見本

応用編

第4章
拡張ハードウェアの製作とコントロール事例
…VisualBasicからUSB汎用インターフェースを操作する

今までUSB機器を自作するには，USBターゲットのファームウェアやドライバを書かなくてはならないのが普通でした．ちょっとした機器を接続する程度のことであってもUSB規格やデバイスの使い方の理解，開発環境の整備など何かと面倒が多く，二の足を踏まれていた方も多いのではないかと思います．

本キットでは，ファームウェアEzFirm/FX2の搭載されたUCT-203ボードとサイプレス社のドライバを利用することで，VisualBasicなどからも簡単にUSB経由でのデータ転送やI/O操作を行うことができるようになっています．また，EzFirm/FX2はUSB2.0のハイスピードに対応したデバイスであるサイプレス社のEZ-USB/FX2シリーズに対応しています．EzFirm/FX2は単純なポートのリード/ライトだけではなく，480Mbpsのハイスピード・モードを生かすことができる高速データ転送モードにも対応していますので，大容量データの転送が必要な用途でも充分に役立つことでしょう．

ここでは実際に，本キットのUSB汎用インターフェース・ボードUCT-203に簡単な周辺回路を接続し，VisualBasicのプログラムによってパソコンから制御する事例を紹介します．

4-1　早押し判定器…PIOモードの利用例

EzFirm/FX2のPIOモードでの入出力を行う例として，イベントなどでよく行われる早押しクイズ用の判定器を製作してみました．スイッチの状態を読み出して，どれか一つでも押されたらチャイムを鳴らすとともに，先に押した人の手元のランプを点滅させます．

●部材と加工

スイッチやランプ，チャイムなどは100円ショップで売っているものを利用しました．実験中のようすを**写真1**に示します．購入したチャイムは，スイッチとコードが付いたものです．音量も大きくなかなか良いのですが，呼び出し用であることから「ピーポー・ピーポー・ピーポー」と3回鳴ってしまいます．実際のイベントに使うのには少々間が抜けていますので，チャイム用の回路は別に製作したほうがよいかもしれません．

購入したランプは，荷物に付けて夜間に安全確保するためのものとして作られたようですが，明るく点滅するので都合が良いため使ってみることにしました．スイッチを一度押すと自動的に点滅し，もう一度押すと消灯し

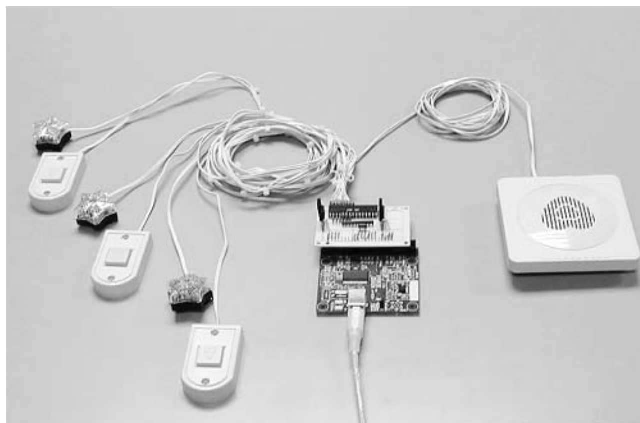

写真1　製作した「早押し判定器」で実験中のようす

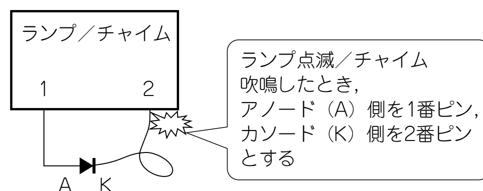

図1　電池内蔵のランプやチャイムのスイッチの極性を調べる

ます．プラスチックのケースがレンズのようになっていて，正面から見ると少々目が痛い程度に明るいので，教室でのイベント程度ならそのまま使えそうです．

使用したチャイム，ランプとも電池を内蔵しており，無極性ではありませんので，**図1**のようにダイオードを付けて極性を確認しました．ダイオードを付けて作動したとき（スイッチがON状態になるとき），ダイオードのアノード（A）側が付いているほうを1番ピン，カソード側が付いているほうを2番ピンとしました．したがって，ONのときに1番ピンから2番ピンの方向に電流が流れます．

ブザーはスイッチ用にケーブルが出ているものでしたが，ランプのほうには出ていませんので，1回分解して，接点のパターンにはんだ付けしてビニール電線を引き出しました．中をあけるとスイッチの接点部分で平行して走っている曲がりくねったパターンがありましたので，スイッチの接点用の導電ゴムと干渉しない位置から線を引き出しました．導電ゴムと干渉すると手動で消灯させることができなくなってしまいます．

チャイムとスイッチの間の線が長いので，途中で切ってランプ用のケーブルとしてそのまま流用して，スイッチとランプのペアで結束バンドでまとめました．ケーブルには極性の目印を付けておくとよいでしょう．

●制御対象の回路

回路を**図2**に，試作した制御ボードの外観を**写真2**に示します．UCT-203ボードには50ピン（25ピン×2列）のソケット・タイプ（オス型）のピン・ヘッダを取り付けています．制御ボード側は40ピンの角型ピン・ヘッダを取り付けて，UCT-203と接続しています．

今回は3人用ですが，8人まで対応可能にしてあります（R_3でプルアップしている部分）．下部の点線で囲った部分は，次節のモータ制御の例で同じ基板を流用するためために付加したものです．早押し判定器では使用していませんので，省略してもかまいません．

スイッチ入力はプルアップしておいて，スイッチが押されるとLレベルになるようにしました．チャイムやライトは接点がONしたときにチャイムが鳴り，点滅を始めるようになっているため，FETやダイオードを使ってLレベルを出力したときにON，HレベルのときはOFF（ハイ・インピーダンス）状態になるようにしています．

ダイオードの許容電流や順方向降下電圧が気になるところですが，実測してみるとランプの接点電流はごく小さかったことや，ダイオードの順方向降下電圧程度なら動作上の問題にはならないようなので，簡単にすませました．チャイム側はFETを利用したスイッチにしています．こちらも実測した限りではダイオードだけでもよさそうなのですが，チャイムの種類によってはもう少し大きな電流を流す可能性もあることと，モータ制御のほうにそのまま流用できるようにと考えて，FETによるバッファを付けることにしました．2SK2956は低電圧で動作可能なスイッチング用のFETです．筆者は秋葉原の若松通商で入手しました．ゲート-ソース間電圧は4V以上が推奨ですが，今回のような使いかたならば2～2.5V程度の電圧でも動作可能です．

ポートの使用方法は次のようにしました．

　PORTA（PA）：スイッチ入力
　PORTB（PB）：チャイム出力
　PORTD（PD）：ランプ出力

ビット割り付けはPA［0..2］をスイッチ入力，PB［0］をチャイム出力（‘1’でON）にして，PD［0..2］をランプ用出力（‘0’でON）にしています．PAの空き端子がフラフラして誤認識することを避けるため，使用してないところもプルアップしておきます．

【注】UCT-203ボードと接続するコネクタ部分の配線には十分に注意してください．はんだブリッジや誤接続があると，最悪の場合にはUCT-203ボードを破損させてしまいます．配線をよく確認してから接続してください．

●プログラム

VisualBasicで作成したアプリケーション画面は**図3**のようなものです（VisualBasicのソース・ファイルは付属CD-ROMのEzJudgeフォルダに収録）．8個のチェック・ボックスでスイッチが押されたときの状態を示します．今回の工作では入力は3個しか使っていませんが，ソフトウェアはPA0～PA7までの8ビットぶんの入力に対応させています．

スイッチが押されると，チェック・ボックスに状態をセットして，勝った人のランプとチャイムにパルス出力を行います．これでチャイムが鳴り，勝った人のランプが点滅するわけです．今回はランプ側に単体で点滅するものを使いましたので，ソフトウェアでの点滅動作はさせていません．

