count 1　8bit×1=1byte
81 01	input (constant)
05 07	usage page(key code)　**一般キー②**
19 00	usage min 0
2A FF 00	usage max 255
15 00	logical min 0
26 FF 00	logical max 255
75 08	report size　8bit
95 06	report count　8bit×6=6 byte
81 00	input (data array)
05 08	usage page(LED)　**LED関係③**
25 01	logical max 1
19 01	usage min 1
29 05	usage max 5
75 01	report size 1bit
95 05	report count　1bit×5=5bit
91 02	output (data valiable absolute)
75 03	report size 3　**パディング**
95 01	report count 1
91 01	output(constant)
C0	End Collection

表12　実際にデバイスから出力されるデータ列（表12のデバイス）

オフセット	内容
0	CTRL/ALT/SHIFTキーデータ
1	未定義
2 ～ 7	キーデータ

ブートプロトコル

さて、前述のレポートデスクリプタの話を読んで頭が痛くなった人も多いでしょう。

一見すればわかるように、レポートデスクリプタはあまりに設定が柔軟なので、ホストでこれをきちんと解析するのは大変なのです。スレーブデバイスは自分の設定だけ考えればよいのですが、ホストは繋がるデバイスすべてのレポートデスクリプタを考慮し、きちんと解析しなければいけません。

例えば、PC の BIOS で起動用にキーボードだけをサポートしようとする場合、このような複雑なレポートデスクリプタを解析するのは無意味です。そのため、キーボードなど基本的な入出力機器では、**ブートプロトコル**という特別なプロトコルが定義されています。逆にレポートデスクリプタを使って制御するプロトコルを**レポートプロトコル**と呼びます。

このブートプロコルでは、レポートデスクリプタの内容に左右されず、特定の形式でデータフィールドの意味が固定されたパケットでデータが出力されるようになっています。つまりレポートデスクリプタを解析しなくても、目的のデータを確実に取得できるというものです。ちなみに**表12**は**表11**のレポートデスクリプタの形式として紹介しましたが、ブートプロトコルでも同様の形式で出力されます。USB1.1 規格書ではブートプロトコルとして別のレポートデスクリプタを定義していますが、出力結果としては同じです。特殊な理由でも無い限り、キーボードではブートプロトコルとレポートプロトコルで別の出力形式を取る理由はありませんから、至極当たり前ではあります。

現状、ブートプロトコルが使えるデバイスは、キーボートとマウスだけですので、これ以外のデバイスはきちんとレポートデスクリプタを使用するレポートプロトコルを使用しないといけません。また、ブートプロトコルが使えるかどうかは、インターフェイスデスクリプタのプロトコルのメンバに書かれていますので、使う前には必ずこれをチェックしなければなりません。このような使用するプロトコルにブートプロトコルとレポートプロトコルのとちらを選択するかには SET PROTOCOL コマンドを発行します(**表15**)。

入力データの読み出し

一般的に HID デバイスは人間が入力する機器ですので、そのデータ転送は突発的で少ない場合がほとんどです。このため、一般的にはインタラプト転送によりそのデバイスの出力データは送られます。送られるデータの形式はレポートプロトコルであれば、レポートデスクリプタの形式に従い、ブートプロトコルであれば規格書で定められた形式で出力されます。

USB キーボードでブートプロトコルを選択すると**表12**のような形式でキーデータが出力されます。オフセット 0 がシフトキーなどの情報、オフセット3～7までがキーデータです。この6バイトのキーは押された順に FIFO でバッファリングされる感じで入っているわけではなく、その時点で押されているキーデータがすべて入り、先頭から順に早く押された順で入ります。例えば、A,B,C の順でキーを押し続けると、最

初に A だけ入ったデータが入り、B も押した時点で2番目に B が入った AB というデータが、さらに C を押すと ABC というデータが入った状態でデータパケットが取得できます。

出力データの設定

出力するデータ、例えばキーボードで言えば CAPS や NumLock の LED の表示の設定となりますが、これは普通は SET REPORT コマンドを使い、レポートデスクリプタの OUTPUT(表11③)で設定された形式のアイテムをデバイスに送ります。

この例で言えば、PC/AT のキーボードでは、PS/2 インターフェイスのものでもステータス LED はPCが全部制御します。そして、USB インターフェイスのキーボードでも、基本的に同様に PC 側から CAPS が押されたら CAPS LED を点灯させるなどの処理をする必要があります。キーボードの場合、LED の割り当てはブートプロトコルでは**表13**のようになっています。特定の LED を点灯させたい場合には、その LED に対応したビットを立てて、1バイトを SET REPORT コマンドで送るようにします。

ここではキーボードに関する LED の設定のみを紹介していますが、LED の定義には電話に関するものや、プリンタ、音源に関するもの実際に定義されているものは多岐に渡ります。LED に限らず、これらの HID に関するテーブルの定義は USB 規格書の HID Usage Tables にすべて書かれていますので、興味のある方はそちらを読んでみてください。

表13 ブートプロトコルでのLEDの割り当て

MSB LSB

Undefined (3bit)	かな	Compo-se	Scroll Lock	CAPS Lock	Num Lock

表14 レポートデスクリプタの1アイテムの要素の設定例

Input (Output / Feature)	設定値が使われるところ（入力、出力、設定）
Usage	インデックス
Usage Page	Logical/Usageで設定される項目
Logical Minimum	論理値の最低値
Logical Maximum	論理値の最大値
Usage Minimum	このアイテムに該当する値の最低値
Usage Maximum	このアイテムに該当する値の最大値
Report Size	Logical/Physicalで設定される値のビットサイズ
Report Count	ReportSize Bitのデータがいくつあるか

表15 HIDクラスデバイス用コマンド群

Request	bmRequestType	bRequest	wValue	wIndex	wLength	データ
GET REPORT	10100001B	GET_REPORT(1)	レポートタイプとID	インターフェイス	レポート長	レポート
SET REPORT	00100001B	SET_REPORT(9)	レポートタイプとID	インターフェイス	レポート長	レポート
GET IDLE	10100001B	GET_IDLE(2)	レポートID	インターフェイス	1	アイドルレート
SET IDLE	00100001B	SET_IDLE(10)	期間とレポートID	インターフェイス	0	なし
GET PROTOCOL	10100001B	GET_PROTOCOL(3)	0	インターフェイス	1	プロトコル
SET PROTOCOL	00100001B	SET_PROTOCOL(11)	プロトコル (0:ブート,1:レポート)	インターフェイス	0	なし

USBホストシステムの設計

今回、小規模の組み込み向け USB ホストコントローラが入手できましたので、これを使って実際に小規模な USB ホストシステムを設計してみます（というより、このチップをある方から頂いたので、この原稿の企画が発生したわけですが（笑））。

今回はあくまで試作ですので、HID デバイスの USB キーボードやマウスをポートに接続し、それらのデバイスから受け取ったデータをシリアルに変換して流す、というような基本的な動作を行うシステムとしました。

CPU には ATMEL の 8 ビットマイコンである AT90S8515-8 を使用します。この CPU は ROM 4KWord/RAM 512Byte と容量は少ないですが、高速で使いやすいのでこれを選択しています。逆にこれで動けば、まず H8 などの規模の大きな CPU でもきちんと動作させることができると言えます。

SL811HSTについて

使用した USB ホストコントローラは ScanLogic 社の SL811HST というものです。このコントローラは以下のような特徴を持っています。

- USB ホスト／スレーブ制御両対応
- USB Specification 1.1 対応
- FS(12Mbps)、LS(1.5Mbps)対応
- デバイススピード自動検出対応
- 8 ビットマイクロプロセッサ直結可能
- USB バストランシーバ内蔵
- USB ルートハブ機能内蔵
- 12/48MHz 発振子対応
  (12MHz 時は内部 4 逓倍)
- 5V インターフェイス直結可能
- 3.3V 単一電源動作
- SOF、CRC5/16 ハードウェア自動生成

コントローラ自体は 3.3V 動作となっていますが、I/O の入力耐圧が 5.0V となっていますので、5V 電源のバスに直接繋ぐことができます。また、きちんと USB 入出力インターフェイスがスレーブとホストに両対応していれば、ホストモードとスレーブモードをソフト的に切り替えることもできます。

さらに、USB ホストで必要な機能である SOF パケットや、PRE パケットの生成をハードウェアが自動的に行ってくれるだけでなく、パケットを構築するときに必要な CRC5/16 の生成も自動で行ってくれるようになっていますので、ソフトウェアの負担がかなり軽くなっています。

コントローラとCPUの回路構成

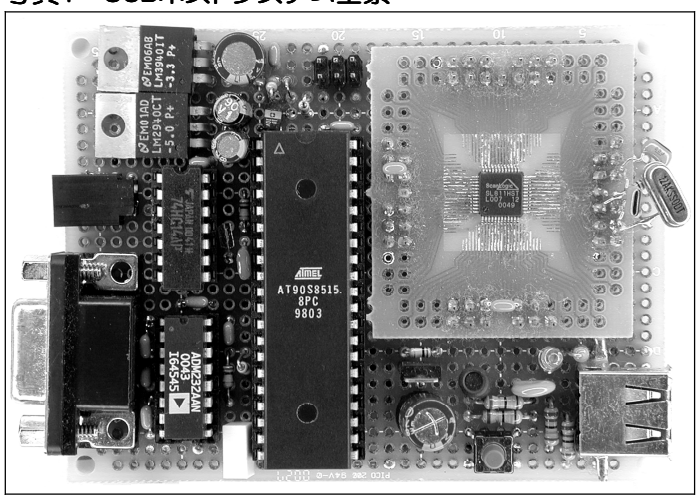
はっきり言って回路的にはあまり説明することはありません。AT90S8515 を SRAM アクセスモードにして、SL811HST を CPU に直結している形になっているだけです。