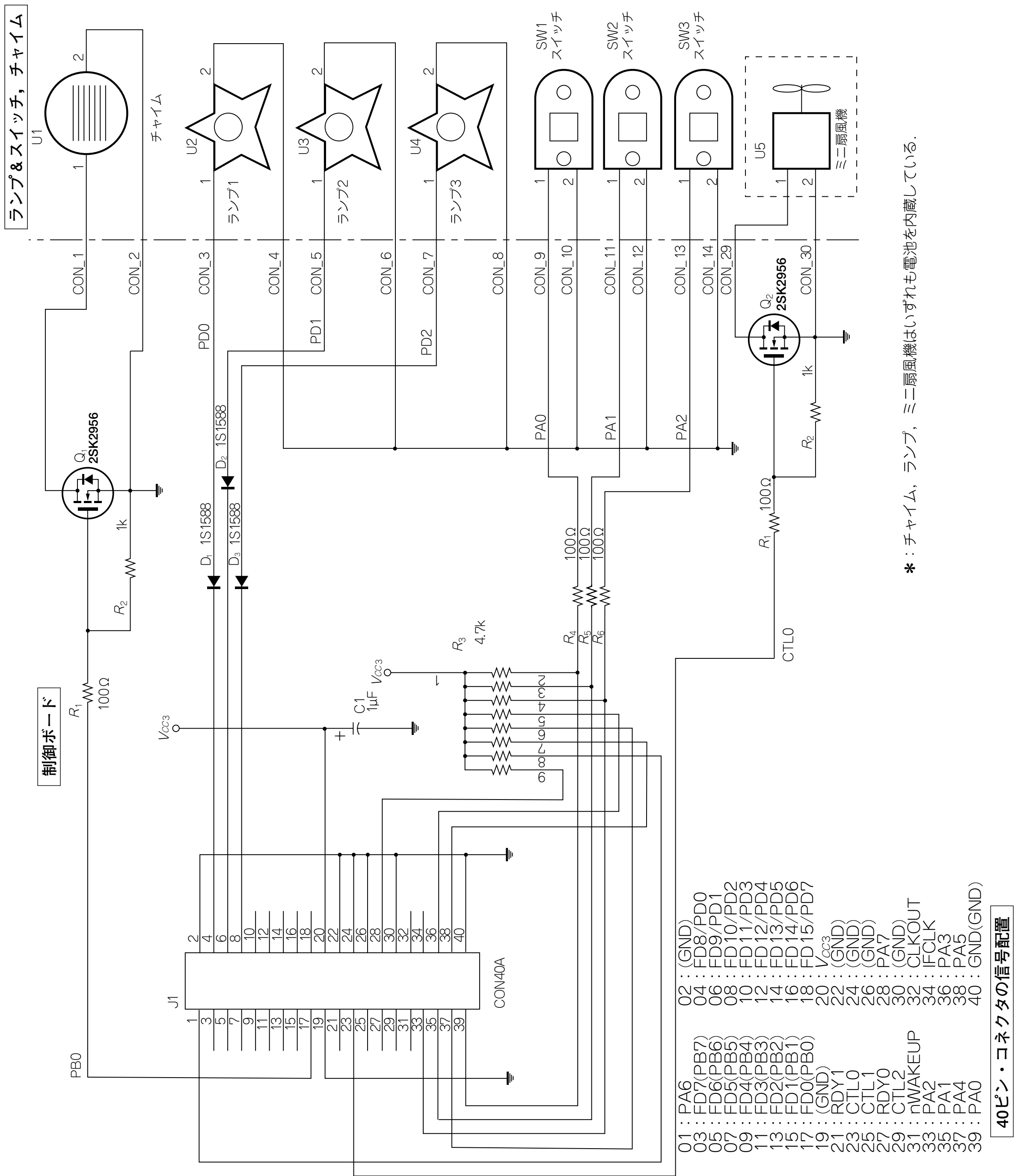

図2　早押し判定器の回路

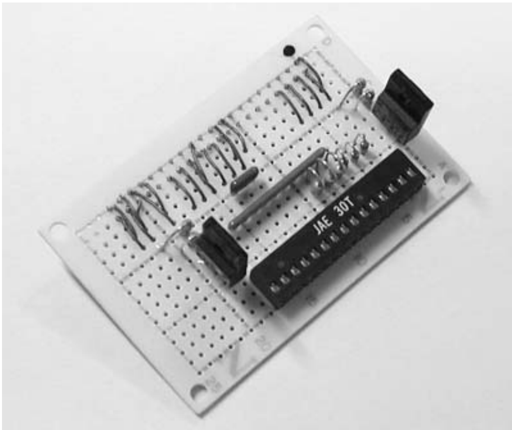

写真2　試作した制御ボード

左上部の裏側に40ピンの角型ピン・ヘッダを取り付けてある．右下部のコネクタはランプ&スイッチ，チャイムを接続するためのもの．

図3　作成したアプリケーションの画面

［RESET］ボタンは，勝利判定が終わったあとで，ランプを消すために点滅中のランプに対してもう一度パルスを与えるとともに，チェック・ボックスをクリアし，次のスイッチ入力を待つようにするものです．

［LEDTEST］，［BUZZTEST］のボタンはそれぞれ動作確認用に用意しました．［LEDTEST］が押されると全部のランプ出力にパルスを出力しますので，すべて一斉に点滅が始まります．もう一度押すと全部消灯になります．今回使用したランプの場合，パルス出力で点滅状態と消灯状態が切り替わるだけですので，もし何らかの理由で一つだけ状態が逆転してしまったような場合には，個別にスイッチを押して状態を合わせておきます．

早押し判定器のプログラムの主要部分は**リスト1**（EzJudge.frm）のようなものです．まず，`Form_Load()`で，ポートの初期値を`EZ_PIOWrite()`で設定したあとに，`EZ_SetPortConfig()`を使ってPIOモードに設定します（`EZ_**`はEZCTL.BAS内の関数）．`EZ_PIOWrite()`をあとにしたほうが自然なのですが，この時点で書き込んだデータは出力されないもののFX2の内部ラッチには取り込まれるため，先に設定しておくことでモード設定直後に期待する初期値と反対のデータが出ないようにすることができるのです．

スイッチの読み込みはタイマ・イベント（`Timer1_Timer()`）の中で行っています．読み込み動作は`EZ_PIORead()`を利用します．16ビット・データが返されるのですが，実際に有効なのは下位8ビットだけです．スイッチ入力は通常は‘1’で，押されると‘0’になるような回路になっているので，わかりやすさのため，入力されたデータをいったん反転してから利用することにしました．

判定ループの中ではビットを下からスキャンしていき，押されている（入力データが‘0’で，反転して判定しているので判定部分では‘1’）になっているビットがあれば，そのビット位置を変数`Winner`に記録します．ここでループを抜けるようにすれば，下位ビットに接続されている側の人の勝ちになります．

このプログラムでは‘1’のビットを見つけてもさらに上のビットまで判定にいきますので，もし上のビットにも‘1’になっているものがあれば，そちらが`Winner`に記録されます．これにより，同時押しされた場合には上

リスト1　早押し判定器のプログラム（EzJudge.frmの一部）

```
'=================================================================
'=  EzFirm/FX2ボード応用例                                       =
'=  早押し判定機コントロールプログラム                            =
'=                                                               =
'= PB[0]  : ブザースイッチ（通常'L'、100msだけ'H'になるパルス）   =
'= PD[0..7]：LEDスイッチ（通常'H'、100msだけ'L'になるパルス）    =
'= PA[0..7]：スイッチ入力（通常'H'、スイッチが押されると'L'）     =
'=================================================================
Option Explicit
Dim hUSB As Long
Dim sts As Long
Dim PBDAT As Byte
Dim PDDAT As Byte
Dim SwitchData As Byte
Dim Winner As Integer

' ブザーテスト
Private Sub CMD_BUZZTEST_Click()
    PBDAT = &H1
    sts = EZ_PIOWrite(1, PBDAT)
    Sleep (100)
    PBDAT = 0
    sts = EZ_PIOWrite(1, PBDAT)
End Sub

Private Sub CMD_END_Click()
    CloseDriver (hUSB)
    End

End Sub

' LEDテスト
Private Sub CMD_LEDTEST_Click()
    PDDAT = &H0
    sts = EZ_PIOWrite(3, PDDAT)
    Sleep (100)
    PDDAT = &HFF
    sts = EZ_PIOWrite(3, PDDAT)
End Sub

' 勝利判定後のLED消灯とフラグ消去
Private Sub CMD_RESET_Click()
    Dim i As Byte
    For i = 0 To 7
        Chk_SW0(i) = 0
    Next
    sts = EZ_PIOWrite(3, Winner Xor &HFF)
    Sleep (100)
```

位ビットにつながっている側が勝ちになります．早押しクイズの類でより公平を期するならば，前回の勝者の優先度が低くなるようなアルゴリズムにしたほうがよいのかもしれません．

出力はパルス状に出します．実は，今回購入したチャイムはONのままにすると鳴り続けてしまいますので，人間がスイッチを押したときのようにパルス出力するようにしなくてはならないためです．また，ランプはOFFからONの変化で点滅/停止が切り替わるようになっているので，こちらもパルス出力にしました．パルス幅はチ

```
    sts = EZ_PIOWrite(3, &HFF)
    Winner = 0
    SwitchData = 0
End Sub

Private Sub Form_Load()
    Dim sts As Long
    Call EZ_Open
    SwitchData = 0
    PBDAT = 0
    PDDAT = &HFF

    sts = EZ_PIOWrite(1, PBDAT)
    sts = EZ_PIOWrite(3, PDDAT)
    sts = EZ_SetPortConfig(0, 2, 1, 1, 1, 1, 0)
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Call EZ_Close
End Sub

' タイマー割り込み
' スイッチの読み込みと、勝利者判定
Private Sub Timer1_Timer()
    Dim swdat As Integer
    Dim mask As Integer
    Dim i As Byte
    Dim sts As Long
    If (SwitchData = 0) Then    'すでに誰かが押した後
        swdat = EZ_PIORead(0)
        swdat = (swdat Xor &HFF) And &HFF
        If (swdat <> 0) Then    ' 誰かがスイッチを押した
            SwitchData = swdat
            mask = 1
            For i = 0 To 7
                Chk_SW0(i) = (swdat And mask) / mask
                If (Chk_SW0(i) <> 0) Then
                    Winner = mask
                End If
                mask = mask * 2
            Next i
            sts = EZ_PIOWrite(1, 1)
            sts = EZ_PIOWrite(3, Winner Xor &HFF)
            Sleep (100)
            sts = EZ_PIOWrite(1, 0)
            sts = EZ_PIOWrite(3, &HFF)
        End If
    End If
End Sub
```

ャイムやランプが認識できる幅が必要です．人間が扱うことが前提なので，それほど厳密なものではありませんが，目安として100 msオーダの値ならばまず問題ないでしょう．

今回は100 msにしていますが，使用しているユニットの仕様が変更されたり，繋ぐ相手が今回とは違うなどの理由でうまくいかない場合もあるかもしれません．そのときは適宜この待ち時間を調整してみてください．

100 ms待たせるのはフラグを使ってタイマ・イベントの先頭でチェックするという手もあるのですが，より簡

単な方法としてWindowsのシステム・コールである**Sleep()**を利用しました．**Sleep()**はEZUSBDRV.BASの中で，

```
Declare Sub Sleep Lib "kernel32" (ByVal msec As Long)
```

として定義しています．これで，

```
Sleep (待ち時間)
```

とすれば，指定した待ち時間（単位はms）だけ経過してから，次の行が実行されるようになります．

●実行例

完成した早押し判定器を動かしてみます．まず，ランプは全部消灯状態にセットしておきます．プログラムを実行して［LEDTEST］ボタンを押すと全部のランプの点滅と消灯が切り替わり，［BUZZTEST］でチャイムが鳴ることを確認します．もしうまく動かない場合には，配線が間違ってないか，極性があっているかなどをもう一度確認してください．

ここまでうまくいったらスイッチを一つ押してみます．判定が行われ，チャイムが鳴り，押した人のランプが点滅するはずです．他のスイッチについてもスイッチを押すとチャイムが鳴り，対応するランプが点滅すれば完成です．

電池駆動で，スイッチ入力で動く機器であればたいていのものは同じように接続可能であると思います．スイッチ入力をいったん受けてから，演算処理し，必要ならばスイッチが押されたようにふるまって駆動するという，スイッチ入力とパルス状のディジタル出力を行うものはいろいろと応用が利くのではないでしょうか．

4-2　電力のパルス制御の実験…PIOモードによるDCモータの回転速度とランプの輝度の制御実験

UCT-203ボードのI/Oポートをより積極的に利用した例として，パルス出力による電力制御を行ってみることにしましょう．制御ターゲットは100円ショップで買ったミニ扇風機とランプ型ライトにしましたが，インターフェース部分の変更でさまざまなものに応用が可能でしょう．

●電力制御の考えかた

DCモータの回転数やランプの明るさを変えるということで真っ先に思い浮かぶのは，**図4**のように抵抗で電流を制限する方法です．抵抗値が小さければモータが速く回り，抵抗値が大きければ遅くなります．

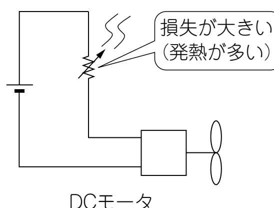

図4　抵抗による回転数制御

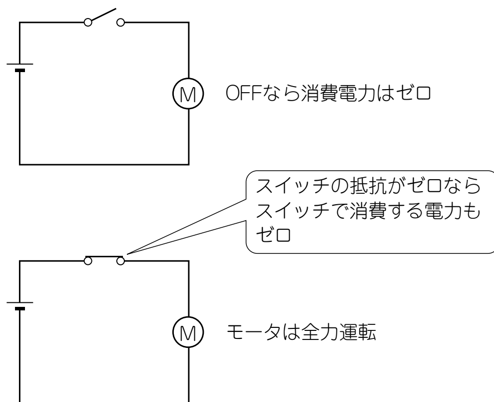

図5　ON/OFFだけなら効率が良い

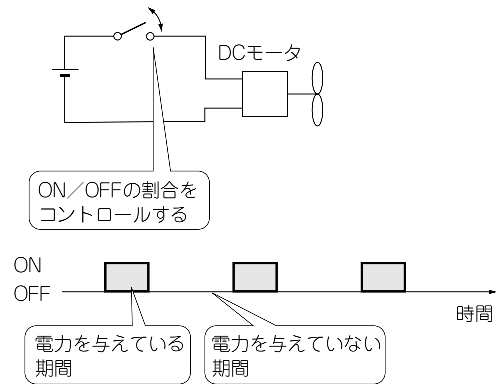

図6　パルス幅による制御の考えかた

この方法は簡単ではありますが抵抗による損失が大きい，つまり効率が悪いというのが大きな欠点です．たとえば，負荷の抵抗値を一定とすると，抵抗がないときの半分の電圧にするためには，残り半分は抵抗の電圧降下にするしかありません．電源が10 Vで負荷の内部の抵抗が10 Ωとしましょう．このとき，負荷にかかる電圧を半分にするためには10 Ωの抵抗を繋ぐことになります．10 Ωの負荷と10 Ωの抵抗が直列で20 Ωになりますから，10 Vかければ流れている電流は0.5 Aです．

この場合，抵抗も負荷もそれぞれ$0.5^2 \times 10 = 2.5$ Wの電力を消費することになります．つまり，負荷で消費している電力と同じだけの電力を抵抗で消費させていることになります．このぶんの電力がまるまる熱になってしまいますので，放熱の問題も出てきます．

実際には可変抵抗を使うのは難しいので，トランジスタやFETを利用した回路が使われることが一般的ですが，発熱が大きいとそれだけ大きな損失に耐えられるトランジスタ/FETを選定しなくてはならなくなり，熱対策もたいへんです．

●ON/OFFで制御する

アナログ的に変化させるのが難しいならば，ONとOFFの二つの状態だけを使って制御すればよいのではないかというのが，パルス制御という方法です．**図5**にこの考えかたを示します．

完全にOFFのときは電流が流れないのですから消費電力は制御側，負荷側ともゼロです（実際には若干漏れ電流があったりするが，無視できる程度）．逆に，完全にONで抵抗が0 Ωならば負荷側は100％動作ですが，制御側の消費電力はやはりゼロになります．実際には0 Ωということはありえませんが，たとえば負荷の抵抗の100分の1ならば負荷の100分の1の電力しか食わないのですから，先ほどのようなアナログ的な制御のときに比べればはるかに効率が良いと言えるでしょう．

そこで，アナログ的に変化させるのではなく，ONの状態とOFFの状態を高速に切り替えることで負荷に供給する電力を変化させようというのがパルス制御方式の考えかたです．これを**図6**に示します．たとえば，ONと

（a）ミニ扇風機　（b）ライト

写真3　利用したミニ扇風機とライト

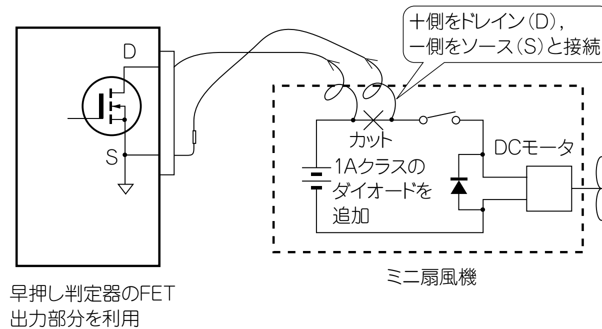

図7　ミニ扇風機の改造と接続

OFFの状態が半分ずつならば供給する電力は半分，ON時間1に対してOFF時間が2ならば1/3という具合に，ON時間の比率で負荷に与える電力を制御できるという考えかたです．もちろん，負荷の側がこの切り替え速度に追従して動いてしまうようでは動きがギクシャクしてしまいますが，追従しきれない程度に充分に高速に切り替えるならば問題はないだろうというわけです．

＊　＊

それでは，この考えに基づいて，実際にパルス制御によるコントロールの実験をしてみましょう．

●部材の調達

コントロールするターゲットは，先ほど触れたように，100円ショップで買ってきたミニ扇風機とランプ型のライトを選んでみました（**写真3**）．どちらもスイッチが付いていますので，スイッチと直列にON/OFF制御用の信号を引き出します．

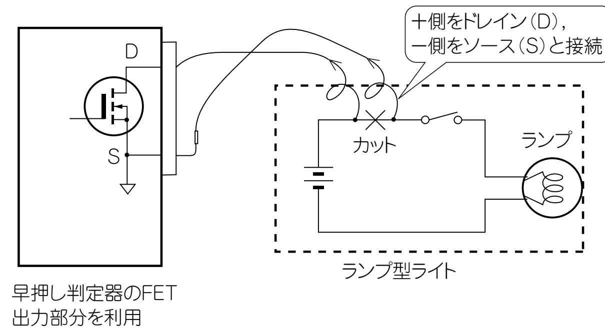

図8　ランプ型ライトの改造と接続

扇風機のほうは**図7**のようにモータを使っていますので，OFFのときの逆起電力対策として，モータの端子に並列にダイオードを入れておきます．ダイオードの向きを間違えると，電池をダイオードでショートすることになってしまうので，気を付けてください．

接続先は，早押し判定器のときにチャイムを繋いだのと同じところで，ドレイン側をプラス側，ソース側をマイナス側と結線します．電線はチャイムについていたものが余ったのでそのまま流用しましたが，極端に細いものでなければ何でも良いでしょう．もし，極性がわからなくなったときにはダイオードをつないでみて，回転するときのアノード側がプラス，カソード側がマイナスです．

ランプのほうはランプのスイッチつまみを引き抜いてから，底面のねじをはずすと簡単に分解できました．こちらも同様に，**図8**のようにバッテリ・ケースのプラス側の端子からスイッチに行っている線をはずして外に引き出します．極性も同様に区別できるようにしておきます．