SL811HST は内部のレジスタにアクセスする際に、アドレスを書き込んだ後にデータを書き込むというマルチプレクスバスの一種となっています。アドレスを書き込むか、データを書き込むかの選択は A0 端子で行います。A0 が 0 のときにアドレスが書き込まれ、A0 が 1 のときに指定されたアドレスにデータが書かれます(図9,10)。

表16 USBホストシステムの部品表

部品番号	種類	メーカー	型番	個数	備考
U2	CPU	ATMEL	AT90S8515-8	1	
U1	USB ホストコントローラ	ScanLogic	SL811HST	1	
U3	RS232Cレベルシフタ	Analog Devices	ADM232A	1	
U4	CMOS	東芝	TC74HC14	1	
Q1	デジタルTR	ローム	DTC124XSA	1	
Q2	Power-MOSFET	NEC	2SJ325	1	I_{DS}=500mA以上でON抵抗が低いこと
U5	5.0Vレギュレータ	NS	LM2940CT	1	1A以上の出力があること
U6	3.3Vレギュレータ	NS	LM3940CT	1	
D1	LED	任意	任意	1	
X1	X'tal	任意	12.000MHz	1	48MHzも可(48MHz時、逓倍OFFで使用)
X2	セラミック振動子	ムラタ	4.912MHz	1	発振周波数はタイミング要考慮
R5,6	抵抗		15KΩ	2	
R9,10	抵抗		24Ω	2	
R1,3,4,8,11,14	抵抗		100KΩ	6	
R7	抵抗		680Ω	1	
R2,13	抵抗		100Ω	2	
R12	抵抗		1MΩ	1	
C2,3,6,7,8,9,10	セラミックC		0.1μF	7	
C12,13	セラミックC		20pF	2	
C5	電解C		100μF	1	
C14	電解C		10μF	2	
C11	電解C		330μF	1	
D2	Di	東芝	1S1588	1	
POLYSW	ポリスイッチ		max 500mA	1	過電流保護のため

写真1 USBホストシステム全景



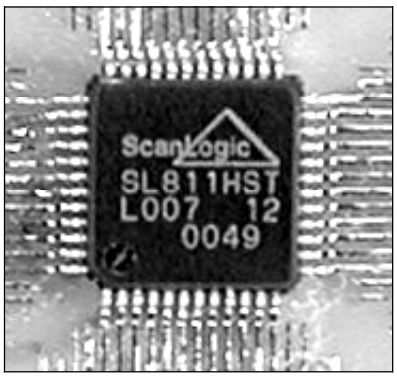
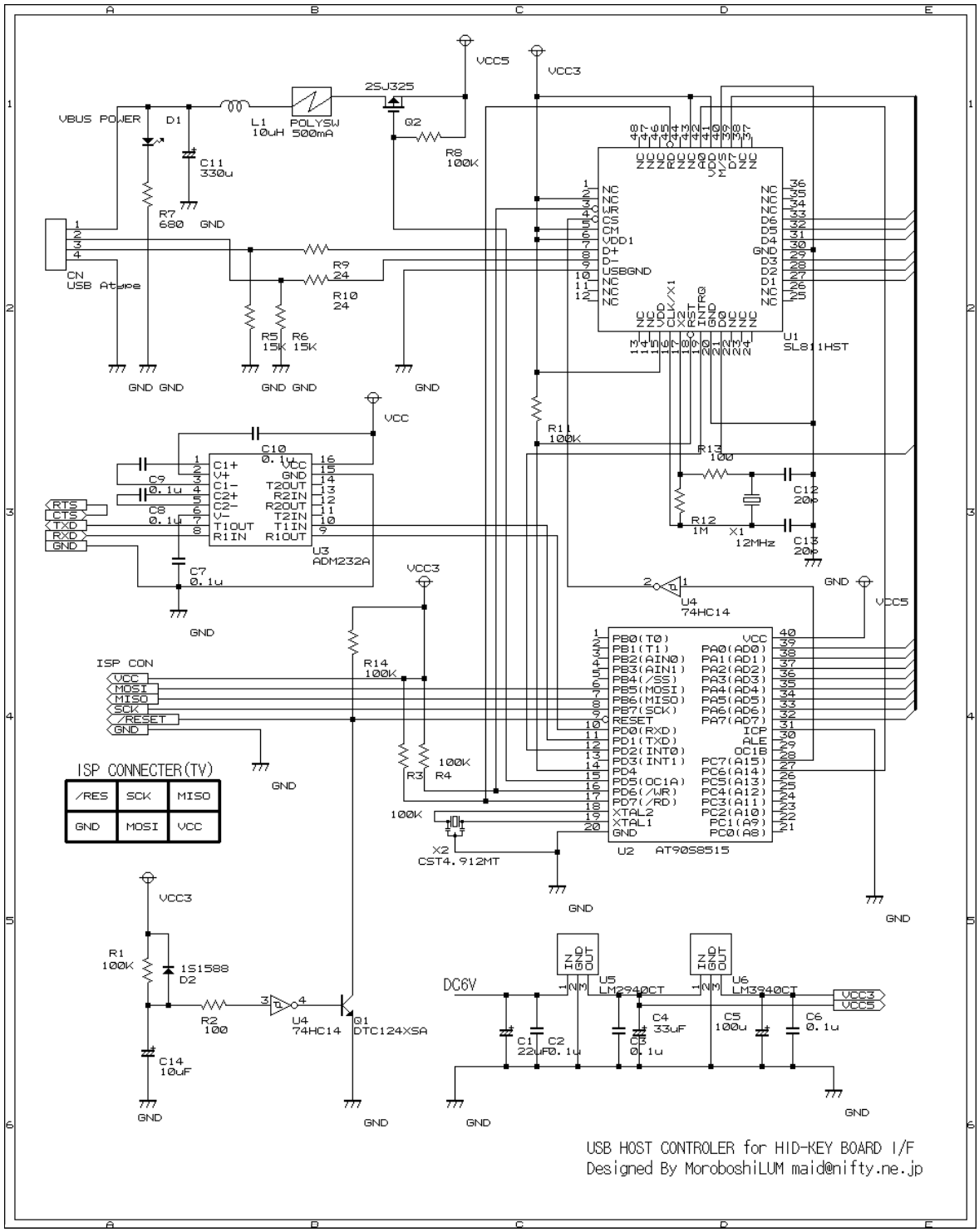


写真2 SL811HST USB Host/Slave Controler

図8　USBホストシステム全回路図



SL811HST では、通常のアドレスを書いてからデータを書くようなモードの他に、オートインクリメントモードという最初にアドレスとデータを書いた後に、アドレスを書かずにデータを読み書きするときには、自動的に内部で前回アクセスしたアドレスに 1 が加算されたアドレスにアクセスするモードがあります。

これを使用することで、SL811HST 内部のバッファに連続的に書き込むときには、一度書き込み開始アドレスを指定したあと、続けて連続してデータを書き込むだけで済むことになります。これにより必要なバスサイクル数が削減されますので、普通にアクセスするよりもいくらか高速なデータの転送が可能になります。

タイミングチャート上で注意しなければならないのは、アドレスサイクルからデータサイクルまでの間を 85ns 取る必要があることです。命令実行速度が速いマイコンでは、簡単にこの制限を越えてしまいますので、ソフト、またはハードできちんと抑えておく必要があります。またオートインクリメントモードでは、連続書き込み・読み込み間隔が最低 150ns ないといけませんので注意が必要です。

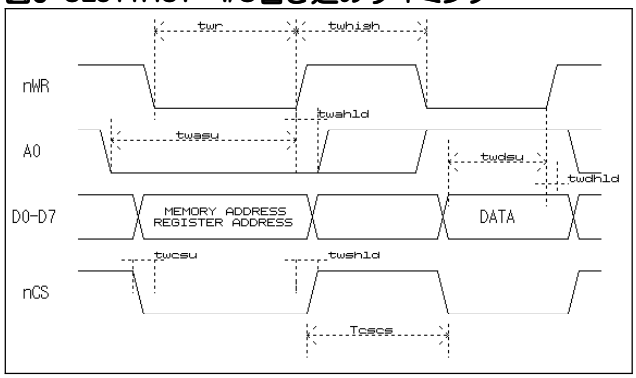
今回の回路では CS を CPU の A15、A0 を CPU の A14 に繋いでいます。このため、アドレス Cxxxh に書き込めばアドレスが、8xxxh に書き込めばデータが書き込めるようになっています。アクセスには AT90S8515 の SRAM アクセスモードを使っていますが、あまり速度と手間に拘らなければ、汎用 I/O ポートとして1ビットづつ端子を制御しても構いません。今回は特に他に制御するものもないのでこうしていますが、SRAM アクセスモードに設定することで無駄になるポートを有効活用したい場合などは I/O ポートとして設定して制御した方がよいでしょう。