●供給電力を変化させる方法

ハードウェアの準備は整いましたので，次にソフトウェアの検討に移りましょう．まず，どのように変化させるかを考えます．パルスによる制御の場合，ある一定時間を切り取ってみたときに，そこでONになっている時間の割合で供給される電力が決まってくるのでした．

すぐに思いつくのは，たとえば一周期を100 msにして，1/10にしたいなら，最初の10 msをON，残り90 msをOFFにしておくという方法です．与えたい電力によってパルスの幅が変わるので，PWM（Pulse Width Moduration）方式といわれます．

この方法はハードウェアで実現するのも非常に簡単で，便利な方法です．一定期間ごとにトリガをかける部分と，指定された時間だけONになる回路を用意しておくだけで実現できるといういう簡便性もあって広く利用されています．それではまず，この方法をVisualBasicのタイマ・コントロールを使って試してみることにしましょう．

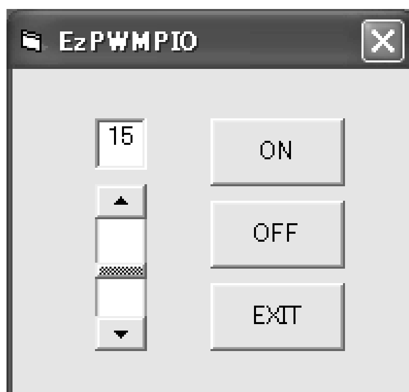

図9　電力制御のプログラムの実行例

●プログラムの作成

作成したアプリケーションの画面は**図9**のようなものです．［ON］と［OFF］のボタン・スイッチで動作開始/停止を行い，スクロール・バーで供給電力を0～32の33段階にコントロールします．ソース・プログラムは付属CD-ROMのEzPWMPIOフォルダに収録してあります．

ON/OFFのタイミングはVisualBasicのタイマ・コントロールを使用しました．実測してみるとタイマは10ms周期が最小時間のようなので，10msに設定しています．1周期の間でスクロール・バーで指定されたカウント数ぶんだけこのタイマ・イベントの中でONにして，残りの時間がOFFになるようにしました．

●実行例

FETの先に扇風機やランプをつないでプログラムを起動します．スクロール・バーをとりあえず真ん中よりも上のほうにしてONボタンを押すとファンが回ったり，ランプが点灯するはずです．スクロール・バーを移動させれば，確かにファンの回転は速くなったり遅くなったりします．

このときの出力波形を取ってみました．**図10**が設定値を「8」にしたときの波形，**図11**が設定値が「20」のときの波形です．図中の波形の一番上のLine0が上側（Hレベル）にあるときがON，下側にあるときがOFFです．設定値が増えるに従って1周期（縦の2本の破線の間，320ms）の中でのON時間が伸びていることがわかります．

しかし，実際に動かしてみるとわかるとおり，この方法ではモータの動作はギクシャクした動きで，お世辞にもスムーズとは言いがたい状態です．ランプは明るさが変わっているのではなく，信号機のように点滅してしまっています．

考えてみればわかるとおり，32カウントが1周期で，1カウントが10msですので，1周期が320ms，つまり0.3秒もあるのです．これではいくらモータには慣性が働くとはいっても，この周期では遅すぎてON/OFFされていることがわかってしまうのは仕方ないでしょう．ランプにしても点滅が目に見えてしまうのは当然です．

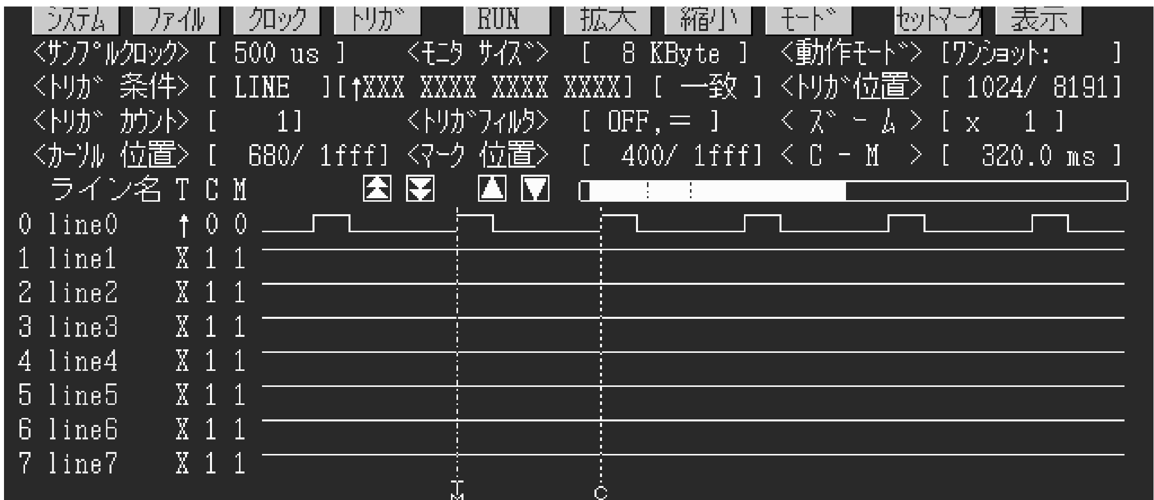

図10　設定値が「8」のときの出力パルス

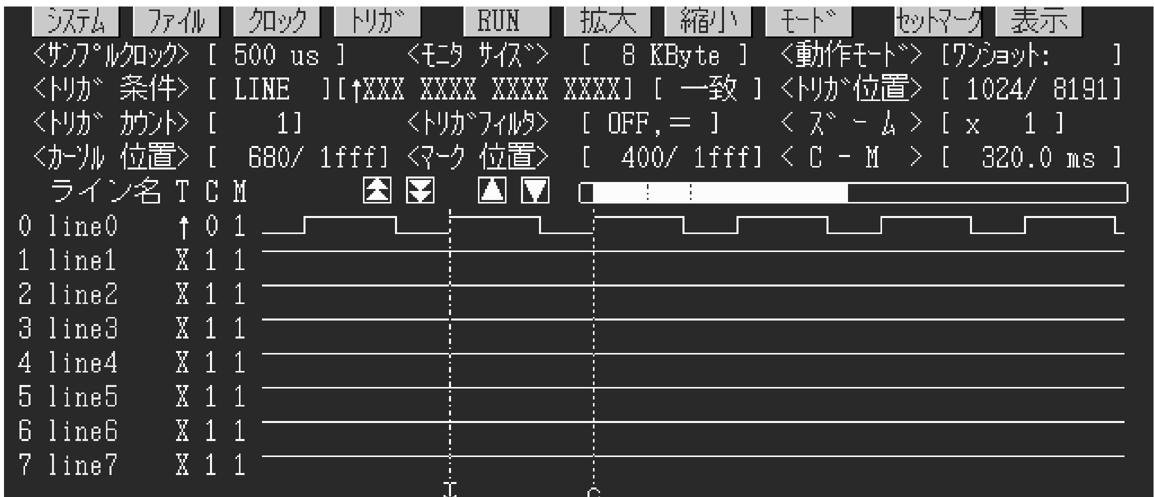

図11　設定値が「20」のときの出力パルス

●プログラムの改良

10msという周期はVisualBasicのタイマ・コントロールを使うかぎりはいかんともしがたいようですが，何とかもう少しスムーズな感じにならないものでしょうか．ここで，思いついたのが，パルスの数を変えずに位置を分散させるという方法です．たとえば，先ほどの方法では設定値が「2」の段階のときは20ms連続でON，残り300msがOFFという状態ですが，これを10ms ONで150ms OFF，再び10ms ONで150ms OFFという具合にするのです．320msの中でのON期間が20msというのは同じですが，連続してOFFになっている期間が半分になるので，もう少しスムーズに感じられるのではないかという考えかたです．

この方法ではパルスの分散のさせかたがポイントになります．単純に1/nならば1個ON，（$n-1$）個OFFという方法でよいのですが，3/5などといった場合はどうしたらよいでしょうか．このときは，加算していって余りが蓄積されるようにすることで対処できそうです．つまり，与えられた値を足していって，32以上になったら32を引くということを繰り返すわけです．

32は少々大きいので，一例として5までの段階で考えてみます．累積値に設定値を足してが5未満ならOFF，5以上になったらONにして，5を引いておくという操作を繰り返してみます．

初期値を0とします．もし与えられた値が1ならば，

0→1（OFF）→2（OFF）→3（OFF）→4（OFF）→（5→0）(ON)→1（OFF）…

という繰り返しですから，5回のうち1回がONです．

与えられた値が3ならば，

0→3（OFF）→（6→1）（ON）→4（OFF）→（7→2）（ON）→（5→0）（ON）→3（OFF）…

リスト2　電力制御のプログラムの主要部（EzPFWMPIO.frm）

```
'=================================================================
'=  EzFirm/FX2ボード応用例                                        =
'=  ＰＩＯ操作によるＰＷＭ                                        =
'=   10msのインターバルタイマーを利用                              =
'=                                                               =
'= PB[0] : モーターなどを接続（'1'でONになると想定）                 =
'=================================================================
Dim Period As Byte
Dim OnTimer As Byte
Dim CTimer As Byte
Dim Enable As Boolean

Private Sub Motor_ON()
    Dim sts As Long
    sts = EZ_PIOWrite(1, 1)
End Sub

Private Sub Motor_OFF()
    Dim sts As Long
    sts = EZ_PIOWrite(1, 0)
End Sub

Private Sub CMD_END_Click()
    Enable = False
    Call Motor_OFF
    Call EZ_Close
    End
End Sub

Private Sub CMD_OFF_Click()
    Enable = False
End Sub

Private Sub CMD_ON_Click()
    Enable = True
End Sub
Private Sub Form_Load()
```

となりますので，5回のうち3回がON（OFF，ON，OFF，ON，ON）の繰り返しになります．

単純な加減算だけでよいですし，パルスの位置は比較的うまく散らばってくれそうです．

●プログラミング

それでは実際にプログラムに落としてみることにしましょう．プログラム（EzPFWMPIO.frm）の主要部分を**リスト2**に示します．タイマ・イベントの処理のうち今回のパルス分散をやっているのが，下記の部分です．

```
CTimer = CTimer + OnTimer
If CTimer >= Period Then
    Call Motor_ON
    CTimer = CTimer - Period
Else
```

```
    Dim sts As Long
    Enable = False
    Period = 32
    CTimer = 0
    VScrl_Speed.Max = Period
    VScrl_Speed.Min = 0
    VScrl_Speed.value = VScrl_Speed.Max
    OnTimer = VScrl_Speed.Max - VScrl_Speed.value
    Text_Speed = Str$(OnTimer)
    Call EZ_Open
    Call Motor_OFF
    sts = EZ_SetPortConfig(0, 2, 1, 1, 1, 1, 0)
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Call EZ_Close
End Sub

Private Sub Timer1_Timer()
    Dim sts As Long
    If Enable = True Then
        CTimer = CTimer + OnTimer
        If CTimer >= Period Then
            Call Motor_ON
            CTimer = CTimer - Period
        Else
            Call Motor_OFF
        End If
    Else
        Call Motor_OFF
    End If
End Sub

Private Sub VScrl_Speed_Change()
    OnTimer = VScrl_Speed.Max - VScrl_Speed.value
    Text_Speed = Str$(OnTimer)
End Sub
```

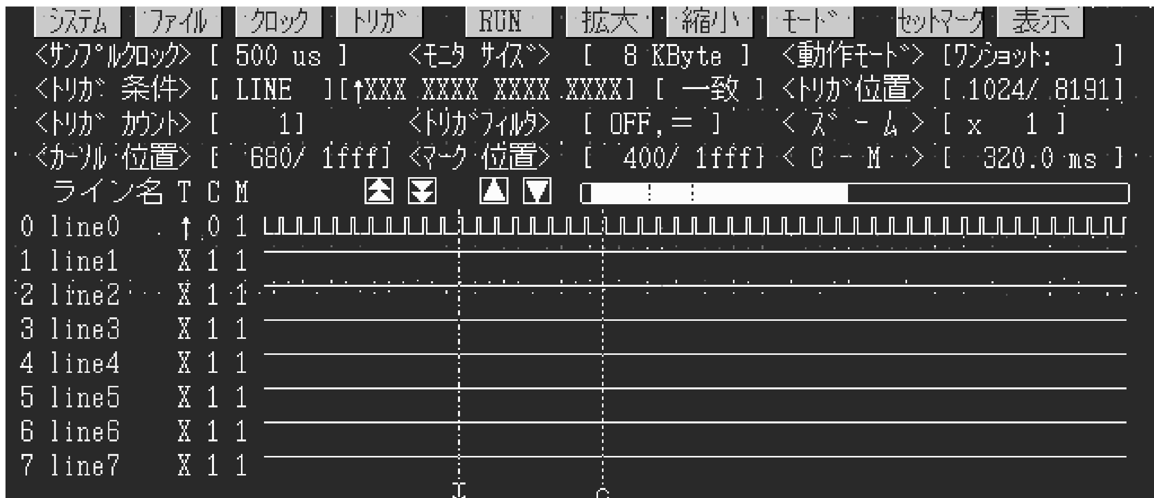

図12　設定値が「8」のときの出力パルス

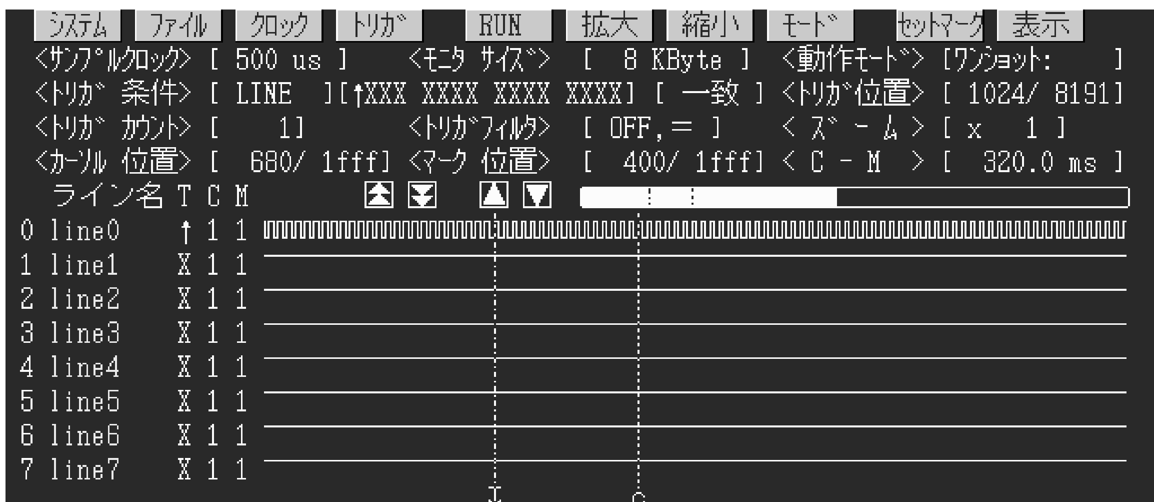

図13　設定値が「16」のときの出力パルス

```
            Call Motor_OFF
        End If
```

先ほど考えたとおり，**Ctimer**に**OnTimer**の値を足して，周期値（**Period**）以上になったらモータをONにして周期ぶんを引くという作業を繰り返すだけです．

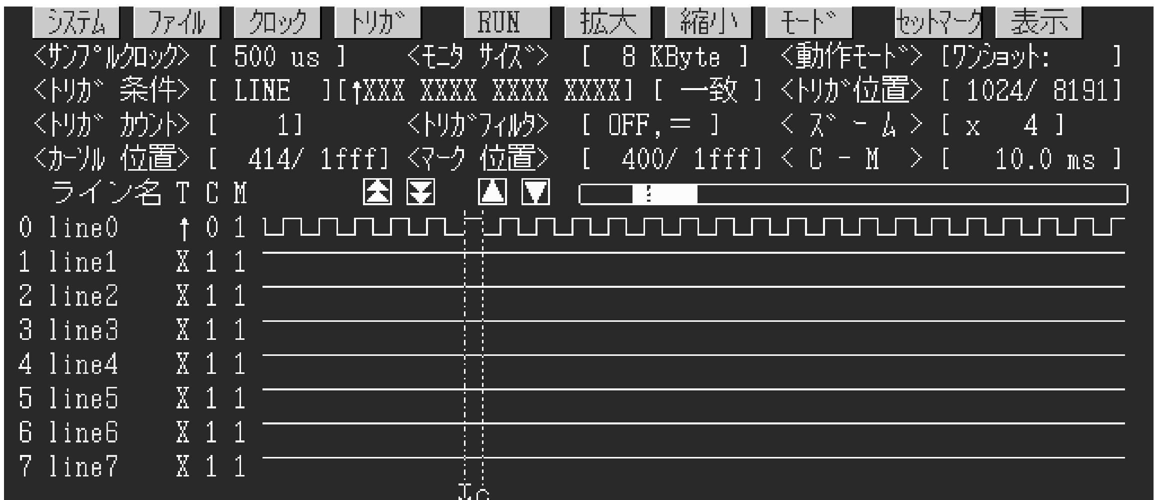

図14　図13の時間軸を拡大したようす

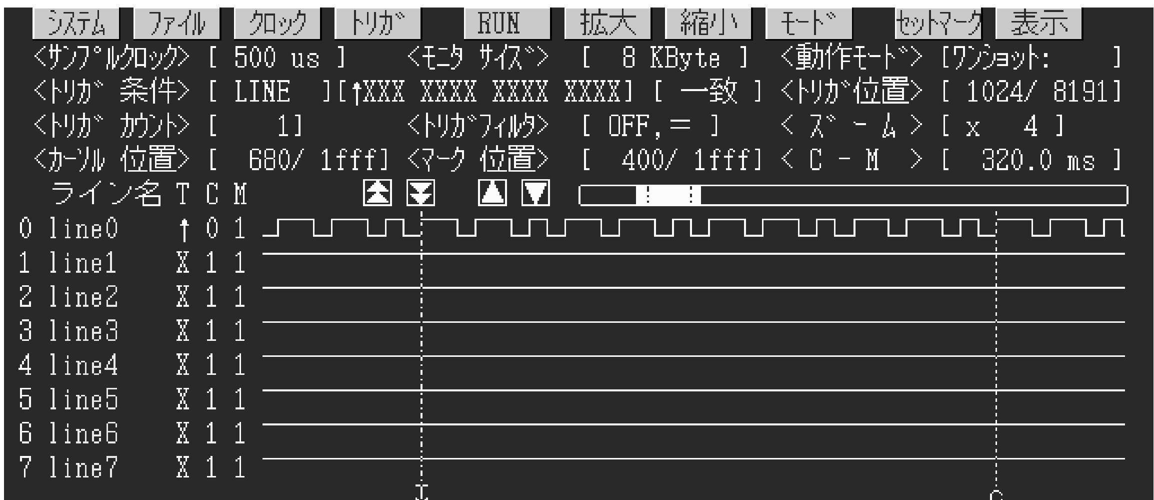

図15　設定値が「20」のときの出力パルス

●動作確認

それでは実際に動かしてみましょう．値を16程度にしてONしてみると，どうでしょうか．先ほどの単純PWMよりも，はるかにギクシャク感がなくスムーズになったことが感じ取れます．ランプのほうも確かに点滅してはいますが，単純PWMのときの信号機のような点滅とは違い，細かい点滅になったことが見て取れます．