USBホストインターフェイス回路

この回路では、最低限 USB のバス電源を FET などで ON/OFF できることと、ポリスイッチなどの過電流保護回路を入れておくようにしておく必要があります。あとはホストインターフェイスですので、**図4**のように D+/D-端子をすべて 15KΩ でプルダウンし、24Ω のダンピング抵抗を入れておきます。できればデータラインにフィルタを入れておく方がよいでしょう。

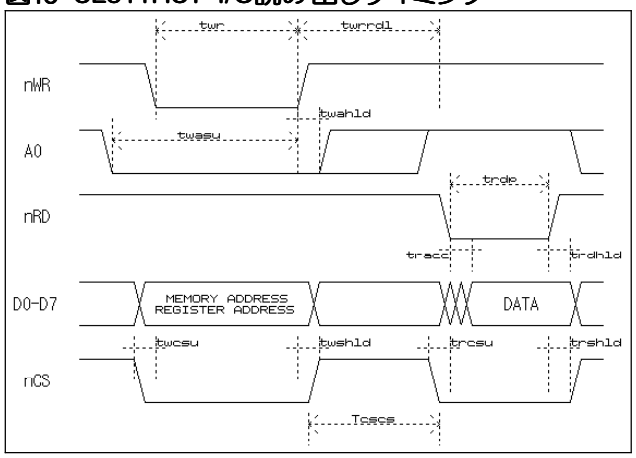
特に USB ホストでは電源をデバイスに供給するために、過電流保護回路は必須です。今回は VBUS にポリスイッチという自動復帰型のポリマーでできたブレーカーを使っています。これは、定格以上の電流が流れたときに電流を遮断しますが、時間が経つと元に戻る便利な部品です。今回は USB の1ポート当たりの制限である 500mA の物を使っています。

図9 SL811HST I/O書き込みタイミング



symbol	パラメータ	Min.	Max
twr	書き込みパルス幅	65ns	
twcsu	nWR=LまでのCSセットアップ時間	0ns	
twshld	nWR=H後のCSホールド時間	0ns	
twasu	A0アドレスセットアップ時間	65ns	
twahld	A0アドレスホールド時間	10ns	
twdsu	nWR=Lまでのデータセットアップ時間	60ns	
twdhld	nWR=H後のデータホールド時間	5ns	
tcscs	nCS非アクティブ→アクティブ時間	85ns	
twrhigh	アドレス書き込み後のnWR=H保持時間	85ns	

図10 SL811HST I/O読み出しタイミング



symbol	パラメータ	Min.	Max.
twr	書き込みパルス幅	65ns	
twcsu	nWR=LまでのCSセットアップ時間	0ns	
twshld	nWR=H後のCSホールド時間	0ns	
twasu	A0アドレスセットアップ時間	65ns	
twahld	A0アドレスホールド時間	10ns	
twdsu	nWR=Lまでのデータセットアップ時間	60ns	
twdhld	nWR=H後のデータホールド時間	5ns	
trd	読み出しパルス幅	65ns	
tracc	nRD=L後のデータ有効時間	20ns	25ns
trdhld	nRD=H後のデータホールド時間	5ns	
trcsu	CSアクティブ→nRDアクティブ時間	0ns	
trshld	nRD=H後のnCSのホールド時間	0ns	
twrrd	アドレス書き込み後のnWR=Hのnrdアクティブまでのホールド時間	85ns	

(注意) オートインクリメントモードでアクセスするときには、nRD/nWRともにアサート間隔を150ns以上広げること。

ファームウエアの全体の動作

設計したファームウェアの動作フローは、**図11**のようになっています。最初にポートを初期化したあと、ポートに繋がる機器を判別し構成した後、USB ハブでなければ検索処理を完了します。USB ハブであれば、そのハブに存在するダウンストリームポートに繋がるデバイスをすべて列挙して、デバイス情報を RAM に保存します。

すべてのデバイスを列挙したあとにキーボードやマウスデバイスをすべて探しだし、その初期化を行います。そのあと、見つけだしたデバイスに対して定期的にインタラプト転送を行ってデータの取得を行います。ハブがあればハブのダウンストリームポートの監視も行い、ポート状態に変更があれば、そのポートの内部デバイス情報を更新します。

デバイスの初期化と制御

今回サポートする USB デバイスは、HID デバイスでブートプロトコルをサポートするもののみ、としました。このため、デバイスの探索ではデバイスがキーボードかマウスかなどの種別だけでなく、インターフェイスデスクリプタを参照して、ブートプロトコルをサポートするかでもチェックしています。

デバイスを見つけだした後、SET PROTOCOL コマンドでブートプロトコルにセットしたあと、SET IDLE コマンドでデータの読み出し間隔を設定し、インタラプト転送を発生させる CPU の管理するタイマデータの初期化を行っています。

このタイマデータは 2ms 単位でタイマハンドラによりカウントダウンされます。メインルーチンではこのカウンタを常に監視しており、これが 0 になった時点でプログラムは指定のデバイスにインタラプト転送を行い、デバイスの種類に応じた処理を行うようになっています。処理が終わった後に、またカウンタの値を初期値に戻して、再度カウントダウンが始まります。これが連続して続くようになっています。

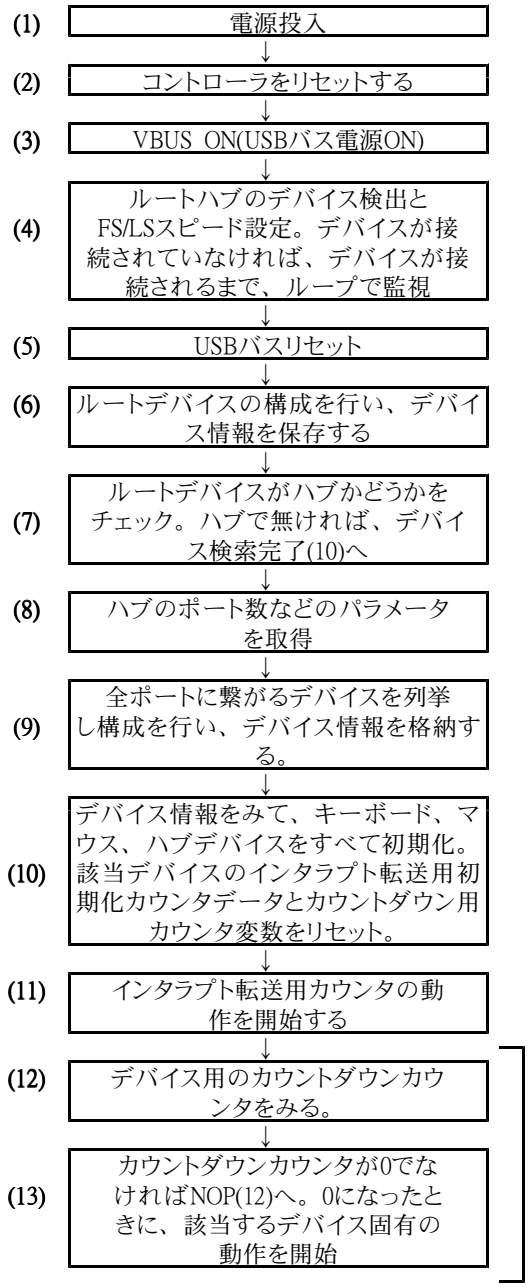
USB ハブに関しても同様なタイマを使って、定期的にポート状態データが受信、検査されます。もしポートの状態に変化があった場合には、そのポートの初期化処理と接続されているデバイスの認識、登録処理を行います。

SL811HST のレジスタ構成は**表17**のようになっています。基本的には、パケットレベルで制御します。EP0Status、EP0Address、EP0Counter などのレジスタに送受信したいパケットの情報とその対象となるデータのアドレス、または格納先のアドレスをセットしたあとに EP0Control レジスタに対して Arm(転送開始ビット)を Enable にして転送を開始したあと、USB 完了割り込みを待つという形が基本です。

割り込みが発生したら、ステータスを読み出し NAK であればリトライ処理、STALL であれば中断、ACK であればデータの読み出し、更新処理を行います。なお、実際に送受信するデータは SL811HST の内部 RAM に対して読み書きを行い、EP0Address レジスタでその内部 RAM アドレス指定しておきます。SL811HST は PingPong 転送に対応していますので、バッファを 2 面用意して、割り込みが入るたびにスイッチする形にしておけばよいでしょう。

注意しなければならないのは、ハブのダウンストリームポートに繋がる LS デバイスへのアクセスです。このときは、必ずレジスタの PREAMBLE ビットを立てて、アクセスを行います。