波形を見てみましょう．先ほどと同じサンプリング時間で設定値が「8」のときの状態が**図12**です．同じ1周期（320ms）の中に8個のパルスが分散されていることがわかります．ちょうど設定値が真ん中の「16」の状態

が**図13**です．細かくてわかりづらいので拡大したのが**図14**です．デューティ比50％のきれいなパルスになっていることが読み取れます．さらに設定値を「20」にしたのが**図15**です．比較的うまく分散配置されていることが見て取れます．

減算処理が必要になるなど，74シリーズの標準ロジックなどで実装するのはやや面倒そうですが，FPGA/CPLDを使ったり，ソフトウェアで処理することが前提ならば，この方法のほうが良さそうです．

4-3　よりきめ細かい動作へ…GPIFを使ったDCモータとランプの制御

PIOを使った方法はわかりやすく簡単ですが制御の最小単位，つまりON期間やOFF期間の最短時間をあまり短くできません．サンプルでは10 msに設定しましたが，VisualBasicとWindowsの組み合わせでは，このくらいが限界としておくよりないというところでしょう．ただ，やってみるとわかるとおり，この周期は小型のモータやランプ用としては少々長すぎるように思えます．

もう少し何とかならないかということで，FX2のもつ（EzFirm/FX2がサポートしている）GPIF機能に目を付けてみました．

■GPIFを使う

先ほどの実験で，モータやランプ制御用のFETはPORTB［0］に接続しました．このポートは，GPIFモードで使ったときにはFD［0］になり，ホストからEP2に送られたデータの最下位ビットが出力されます．

したがって，FX2をGPIFモードに設定して，ホスト側から0001h，0000h，0001h…という具合に最下位ビットを変化させたデータを送れば，モータやランプのON/OFF状態を切り替えることができます．GPIFは48 MHzのクロックで動作しているので，最小パルス幅は約21 nsまで短くすることができる計算です．もっとも，これは計算上の話で，現実にはこれではあまりにも短すぎてFETがスイッチング速度に追従できませんから，実際にはもっと長いパルス幅が必要です．

データ伝送を利用したこのモードをうまく動かすには，データ転送時間について考えておかなくてはなりません．FX2はUSB2.0のハイスピード伝送（480Mbps）に対応していますが，USB1.1ポートに接続したときのことも一応考えておいたほうがよいでしょう．

USB1.1のバルク転送では，1パケットのサイズは64バイトです．EzFirm/FX2はFX2のエンド・ポイントを4バンク構成で動作させているので，まとめて送ることができるデータ量は256バイトとなります．GPIFモードではデータ・バス幅は16ビットなので，1回の出力動作で2バイトが出力されますから，128転送ぶんのデータまでしか蓄えることができません．USB2.0で接続すれば，1パケットは512バイト（256ワード）になりますから，4バンクの合計2Kバイト（1Kワード）ぶんまで蓄積することができます．

もちろん，蓄えることができるとはいっても，平均したデータ転送速度はホスト側のほうが上回っていないとデータの欠落が起きてしまいます．つまり，データがFX2から出るよりも速い平均速度でデータを送ってやらなくてはならないことになります．たとえば，出力が2クロックで終了してしまうと，24 Mワード/秒＝48 Mバイト/秒ですから，ホスト側からはおおむね50 Mバイト/秒という速度でデータを供給しなくてはならないことにな

り，これではUSB2.0でも苦しい領域になってしまいます．

なんとかしてFX2からデータが出ていく速度を落とすよりありません．それでは，GPIF単体ではどのくらいの時間まで引き延ばすことが可能なのでしょうか．GPIFはフレキシブルに設計されていますから，外部にタイミングを取るためのタイマを設けて一定周期ごとに転送させることもできますが，今回はなるべく単体で行うようにすることを考えてみます．

外部回路を使わずに転送にかかる時間を延長するためには，GPIFのノンディシジョン・ポイントを利用します．ノンディシジョン・ポイントでは，あるステートに留まる時間を1クロックから256クロックまで任意に設定することができます．ユーザが自由に使えるステートは7ステートありますが，データの更新のためのステートが必要ですので，これに1ステート使うと残りは6ステート，したがって最大256×6クロックぶんの時間だけデータを出し続けてから次のデータに更新するということができる計算です．1ワードの転送にかける時間が1÷48［MHz］×（256×6＋2）≒32 μs（＋2はステート6と7のぶん）です．

USB2.0ならばこれが1024ワードぶんということは，おおよそ38msぶんのデータになります．先ほどのデータ転送実験で10ms間隔までは何とかなりそうだったので，ここは38msでよしとしましょう．

一方，USB1.1で接続した場合にはパケット・サイズが1/8になってしまうので，4.7msごとに128ワードを送ることになり，ちょっと忙しいことになります．リアルタイムOSならば何の問題もない程度の時間ですが，WindowsとVisualBasicという環境では少々厳しいようにも思われます．

とりあえず今回は，USB2.0を前提にして実験することにしました．

●プログラミング

GPIFを使用する時にはウェーブフォーム・ディスクリプタを作らなくてはなりません．少々面倒に感じられるかもしれませんが，一つずつ順を追って考えていけばそれほど難しいものではありませんし，今回のようなものは比較的単純です．

プログラム（EzPWMGPIF.frm）の主要部は**リスト3**のようになります．

まず，ステート0からステート5までの間（データを出しつづけるところ）では，

LENGTH/BRANCH = 00h（256クロック）
OPCODE = 02h（ノンディシジョン・ポイントでデータ出力）
OUTPUT = 3Fh（CTLラインは使わないので何でもよい）
LOGIC FUNCTION = 00h（ノンディシジョン・ポイントでは未使用）

となります．ステート6はノンディシジョン・ポイントでも実現可能ですが，データを更新してステート7に飛ぶというディシジョン・ポイントを使いました．こちらは次のようになります．

LENGTH/BRANCH = 3Fh（論理演算の結果が‘1’でも‘0’でもステート7へ飛ぶ）
OPCODE = 07h（ディシジョン・ポイント，データは出力したまま次のデータに更新）
OUTPUT = 3Fh（CTLラインは使わないので何でもよい）
LOGIC FUNCTION = 00h（CTL0同士のAND条件）

プログラム中ではGPIFが使用しないステート7用のデータ・エリアにもデータをセットしていますが，これは

リスト3　GPIFモードでのパルス出力のプログラムの主要部（EzPWMGPIF.frm）

```
'================================================================
'=  EzFirm/FX2ボード応用例                                        =
'=  ＧＰＩＦデータ転送によるＰＷＭ                                    =
'=                                                              =
'=  '1'/'0'のデータ列を送ることでPWMにする                          =
'=  GPIFは48MHz動作                                              =
'=  256クロックのノン・ディシジョンポイントで7ステート                  =
'=  1/48MHz×256×7=37μs                                       =
'=  USB1.1だと64バイト×4=128ワードなので約4.7ms分                   =
'=  USB2.0なら512バイト×4=1024ワードなので、約38ms分                =
'=                                                              =
'= PB[0] : モーターなどを接続（'1'でONになると想定）                  =
'================================================================
Dim waveform(2047, 32) As Byte
Dim wsw(31) As Byte
Dim Period As Byte
Dim OnTimer As Byte
Dim CTimer As Byte
Dim Enable As Boolean

Private Sub Motor_ON()
    Dim sts As Long
    sts = EZ_PIOWrite(1, 1)
End Sub

Private Sub Motor_OFF()
    Dim sts As Long
    sts = EZ_PIOWrite(1, 0)
End Sub

Private Sub CMD_END_Click()
    Enable = False
    Call EZ_Close
    End
End Sub

Private Sub CMD_OFF_Click()
    Enable = False
End Sub

Private Sub CMD_ON_Click()
    Enable = True
End Sub

Private Sub Form_Load()
    Dim sts As Long
    Dim i, j, k As Integer
' Length/Branch :: Opcode ::  Output :: LogicFunction
    wsw(0) = 0: wsw(8) = 2: wsw(16) = &H3F: wsw(24) = 0
    wsw(1) = 0: wsw(9) = 2: wsw(17) = &H3F: wsw(25) = 0
```