図11 ファームウエアの簡易動作フロー



(12)～(13)間は無限ループ。ハブのチェックでポートに変更があれば、該当デバイスの情報を更新する。ルートデバイスが抜かれた場合には、(4)へ戻る

表17 SL811HST内部レジスタ表

アドレス	レジスタ名	機能
0x00	EP0Control	USB-A ホストコントロールレジスタ
0x01	EP0Address	USB-A ホスト送受信アドレス
0x02	EP0XferLen	USB-A ホスト送受信長
0x03	EP0Status	USB-A ホストパケットID、デバイスエンドポイント(Write) USB-Aホストステータス(Read)
0x04	EP0Count	USB-A デバイスアドレス(Write) 転送カウント(Read)(EP0XferLenで指定された値が全て転送されれば0)
0x05	CtrlReg	コントロールレジスタ1
0x06	IntEna	割り込み許可レジスタ
0x07	Reserved	予約
0x08	EP1Control	USB-B ホストコントロールレジスタ
0x09	EP1Address	USB-B ホスト送受信開始アドレス
0x0A	EP1XferLen	USB-B ホスト送受信長
0x0B	EP1Status	USB-B ホストパケットID、デバイスエンドポイント(Write) USB-Bホストステータス(Read)
0x0C	EP1Count	USB-B デバイスアドレス(Write) 転送カウント(Read(EP1XferLenで指定された値が全て転送されれば0)
0x0D	IntStatus	割り込みステータスレジスタ
0x0E	DATAset	SOFカウンター下位8bit(Write) USB-A/Bトグルビット、ハードウェアリビジョン(Read)
0x0F	CSOFcnt	SOFカウンター上位5bit、コントロールレジスタ2(Write) SOFカウンターBit6～13(Read)

エラッタの対応

この手の初物デバイスでありがちなのが、チップのバグ(エラッタ)です。このチップにもしっかりバグがありますので、新しいシリコン 1.4 以外のチップでは、規定の対処コードを入れておく必要があります。このため、必ず ScanLogic のエラッタ情報には目を通しておいて下さい。筆者のチップはシリコン 1.2 でしたので、このエラッタ対応がファームウェアに入っています。それでもいまいち動作が不安定なときがありますが・・・。ちなみにこのエラッタは、ハブのダウンストリームポートに繋いだ LS デバイスに対するアクセスに関するものです。

最後に

USB 規格では 127 デバイスまでサポートしなければいけないのですが、AT90S8515 ではたったの 512 バイトしかメモリがありませんので、これほどはサポートできません。このため、現状は5デバイス（1ハブ、4デバイス）までとなっています。これ以上増やすには、もっと大きなメモリが必要です。ただ、512 バイトとはいえ、実用上問題ない程度のデバイス数といえる5デバイスはサポートできますので、比較的小規模の環境でも、十分 USB ホストシステムを構築できることがわかると思います。

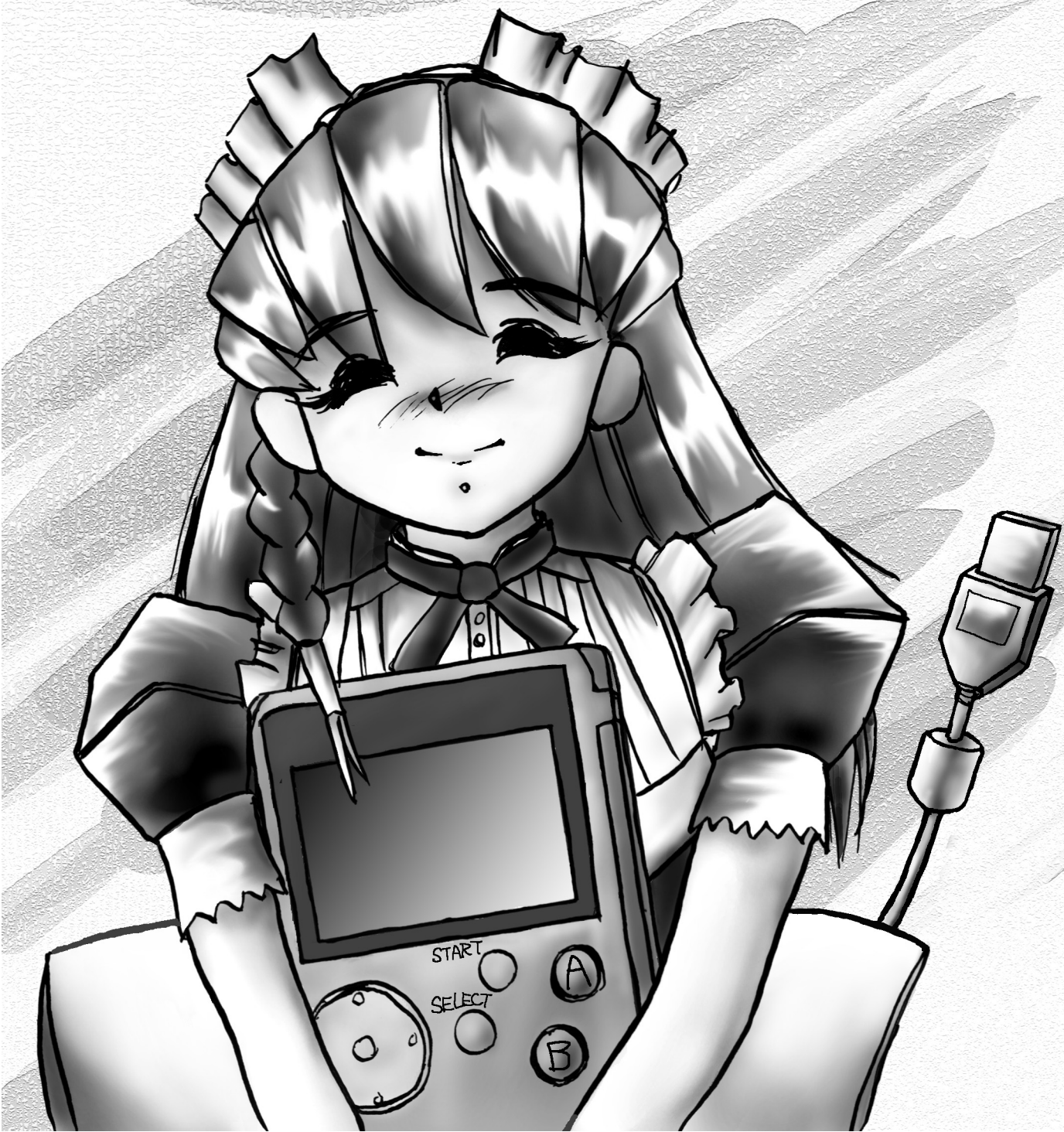
一応、今回はハブの制御、デバイス挿抜の自動認識など、一通りの USB の基本機能を実装してみました。USB の基本的なパケットの動作さえ理解していれば、あとは USB 規格書とアプリケーションノートのサンプルコードを読むだけで十分設計可能でしたので、単に動かすだけではあまり難しい部類には入らないものかと思います。

もちろん Windows のように幅広いデバイスをサポートしようとすると大変でしょうが、限定されたデバイスであれば十分対応可能かと思います。最近はいろいろと魅力的な USB デバイスも増えていますので、そのようなものをインターフェイスとしてマイコンに接続して使用する場合には、1つのインターフェイスで様々なデバイスに対応できますので、十分試してみる価値のあるものと言えるでしょう。

そのようなシステムを設計する際に、この原稿が一助となれば幸いです。なお、ファームウェアなどは下記の URL にて公開していますので、そちらからダウンロードして頂くようお願いします。

URL: http://homepage2.nifty.com/~maid/hardware.html

P/ECEでUSBはいかがですか？



はじめに

P/ECE は Leaf/AquaPlus が 11 月 30 日に発売した携帯ゲーム機です。中途半端なスペック(笑)といまいち不完全なカーネルなど不満はたくさんあるハードウェアですが、それらすべてを駆逐する大きな魅力があります。それは、カーネルを含めたすべてのソフトが公開されており、自分ですべてを書き換え可能であること。さらにはソフトウェアだけでなく、ハードウェアもすべて公開されていることです。

これにより、ソフトに不満があれば自分でカーネルを書き直すこともできますし、最悪ハードがアレなら自分で直すこともできます。まさに自作という言葉に魅力を感じる人には究極のハードウェアです。

今回、USB ホストシステムとしてどすぶいだにょ☆２号の方に、基本的な USB ホストシステムの設計例を解説しましたが、このコピー誌ではそれを応用して、コミケで実演したスレーブデバイスである P/ECE を設計したホストシステムと組み合わせて、USB デバイスのスレーブ－スレーブ間通信の実装方法について、簡単に解説したいと思います。

使用している USB コントローラ

P/ECE の使用している USB コントローラは Philips の PDIUSBD12 というもので、エンドポイントを３つまでサポートしているものです。このうち２本は、アイソクロナス転送にも使用することが可能になっています。P/ECE では一般の通信にはエンドポイント２のみを使用し、ゲームパッドモードのときにエンドポイント１をパッドデータ転送のためにインタラプト転送用に使用します。エンドポイント０は単に PC との接続処理のみに使っているようで、ベンダユニークコマンドなどを使って制御しているわけではないようです。個人的にはそっちを使った方がちゃんと制御できると思うのですけど(^^;;;

このコントローラで特徴的なのは、内部にトランシーバだけでなく D+プルアップ機構を持っていることで、Philips では SoftConnect と呼んでいるようです。これにより、一般のコントローラで接続制御に必要なスイッチ付きのプルアップ回路を外付けで持たずに、ホストとの回線接続・切断を簡単に制御できるようになっています。

P/ECE の USB 入出力周りの構造

P/ECE の USB 入出力処理は、カーネルのメインループに組み込まれています。意図的に切断していない限りは、USB の割り込みルーチンが常にコマンドを受け付けるようになっており、USB ポートからなんらかのコマンドを PC から受信すると、カーネルはそれを検出して、そのコマンドに従い USB 通信処理を随時実行するようになっています。

エンドポイントの動作

P/ECE のコンフィグレーションデスクリプタ、デバイスデスクリプタ、エンドポイントデスクリプタは**表 1** のようになっています。ちょっと普通の USB デバイスと違うパラメータで、デバイスデスクリプタのエンドポイント 0 の最大パケットサイズが、一般的なスレーブデバイスでは８であるところ 16 になっていたりします。さらにエンドポイントデスクリプタでは、本来 USB 規格外のはずである(^^;、同じエンドポイントに対して２つの定義があるので、実際の所ちょっとパラメータ的には問題があったりします。そのため、ホスト側で P/ECE のこれらのエンドポイントを認識する際には、このようなイリーガルな設定もきちんと通さなければなりません。

通常は**表 1** にあるように、P/ECE はエンドポイント 2 をバルク転送の送受信に使います。エンドポイント１はパッドのインタラプト転送用に予約されているようで、ゲームパッドモードで起動すると USB の再接続処理を行った後に、エンドポイント１をパッドのインタラプト転送用に使用するデスクリプタをホスト側に送信しています。エンドポイント０に関しては、単に接続時のネゴシエーションにしか使っていないようです。もったいない(^^;;

このエンドポイント２は前述のように、通常はホスト→スレーブ状態になっており、P/ECE は USB に接続されるとバルク転送で送られるデータを割り込みルーチンで受信待機している状態となっています。ここでコマンドをバルク転送すると、そのコマンドに従い P/ECE で定義される各種 USB 通信トランザクションに入ります。

USB 通信で使われるコマンド

P/ECE の通信で使用されるコマンド（カーネル Ver.1.14）は**表 2** のようになっています。なんとなくつぎはぎで足した感は否めませんが(^^; これらのコマンドをパケット構造で示す形式にしてバルク転送で送ります(このステートを**コマンドステート**と本稿では称します)。そうすると、PC→P/ECE での転送処理要求のパケットを送ったのであれば、続けてそのコマンドで必要な分だけのデータを PC は送信し、逆であればエンドポイント２の状態をリセットし、P/ECE→PC に切り替えて PC は受信処理に入ります(このステートを本稿では**データステート**と称します)。

表1 P/ECE の通常通信時に使用されるデスクリプタ

P/ECE デバイスデスクリプタ

メンバ名	値
bLength	0x12
bDesctiptorType	0x01(DEVICE)
bcdUSB	0x0110(USB 1.1)
bDeviceClass	0 (NONE)
bDeviceSubClass	0 (NONE)
bDeviceProtocol	0 (NONE)
bMaxPacketSize0	0x10
idVendor	0x0e19(OeRSTED, Inc.)
idProduct	0x1000
bcdDevice	0x0100(V1.00)
iManufacturer	1 (StringDesc:"OeRSTED, Inc.")
iProduct	2 (StringDesc:"PIECE PME-001")
iSerialNumber	0 (NONE)
bNumConfigurations	1

P/ECE コンフィグレーションデスクリプタ

メンバ名	値
bLength	0x09
bDescriptorType	0x02(CONFIGURATION)
wTotalLength	0x20
bNumInterfaces	1
bConfigurationValue	1
iConfiguration	4 (StringDesc:"PieceConfig")
bmAttributes	0x80 (No Attributes)
bMaxPower	50 (100mA)

※本来、同じエンドポイントに 2 つの属性は付けてはいけないので、良い子はマネをしないようにしましょう(ˆˆ;

P/ECE インターフェイスデスクリプタ

メンバ名	値
bLength	0x09
bDescriptorType	0x04(Interface)
bInterfaceNumber	0
bAlternateSetting	0
bNumEndpoints	2 (ENDPOINTS=2)
bInterfaceClass	0 (NONE)
bInterfaceSubClass	0 (NONE)
bInterfaceProtocol	0 (NOT USE)
iInterface	5 (StringDesc:"PieceInterface")

P/ECE エンドポイントデスクリプタ(出力用)

メンバ名	値
bLength	0x07
bDescriptorType	0x05 (Endpoint)
bEndpointAddress	0x02 (OUT-ENDPOINT2)
bmAttributes	0x02 (Bulk Transfer)
wMaxPacketSize	0x40 (64)
bInterval	0 (No Interval)

P/ECE エンドポイントデスクリプタ(入力用)

メンバ名	値
bLength	0x07
bDescriptorType	0x05 (Endpoint)
bEndpointAddress	0x82 (IN-ENDPOINT2)
bmAttributes	0x02 (Bulk Transfer)
wMaxPacketSize	0x40 (64)
bInterval	0 (No Interval)

このような USB のコマンドには、一般的な USB 通信を行うコマンドだけでなく、P/ECE 内部の RTC の設定を行うことで、PC 側の時計と同期する機能、さらにデバッグ用にメモリ内にプログラムを展開してそれを実行する機能など、結構面白く便利な機能が多くあります。これらは、カーネルに機能が追加されるたびに増えているようですので、初期リリース版に含まれていない機能(Set RTC 以降)を使う場合には、通信を行う前にカーネルのバージョン情報をチェックして、自分の行おうとする通信方式をカーネルがサポートしているかチェックをしておく必要が絶対にあるでしょう。

なお今回は、USBCOM のみを使用していますので、このようなチェックは行っていません(^^;

USBCOM

USBCOM は P/ECE の API の中で、USB 汎用通信を実現できるもので、**表 2** では USB コマンド 10～14 に関係するものです。これらのコマンドは、以下に解説する P/ECE の API により基本的に制御されますが、**表 2** のコマンドのうち USBCOM Open と USBCOM Close は、あくまで PC 側からみた USBCOM の状態の設定ですので、API とは分けて考えて上げる必要があります。つまり、API の pceUSBCOMSetup()はあくまで P/ECE 側から実行したものですので、PC 側から USBCOM Open をしたのであれば、必ず Close も PC 側から行う必要があります。

表２　P/ECE USB ファンクション表(カーネル Ver.1.14)

※パケットの CMD にはコマンド番号が入ります

ｺﾏﾝﾄﾞ番号	機能名	パケット構造	実行内容
0	Version	[CMD(1)]または、[CMD(1)][LEN(1)]	システム情報(system_info)を取得する。コマンドのみではすべての情報(sizeof(system_info))を、コマンド＋長さで送ると、長さ分の情報を取得できます。
1	Execute	[CMD(1)][ADDRESS(4)]	指定のアドレスから実行を開始する
2	Read Cont.	[CMD(1)][ADDRESS(4)][LENGTH(4)]	指定のアドレスからサイズ分だけ P/ECE から PC へ送信する。
3	Write Cont.	[CMD(1)][ADDRESS(4)][LENGTH(4)]	指定のアドレスへサイズ分、PC から P/ECE へデータを送信する。
4	Set App Status	[CMD(1)][STATUS(1)]	アプリケーションステータス（カーネルの appstat 変数）を設定します。設定できる値は 1 または 3 で、状態により設定できるときとできないときがあります。
5	Get App Status	[CMD(1)]	アプリケーションステータスを 1 バイトを取得します
6	Get Wave Buff Status	[CMD(1)][CH(1)]	指定のチャネルの WAVE データバッファの個数を取得します
7	Out Wave Data	[CMD(1)][CH(1)] [PCEWWAVEINFO(4)]	指定のチャネルから、指定のアドレスにある PCEWAVEINFO 構造体の指定する WAVE データを出力します。
8	Flash Erase	[CMD(1)][ADDRESS(4)]	指定のアドレスのフラッシュメモリのセクタを消去します(範囲は 0x02000h～0x7F000 から開始するセクタ)
9	Flash Write	[CMD(1)][ROM ADDRESS(4)] [ADDRESS(4)][LENGTH(4)]	指定の ROM アドレスに指定されたデータを指定バイト数分書き込みます。ただし、フラッシュメモリへはワードライトアクセスですので基本的に偶数バイトでなければなりません(内部で書き込み長を 2 で割ります)
10	USBCOM Read	[CMD(1)]	PC→P/ECE の転送を開始します。 読み出し位置やサイズは pceUSBCOMStartRx() で指定されたものが使われます。
11	USBCOM Write	[CMD(1)]	P/ECE→PC の転送を開始します。 転送開始位置やサイズは pceUSBCOMStartTx() で指定されたものが使われます。
12	USBCOM Get Status	[CMD(1)]	USBCOM 関係の全ステータス(12 バイト)を取得します。データは順に受信ステータス(1)、送信ステータス(1)、P/ECE 側の pceUSBCOMSetup 実行フラグ(1)、PC 側の USBOPEN フラグ(1)、受信要求中データサイズ(4)、送信要求中データサイズ(4)となっています。
13	USBCOM Open	[CMD(1)]	PC 側より USBCOM 使用要求を出します。これを行った後は USBCOM Close を行うまで、P/ECE 側も pceUSBCOMStop()ができなくなります。このとき取得されるデータ 16 バイトは USBCOM Get Status で取得されるデータと、現在 P/ECE 側で pceUSBCOMSetup を実行したときに設定したシグネチャが取得できます
14	USBCOM Close	[CMD(1)]	PC 側の USBCOM 使用を中止します
15	Set RTC	[CMD(1)][PCETIME(9)]	送信された時間データを使って、P/ECE の時間を設定します。
16	App Pause	[CMD(1)][ONOFF(1)]	ポーズステータスのフラグの設定・解除を行います。
17	LCD Get Info	[CMD(1)]	LCD のサイズ、バッファアドレスなどの情報を取得します。取得されるデータは 12 バイトで順に、ポーズ状態(1)、リザーブ(1)、X サイズ(2)、Y サイズ(2)、リザーブ(1)、バッファアドレス(4)となります。
18	Get Heap Top	[CMD(1)]	ヒープのトップアドレスを取得します。4 バイトのアドレス情報が取得されます。