特に意味はありません．

波形データ（**weveform**配列）のデータを毎回考えているのはたいへんなので，あらかじめ32段階ぶんをすべて用意しておきます．プログラム上では次のようなループでセットしています．今回は送ることができる周期が速いので，単純なPWM方式でもよさそうですが，PIOモードでのコントロールでなかなかうまく動いたパルス

```
    wsw(2) = 0: wsw(10) = 2: wsw(18) = &H3F: wsw(26) = 0
    wsw(3) = 0: wsw(11) = 2: wsw(19) = &H3F: wsw(27) = 0
    wsw(4) = 0: wsw(12) = 2: wsw(20) = &H3F: wsw(28) = 0
    wsw(5) = 0: wsw(13) = 2: wsw(21) = &H3F: wsw(29) = 0
    wsw(6) = &H3F: wsw(14) = 7: wsw(22) = &H3F: wsw(30) = 0
    wsw(7) = 0: wsw(15) = 0: wsw(23) = &H3F: wsw(31) = 0

    For i = 0 To 32
        k = 0
        For j = 0 To 1023
            k = k + i
            If (k >= 32) Then
                waveform(j * 2, i) = 1
                k = k - 32
            Else
                waveform(j * 2, i) = 0
            End If
            waveform(j * 2 + 1, i) = 0
        Next j
    Next i
    Enable = False
    Period = 32
    CTimer = 0
    VScrl_Speed.Max = Period
    VScrl_Speed.Min = 0
    VScrl_Speed.value = VScrl_Speed.Max
    OnTimer = VScrl_Speed.Max - VScrl_Speed.value
    Text_Speed = Str$(OnTimer)
    Call EZ_Open
    sts = EZ_SetPortConfig(2, 2, 1, 1, 1, 1, 0)
    sts = EZ_WaveSet(1, wsw(0))        ' 0:BRD 1:BWR 2:SRD 3:SWR
    sts = EZ_GPIFTrig(0, &H7FF0000)
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Call EZ_Close
End Sub

Private Sub Timer1_Timer()
    Dim sts As Long
    Dim xfrlen As Long
    If Enable = True Then
        sts = WritePipe(hUSB, hBOUT, waveform(0, OnTimer), 2048, xfrlen)
    End If
End Sub

Private Sub VScrl_Speed_Change()
    OnTimer = VScrl_Speed.Max - VScrl_Speed.value
    Text_Speed = Str$(OnTimer)
End Sub
```

を分散させる方法にしてみました．

GPIFモードの場合，FX2から出ていくデータはワード（16ビット）単位ですが，今回は最下位ビットしか使いません．偶数バイト目を計算結果で0または1にして，奇数バイト目は常に0を入れるようにしています．

```
    For i = 0 To 32
```

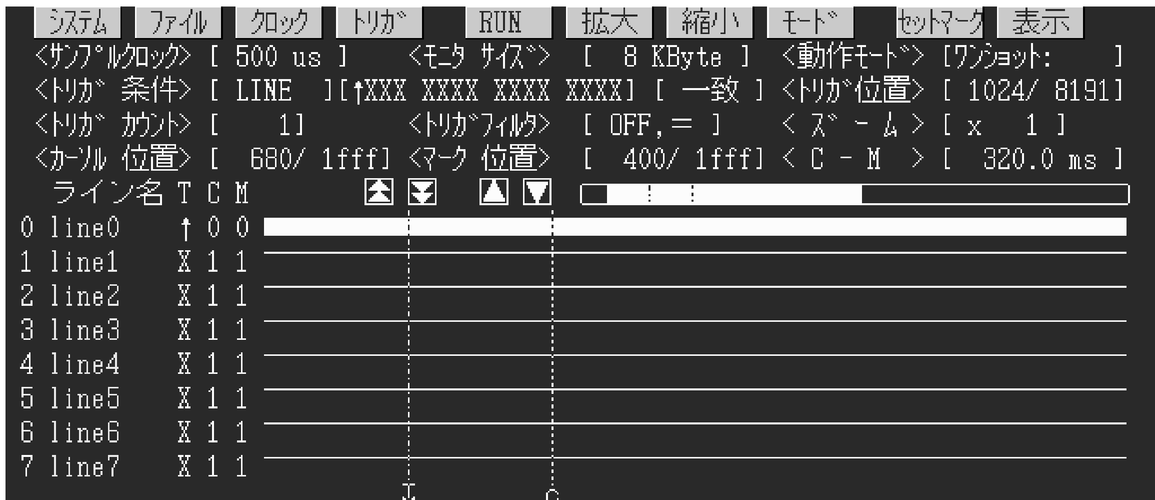

図16　PIOモードと同じサンプリングで測定したようす

```
        k = 0
        For j = 0 To 1023
            k = k + i
            If (k >= 32) Then
                waveform(j * 2, i) = 1
                k = k - 32
            Else
                waveform(j * 2, i) = 0
            End If
            waveform(j * 2 + 1, i) = 0
        Next j
    Next i
```

なお，この方法でフルスケールの1/2を越えるまでの間はONパルスの幅は最小パルス幅以上にならないので，最小パルス幅に対する配慮が必要です．今回，実際の波形を見たかぎりでは，32 μs［1/48 MHz ×（256 × 6ステート + 2)］あれば，充分に追従可能なようなので，パルス幅は細工しませんでした．もし，繋ぐ相手やバッファ/ドライバの特性によってもっと長い時間が必要な場合には，パルスを2個ペアで使ったり，単純PWM方式にするなどの配慮が必要でしょう．

●実行例

プログラムができたら，試しに扇風機を繋いでみましょう．プログラムを起動して，スクロール・バーはとりあえず真ん中以上にして（最大値でもかまわない)，［ON］ボタンを押してみてください．回転がはじまり，スク

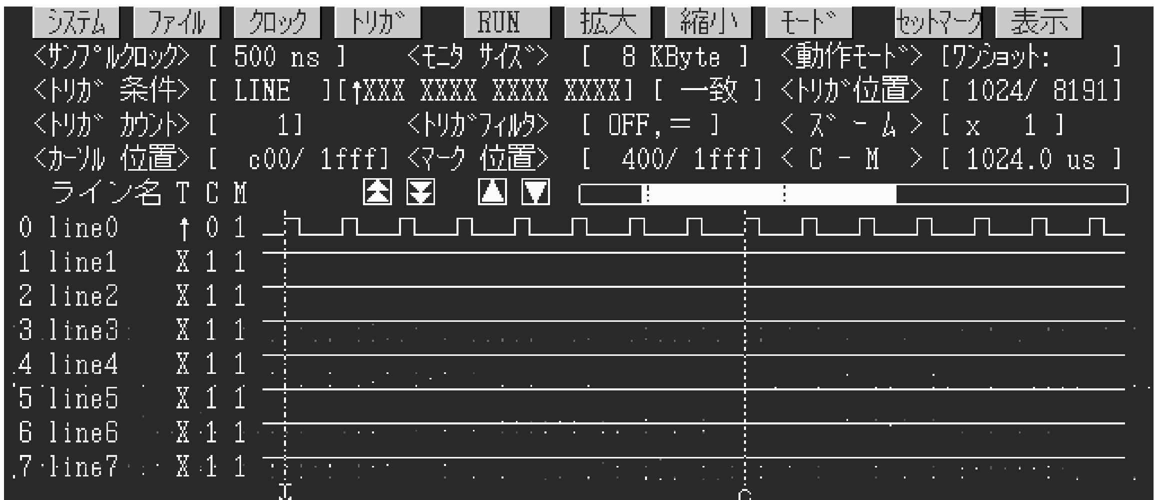

図17　図16の時間軸を拡大したようす

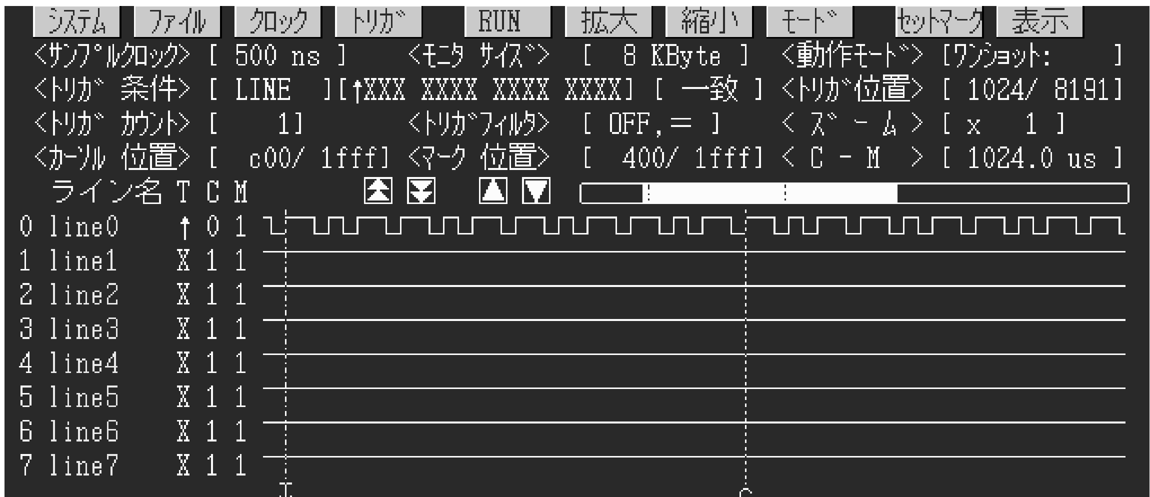

図18　設定値を「20」にしたときの出力パルス

ロール・バーを下げていくとだんだん回転が遅くなるはずです．いきなり回りつづけてしまう場合には，極性を確認してください．

どうでしょうか．筆者の実験では，PIOモードのときのようなギクシャクした感じはなくなりました，低速回転のときには扇風機の音が弱まるのと，パルスの周期が可聴域に入ってくることもあってモータから「ピーッ」という音が聞こえてきます．一度回転が始まると値を相当下げても回りつづけることがわかります．パルスの周期が短いことも手伝って，かなり速度が落ちてもギクシャクした感じもなく，なかなか良い具合に動くことがわかります．