USB ポートをオープンする

void pceUSBCOMSetup(USBCOMINFO *puci)

USB の内部管理情報を初期化します。このときの情報でアプリケーション名は、PC が USB ファンクション 13 を呼び出したときに取得できます。このため、PC 側は自分の目的としないアプリケーションが PC と接続しようとしたときに、すぐに拒否してクローズをすることができるようになっています。

他にこの API は、内部 USB API 使用許可フラグを Enable にします。もしそのフラグが Disable のときは pceUSBCOMStartRx() などの呼び出しが無効にされますので、必ず USB 送受信 API を実行する前に実行する必要があります。

USB ポートをクローズする

int pceUSBCOMClose(void)

内部 USB API 使用許可フラグを Disable し、USB 内部管理情報をクリアします。ただし、ファンクション 13 を使って P/ECE の USB ポートを PC からオープンしている場合には、PC 側からファンクション 14 を使ってクローズ処理をしない限りはこの関数は正常終了できません（クローズできません）。

データの受信を開始する

void pceUSBCOMStartRx(
unsigned char *pData, //データバッファのポインタ
int len // データ長
)

pceUSBCOMSetup()が実行済みであれば、USB 受信ステータスフラグを UCS_RXWAIT にし、データ長とバッファポインタを USBCOM API 内部の受信パラメータにいったんセットします。この状態で PC 側から USBCOM Read コマンドが発行されると、USB 割り込みルーチン内の変数に、API でセットした受信パラメータの値がセットされ、それ以降は PC から送信されたエンドポイント 2 へのバルク転送データは、それらのパラメータで設定された位置に指定されたデータ数分を受信するまで格納する処理が行われます。

なお P/ECE の場合、基本的に pceUSBDisconnect()してない限りは、常に受信状態になっていますので、受信割り込みが入り、データ受け入れモードに入っていれば、コマンドではなくデータとしてすべて扱われます。このような状態になったときに転送をリセットしたい場合には USB バスリセットを発行するか、USBCOM を PC、P/ECE 共にクローズして、再度オープンし直すしかないようです。

受信が完了すると、pceUSBCOMGetStat() によって得られるステータス値に UCS_RXDONE フラグが立ちます。

指定のデータの送信を開始する

void pceUSBCOMStartTx(
unsigned char *pData, // データ・バッファのポインタ
int len // データ長
)

pceUSBCOMSetup()が実行済みであれば、USB 送信ステータスフラグを UCS_TXWAIT にし、送信データ数と読み出しポインタをセットします。これ以降に PC 側へエンドポイント 2 を使って、データをバルク転送で指定データ数分を送信します。 P/ECE の USB 割り込み処理の基本的な手順としては**図 1** のようになっています。送信完了割り込みで送信処理がループするようになっています。

まず、割り込みステータスフラグをクリアした後、エンドポイント 2 にペイロードサイズ 64 に合わせ、

図 1　P/ECE の USB 割り込み処理のフローチャート

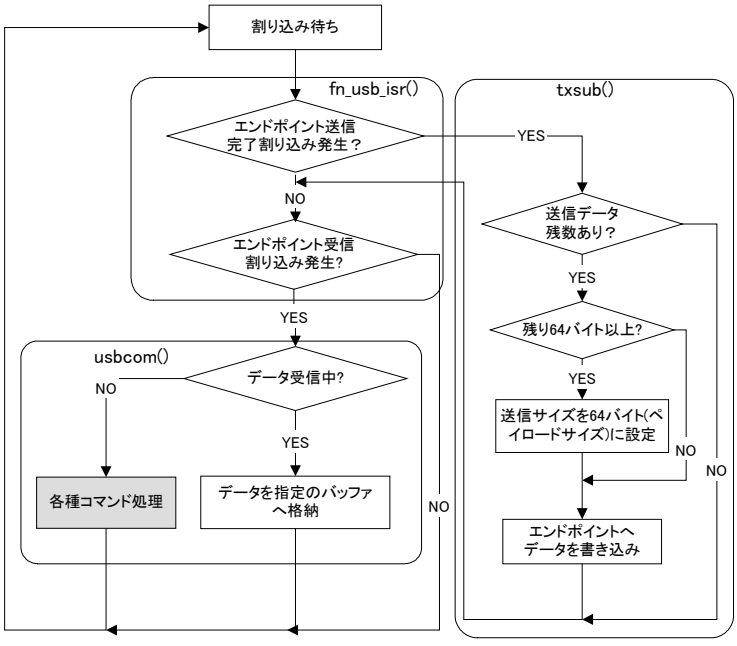
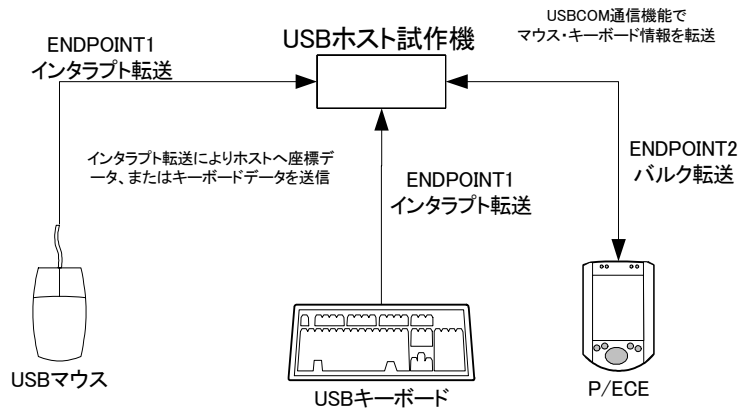


図 2　デモの通信路のイメージ



いちきちんと切れないので Disconnect のつもりもホストは関知してしまうようです(^^; ちゃんと外部でやらないと駄目かも。

コントローラへのクロックなどの供給は止めたり、PLL を止めたりしないようですので、実は P/ECE のオフ時電流の大きい原因は、ここかもしれません（ちゃんと調べてませんが）。

最大 64 バイトが USB コントローラの FIFO にセットされます。あとは送信完了時に送信完了割り込みが発生しますので、その割り込みの際にまた送信サブルーチン(txsub())を呼び出します。もし残りデータがあれば、また送信データがセットされます。なければセットされず、送信完了割り込みも終止します。

設定されたデータの全ての転送が完了すると、pceUSBCOMGetStat() によって得られるステータス値に UCS_TXDONE フラグが立ちます。

USB 通信状況を取得します

int pceUSBCOMGetStat(void)

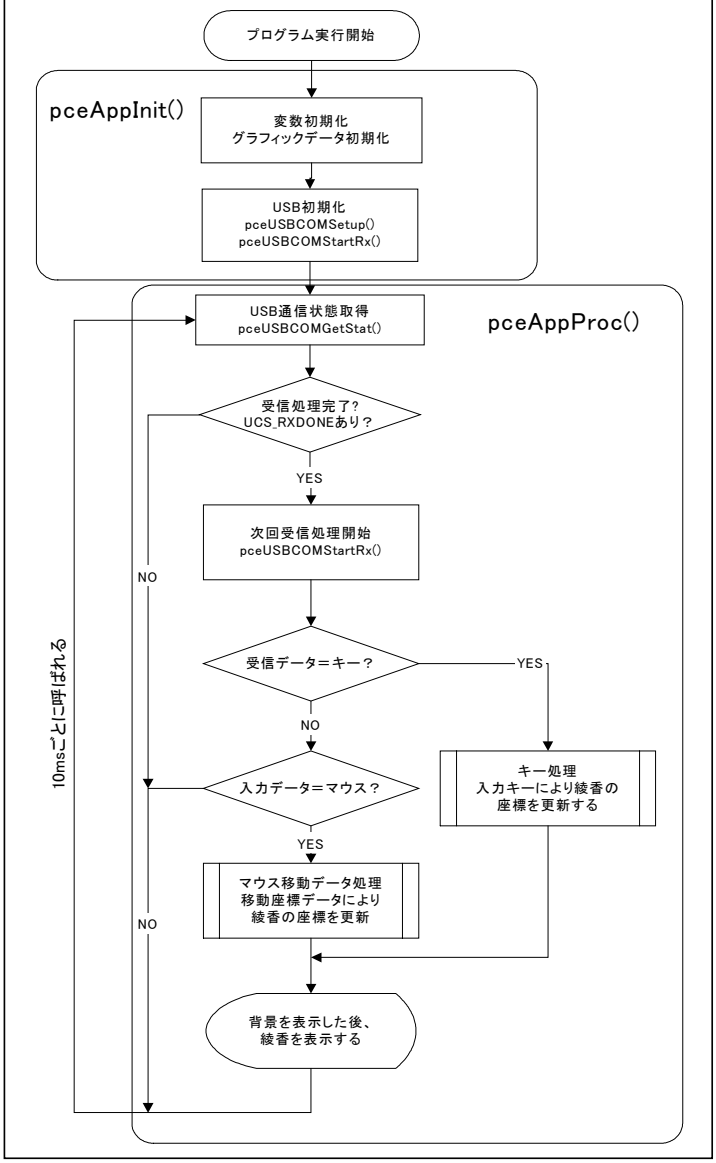
現在の USBCOM の転送状態を取得します。主に pceUSBCOMStartTx()、pceUSBCOMStartRx()を実行して、送信、受信処理をスタートした後に、それらの処理が実際に終了を検知するのに使用します。送信が終了したのであれば UCS_TXDONE が、受信終了であれば UCS_RXDONE が立ちます。一度これを呼ぶと、UCS_TXDONE、UCS_RXDONE フラグはクリアされるので注意する必要があります。

USB を切断します

void pceUSBDisconnect(void)

PC との接続を解除します。具体的には、USB コントローラの内蔵するトランシーバに入っている D+ のプルアップ(SoftConnect)を解除していますが、いま

図 3　綾香たんてくてく☆のフローチャート



USB を(再)接続します

void pceUSBReconnect(void)

PC との接続を許可します。エンドポイント１と２の状態をリセットし、プルアップ(SoftConnect)をON にします。再接続すると、PC は P/ECE を新しいデバイスとして認識し、ネゴシエーション処理に入ります。

USB の動作モードを変更します

```
int pceUSBSetupMode(
    int mode,          // モード
    void *param2,  // 予約 必ず NULL(0)
    void *param3   // 予約 必ず NULL(0)
  )
```

P/ECE の USB 接続モードを通常モード、またはゲームパッドモードに切り替えます。ただし、この API は必ず Disconnect 状態で実行する必要があります。なぜなら、処理的には内部のモードフラグを切り替えているだけだからです。

この設定を切り替えた後に pcdUSBReconnect()が実行されると、通常モードの場合には P/ECE 固有デバイスとして認識されるようなデスクリプタをネゴシエーション時にホストへ送信しますが、パッドモードでは HID ゲームパッドデバイスとしてのデバイスデスクリプタを送信し、P/ECE 自身もそのように振る舞います。

このようにして PC への P/ECE のデバイスとしての認識のされかたを制御しています。

P/ECE に外部 USB デバイスを接続する

どすぶいだにょ☆本誌で解説したように、USB はホスト主導のバスであるため、必ずスレーブデバイスを制御するためにはホスト機能をもつデバイスが必要になります[1] 。PC に USB スレーブデバイスを接続する場合には、PC 自身がホストとなるため、スレーブデバイスへの通信は特に支障なく行うことが出来ます。しかし、PC がない場合にスレーブデバイスに対してスレーブデバイスを接続する場合には、**図 2** のように中間にホストを接続して、このホストが制御側となるスレーブデバイスとターゲットとなるデバイスの仲介を行う必要があります。今回は、このホスト部分をマイコンを使って、キーボードとマウスを制御するデバイスを試作しましたので、さらに P/ECE←USB ホスト→USB マウス、または USB キーボードというような、別々のスレーブデバイスの間でデータの受け渡しを行うような、PROXY 的な動作をするコードを実装して、デモで行ったように P/ECE が同じスレーブデバイスである USB キーボードや USB マウスを制御しているかのように振る舞わせてみました。

実際には、これらの P/ECE を含むスレーブデバイスはホストがすべて管理をしていますが、ホストは P/ECE と通信して自分の管理するバスの中に P/ECE が組み込まれている状態では、自分の受け取ったマウスやキーボード情報を P/ECE にどんどん投げ込んでいくようになっています。そして P/ECE で、USB 通信を行うプログラムが起動されデータ受信状態になると、これらの投げ込まれたデータを受け取って処理を行うようになっています。

コミケデモ「綾香たんてくてく☆」の仕組み

図 3 にデモのフローチャートを示します。

まず、グラフィック系の初期化を行ったあとに pceUSBCOMSetup()を呼び出し、P/ECE の USB モジュールを初期化して USBCOM を使用可能な状態にし、受信処理をスタートします。さらに、デモソフトでは pceAppProc()を 10ms ごとに呼ぶように設定しています。

今回の USB ホストでは、キーボードやマウスの反応が起きるたびに P/ECE へデータを送りますが、一番速いときには 10ms 程度の間隔で送り出されるようになっています[2]。このため、pceAppProc()のイベント発生で補足する場合には、pceAppProc()の呼び出し間隔も相応にしなければならないため、10ms というかなり P/ECE のアプリケーションの中では高速な動作をさせています。実際には、実行速度を遅くしても P/ECE 側がデータ受信準備できるまで NAK を返し、USB ホスト側でそれが ACK になるまで待つようにしてあげればよいのですが(通常の USB デバイスは一般的にそのような通信を行う)、P/ECE の場合コマンドをバルク転送で投げるのと、**P/ECE のアプリケーションが受信準備できていなくても、ホストに ACK を投げてデータを受けてしまうので（図１）、データ落ちが発生してしまう可能性があるのです。**これを防ぐにはプロトコルを根本的に見直す必要があるのですが、それにはカーネルの修正が必要になるため、単にデモプログラムとして設計したこのプログラムでは、そこまではしませんでした。

---

[1] 最近策定された USB On-The Go という規格では、スレーブ間直結ができるみたいです。

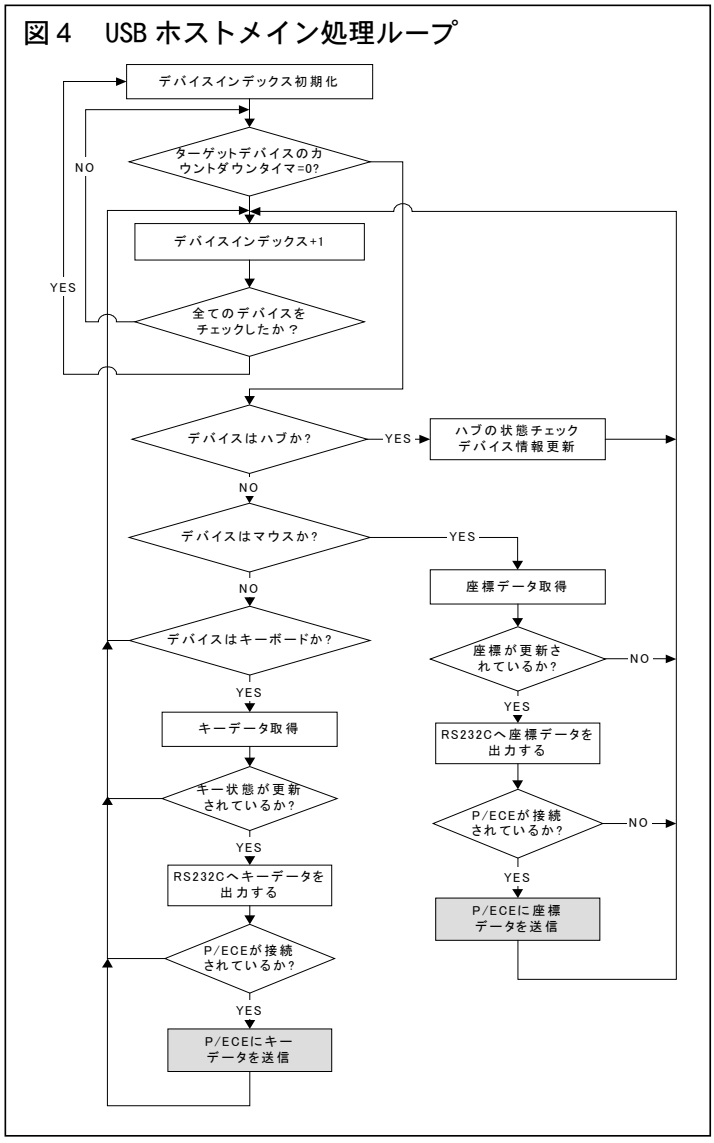
[2] 実際には、インタラプト転送では 1ms で制御していますので、かなり遅くなっていますが、これ以上速くすると P/ECE の処理がついてこないために、このようになっています

リスト１　USB ホスト側の P/ECE への USB データ送信ルーチン

```
; ------------------------------------------------------------------------------------------------
; Send Mouse Data to P/ECE(P/ECE へマウスデータを送信する)
; gr0 : ボタン状態(bit0:Left Button,Bit1:Right Button,Bit2:Middle Button)
; gr1 : X 相対移動座標(char)
; gr2 : Y 相対移動座標(char)
; ------------------------------------------------------------------------------------------------
SendMouseData_to_PIECE:
        lds     gr3,PIECEDEVNUM     ; P/ECE のデバイステーブル番号を取得
        cpi     gr3,0xff            ; P/ECE が存在しない？
        brne    PC+2                ; 存在するのであれば 1 命令先へジャンプ
        ret
        push    YL                  ; 現在のデバイステーブルアドレスを待避
        push    YH
        push    gr0                 ; マウスデータをスタックへ待避
        push    gr1
        push    gr2
        mov     gr0,gr3             ;
        rcall   GetBaseTblAdr       ; P/ECE のデバイス情報テーブルアドレスを取得(Y)
        ldi     ZL,low(DescriptorBuf) ; DescriptorBuf =  送受信用ワークバッファ
        ldi     ZH,high(DescriptorBuf)
        ldi     gr0,11              ; USBCOM Write コマンド設定
        st      Z,gr0               ; 送信バッファへデータを保存
        ldi     gr0,2               ; 出力先(P/ECE)エンドポイントを 2 に指定
        ldi     gr1,1               ; 送信サイズ=1 バイト
        ldi     gr2,PID_OUT         ; OUT トランザクション要求
        rcall   BulkXfer            ; バルク転送の実行(コマンドステート)
        pop     gr2
        pop     gr1
        pop     gr0                 ; 待避していたマウスデータを戻す
        ldi     ZL,low(DescriptorBuf)
        ldi     ZH,high(DescriptorBuf)
        ldi     gr3,0x81            ; マウスデータフラグをセット
        st      Z+,gr3              ;
        st      Z+,gr0              ; ボタン状態をバッファに保存
        st      Z+,gr1              ; X 相対移動アドレスを保存
        st      Z+,gr2              ; Y 相対移動アドレスを保存
        ldi     gr0,2               ; 出力先(P/ECE)エンドポイントを 2 に設定
        ldi     gr1,8               ; 送信データサイズ=8
        ldi     gr2,PID_OUT         ; OUT トランザクション要求
        ldi     ZL,low(DescriptorBuf)
        ldi     ZH,high(DescriptorBuf)
        rcall   BulkXfer            ; バルク転送開始(データステート)
        pop     YH
        pop     YL                  ; デバイステーブルアドレスを戻す

        ret
```

図 4　USB ホストメイン処理ループ



アプリケーションがスタートすると、P/ECE カーネルは初期化ルーチンで設定した 10ms ごとに pceAppProc()を呼び出します。ここで、もしなんらかのデータの受信が完了していれば、pceUSBCOMStat() で取得したステータスに UCS_RXDONE フラグが立っています。これを確認したあと、バッファを別にセットしすぐに受信処理をスタートします。この受信データを格納するバッファをデモプログラムでは 2 面用意しており、このバッファを常に切り替えながら受信しています。これにより CPU が表示処理を行っている最中に USB 受信処理が始まっても、きちんとデータが受信できるようになっています。

USB ホストから送るデータは 8 バイト固定としており、先頭にデータの種別、残り 7 バイトにキー、またはマウスの移動データ、ボタン状態情報が入っています。このデータの種別に対しては、デモプログラムは 0x80 にキーを、0x81 にマウスを割り当てています(リスト 1)。これは、**表 2** のファンクション表を見ればわかるように先頭の 0x10 付近は USB コマンドで使用されているためで、不用意にこれとかぶるデータを送ると、P/ECE がコマンドと誤認する可能性があるためです。このため、データを送る際には必ずパケットの先頭バイトがコマンドとかぶらないようにしておかねばなりません。通常は、USBCOM コマンド実行後のデータ送受信ステートで、データを格納したパケットは受信されるので問題ないはずなのですが、コマンドを読み落とすと危険な状態に陥る可能性がありますので、注意するにこしたことはありません。

ホスト側の実装

ホスト側の実装をリスト 1 に示します。使用している CPU は本誌でも解説したように ATMEL 社の 8 ビット RISC マイコン AVR シリーズ AT90S8515 です。

このリストは実際にデータ送信を行う部分で、バッファにコマンドやデータを設定して、それらのデータをバルク転送によりコマンドステート、データステートと 2 度実行していることになります。

この部分を呼んでいるのは、マウスやキーボードデータのインタラプト転送を行うカウンタアンダーフローを監視するメインループです(**図 4**)。なお、このフローチャートにあるデバイスインデックスとは、各デバイスがどのようなデバイスタイプでどのようなエンドポイントをもつのかなどの情報を格納したテーブルのインデックス番号です。今回設計した USB ホストでは、このインデックスで、すべてのデバイスを管理するようになっています。

ホストが動作しているときは、無限ループ状態で常にカウンタの値をチェックしています。このカウンタはタイマ割り込みにより 1ms ごとに減算され、0 になったところでカウントダウンが止まるようになっています。このカウンタの値をメインループで 0 になったかどうかをチェックし、0 になった時点で該当のデバイスの処理を行います。

ハブであればハブの状態を取得して、ポート状態の更新があればデバイステーブルの更新を行い、マ

ウスであれば座標データの取得を行います。このとき、マウスから NAK、または移動なし(X=Y=0)、ボタン状態変化なしが帰ってきた場合には NOP とし、変化があればそれを RS232C に出力します。さらに P/ECE が接続されていれば、P/ECE に同データを送信します。キーボードも同様に、キー状態に変化があれば RS232C に出力すると共に、P/ECE にもそのキーデータを規定のパケットに組んで USBCOM Write コマンドを使って送り出します。

USB バスに P/ECE が組み込まているかどうかは、新しいデバイスを検出した時にそのデバイスのプロダクト ID とベンダーID が P/ECE のそれと同じかどうかで判断を行います。もし P/ECE であれば、PIECEDEVNUM 変数にそのデバイステーブルインデックスをセットし、もし P/ECE が抜かれた場合には、この変数を初期化します。

どのような感じで動作するか

以上のような感じで実装したものが、コミケでデモを行っていた綾香たんのデモです。このデモは右ボタンを押しながらマウスを動かすと綾香とともに背景も動き、離したままで動かすと綾香だけが動くようなコードになっています。

マウスの座標データの送り出しを 10ms に 1 回に抑えているのと、かなり pceAppProc()のループを高速に回しているため、ちょっとキャラクタを動かすのには重くなっています(^^; やっぱりカーネルを直して、1 トランザクションでデータを送れるようにしないと、この手のものはなかなかうまくいかないようです。いまのままだと、データ落ちでいきなり動かなくなったりする可能性が大きいものですし。もうちょっとベンダユニーク系のコマンドがユーザーで使えれば、自由度が上がるとは思うので、ここらへんも今後のカーネルでサポートしてもらえたらありがたいですね。

ともあれ、外部デバイスで操作できるというのはなかなか楽しいです。USB 規格でも、モバイル向けにスレーブ間接続できる規格が規定されたので、P/ECE の次のバージョンではこれをサポートしてもらえたらいいですね。

最後に

本誌であまり触れなかった、ソフト面を中心に書いてみましたが、いかがでしょうか。

丁度、11 月末に P/ECE が発売されて、大急ぎでこのデモプログラムを作ったのですが、結構数日でこのようなプログラムが書けたので、なかなか気に入っています。P/ECE は簡単で素人さんがプログラムしたい場合にはかなり良いのではないでしょうか。

実は某誌編集さんに P/ECE のことを事前に聞きまくっていて、期待を膨らませつつ発売を待っていた感じだったのです。近年見ない、ハードもソフトも完全公開の製品なので、組み込みプログラミングやハードウェア設計を勉強したい人にもお勧めしたいですね。学生さんにもよい教材となるかもしれません。

取り敢えず、コミケが終わったら今度はハードをばらしていろいろ改造してみるつもりです(^^; 取り敢えずはメモリ増設かな（笑

そんなこんなで、また次回に会う日までごきげんよう～☆

変更履歴

2001年12月30日　コミケット61頒布版（東京ビッグサイト）
表紙フルカラー、80P オフセット誌、頒布価格 500 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*
原稿修正　Word98 → Word2000 ベース修正（ChaN 氏執筆分）
原稿追加　「P/ECE で USB はいかがですか？」
コピー誌配布分（もろぼし☆らむ執筆分）

著者あとがき(50音順)

おゅぅさん
お初にお目にかかります．豪華ライター陣の中で執筆させていただけました．
この機会をいただけたChan氏,もろぼし☆らむ氏,そして読んでいただけた諸氏に感謝．皆で電子工作の年期を積み上げていきましょう(^^)/

きよりんさん
リサイクルとか資源を大切にしようと言いながら、高額な修理代の前に泣く泣くスクラップにされる製品群。そろそろだまされ続けることはやめにしようじゃないですか。
この不景気が良いチャンスになると思うんですがねえ。

ChaNさん
今回の企画の言い出しっぺです。なんか、創刊号でファイナルになるはずだったのになぜか2号(^^;。サークル「どすぶいだにょ☆」は気まぐれなので、次はいつになるか分かりませんが、これからも何か機会があったら出して行きたいと思います。

ROM男さん
ROM男です。結構がんばったつもりですが悲劇的な結末を迎えてしまいました（苦笑）。
歳なのかなぁ？　次にやることは…
1. とにかくユニットを動かすこと
2. これ以降は大電力を扱う分野には手を出さない
でしょうか。ハハハ（涙）

もろぼし☆らむ
本当は去年限りのはずだったのに…。なぜかまた今年も出てしまいました(^^;　しかも表紙フルカラーだったり。うーん(^^;;;;;
USBネタは、もうすっかり手垢がついているのでどうかなと思いましたが、まぁホスト側の話は楽しめるかと思って書いてみました。丁度、LeafからP/ECEが出たので、この原稿を上げたらコミケまでに、これを使ったUSBのデモを作るつもりですが間に合うかな?(^^;(現在進行形)

奥付

発行	どすぶいだにょ☆
発行年月日	2001年12月30日【コミケット61配布版】 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61配布版】
著者	ELM BBS 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/

～でんぱ