確認のために波形を見てみましょう．先ほどのPIOモードでの制御と同じサンプリング時間で取ると**図16**のように真っ白になってしまいました．非常に細かいパルスになっていることがわかります．サンプリング時間を変えて拡大したのが**図17**です．これは設定が「8」のときのもので，図では表示されていませんが，Hレベルの期間が32 μs，Lレベルが96 μsになっていました．

さらに設定値を「20」にしたのが**図18**です．スケールが違うのですが，PIOモードのときと同じようにパルスが分散されているようすがわかります．

■GPIFの無限ループ動作の利用

今までの例ではすべて，ホストからのコントロールで‘1’や‘0’の状態を決定していました．簡単な方法なのですが，実際に動かしてみるとわかるのですが，パソコン上でちょっと他のプログラムが動き出すと制御が止まってしまいますし，ウィンドウをドラッグしたりすると，ググッとファンの回転が不規則になってしまいます．

これを防ぐには，ある程度のリアルタイム性が保障された切り口をうまく利用する方法もあるでしょうが，もっと簡単にホストがいちいちON/OFFを制御するという考えかた自体をやめるという方法があります．忙しくても単純な作業は下に任せて，ホストCPUの負荷やUSBバスのトラフィックを下げてやろうという考えかたです．

USBターゲットのファームウェアを専用に作り込むならば話は簡単で，ホストからは出力パターンや速度パラメータだけを渡して，あとはターゲット側で自動的に繰り返し動作をさせておけばよいのです．しかし，EzFirm/FX2にはそのような繰り返しパターン出力のコマンドは用意されていません．普通に考えると，この方法は無理のように思えますが，FX2の場合にはGPIFがあるということがポイントです．

確かにEzFirm/FX2では，FX2の内蔵CPUが実行するプログラムには手が出せません．ところが，GPIFについてはユーザが自由に書き換えられる切り口をもっています．先ほどの実験でもわかるとおり，GPIFはあるデータを一定期間だけ出力して，次の状態に移動するという，まるでCPUのような動作をさせることが可能です．つまり，GPIFを非常に原始的なCPUとして捉えると，ウェーブフォーム・ディスクリプタをプログラム，ステート・インストラクションを命令コードとして見ることができ，上からプログラムを送り込んで自動的に動かすことができるユニットがあるのと同じようなものなのだということがわかります．

通常，GPIFは1バイトの入出力（シングル・リード/ライト），あるいは複数バイトのデータ入出力（バースト・リード/ライト）を行うために使います．一連の動作が完了したら，ステート7（アイドル・ステート）で停止するというのが一般的な使いかたですが，GPIFのディシジョン・ポイント機能を使って，絶対にアイドル・ステートに戻らないようなウェーブフォーム・ディスクリプタを作ると，GPIFは終わりがないということも知らずに延々とステート・インストラクションの実行を継続します．このなかでパルスを出力するようにしておけば，ホストがいっさい関与しなくてもGPIFが連続したパルスを出力しつづけることになるわけです．

出力するパルス幅を変更したいときには，いったんGPIFを強制停止させて，ウェーブフォーム・ディスクリプタを書き換えて再度実行させればよいわけです．この停止から起動までの間は制御が停止してしまうことになりますが，今回のような使いかたならば書き換えはスクロール・バーの値を変更するときなどに限られ，頻繁に変更するところでもないのでかまわないでしょう．

少々トリッキーな方法でもありますが，一定のパルスを出しつづけたいような場合には便利な使いかたといえ

るのではないかと思います．

●回路の変更

GPIFを利用したデータ出力の場合，データ・バス（FD［0：15］）はあくまでもホストから送られてきたデータの出力用として使われるだけで，GPIFが自らデータを出力することはできません．ステート・インストラクションの中で自由に指定可能なのはCTL端子の状態だけです．

このため，今回の実験ではCTL0のラインにFETを付けてそこから制御を行うようにしました．早押し判定器の回路図で一番下の点線で囲ってある部分がこのための回路です．

●プログラミング

プログラム（EzPWMCTL.frm）の主要部は**リスト4**のようなものです．フォームがロードされたときに，33段階ぶんのウェーブフォーム・データを配列に作成しています．データ出力は特に意味はないので省略で，CTLラインを操作してパルス幅を作ります．

```
wsw(0, i) = Sqr(i) * 256 / Sqr(32) ' ON時間を設定
wsw(8, i) = 0
wsw(16, i) = &H3F     ' CTL出力ON
wsw(24, i) = 0
```

といった具合です．最初の行がLENGTH/BRANCHフィールドですが，ここでパルスの保持時間を決めています．次のOPCODEは通常ならばデータ・バスへの出力やデータ更新をコントロールするのですが，今回はCTLラインを操作したいだけなので特に意味はありません．続くOUTPUTフィールドでCTL端子の状態を決定します．最後はLOGIC FUNCTIONフィールドですが，ノンディシジョン・ポイントなので，このフィールドは使われません．

SQR（平方根）を使った計算をしているのは，直線的に変化させると値が小さいときの動きが芳しくなかったために行った細工です．プログラムを見てわかるとおり，今回のサンプルではステート0とステート1がON期間，ステート2とステート3でOFF期間を決めています．つまり，ON期間は最小で2クロック（約42ns），最大でも512クロック（約11 μs）と，かなり短いパルスで制御することになります．この期間を均等割りすると，特に値が小さくON時間が短いときにはFETなどが追従しきれず，予定どおりの電力が供給できません．これでは少々面白くないので，平方根を使って小さい値のときのON時間を延ばしてみたのです．リニアでやってみたい場合には右辺を単なる**i**にすればよいでしょう．

速度変更は**Moto_ON()**の中で，**EZ_Waveset()**によるテーブル書き換えで行います．GPIFの強制停止コードがありませんが，実はEzFirm/FX2では**EZ_Waveset**リクエストが行われると自動的にGPIFを停止させてからウェーブフォーム・ディスクリプタの書き換えを行うようになっているため，あえてホストから停止させる必要はないためです．

EZ_Waveset()のあと，**EZ_SglWtNW()**を呼び出します．本来は1ワードのデータを送るためのものですが，ここではGPIFに起動をかける目的で使用しています．**EZ_SglWtNW()**はデータ書き込み動作を行ったあと，

リスト4　GPIFの無限ループを利用したプログラムの主要部（EzPWMCTL.frm）

```
'================================================================
'=  EzFirm/FX2ボード応用例                                       =
'=  ＧＰＩＦの無限ループによるＰＷＭ                              =
'=                                                              =
'=  CTL0のON/OFFで制御                                           =
'=  GPIFは48MHz動作                                              =
'=                                                              =
'= PB[0] : モーターなどを接続（'1'でONになると想定）               =
'================================================================
Dim wsw(31, 32) As Byte
Dim Speed As Byte
Dim Enable As Boolean

Private Sub Motor_ON()
    Dim sts As Long
    sts = EZ_WaveSet(3, wsw(0, Speed))      ' 0:BRD 1:BWR 2:SRD 3:SWR
    sts = EZ_SglWtNW(0, 0)
End Sub

Private Sub Motor_OFF()
    Dim sts As Long
    sts = EZ_WaveSet(3, wsw(0, 0))      ' 0:BRD 1:BWR 2:SRD 3:SWR
    sts = EZ_SglWtNW(0, 0)
End Sub

Private Sub CMD_END_Click()
    Enable = False
    Call Motor_OFF
    Call EZ_Close
    End
End Sub

Private Sub CMD_OFF_Click()
    Call Motor_OFF
    Enable = False
End Sub

Private Sub CMD_ON_Click()
    Call Motor_ON
    Enable = True
End Sub

Private Sub Form_Load()
    Dim sts As Long
    Dim i, j, k As Integer

' Length/Branch :: Opcode ::  Output :: LogicFunction
    For i = 1 To 31
        For j = 0 To 31
            wsw(j, i) = 0
        Next
        wsw(0, i) = Sqr(i) * 256 / Sqr(32) ' ON時間を設定
        wsw(8, i) = 0
        wsw(16, i) = &H3F    ' CTL出力ON
        wsw(24, i) = 0
```

```
        wsw(1, i) = Sqr(i) * 256 / Sqr(32) ' ON時間を設定
        wsw(9, i) = 0
        wsw(17, i) = &H3F    ' CTL出力ON
        wsw(25, i) = 0

        wsw(2, i) = 255 - wsw(0, i) ' OFF時間を設定
        wsw(10, i) = 0
        wsw(18, i) = 0      ' CTL出力OFF
        wsw(26, i) = 0

        wsw(3, i) = 255 - wsw(0, i) ' OFF時間を設定
        wsw(11, i) = 0
        wsw(19, i) = 0      ' CTL出力OFF
        wsw(27, i) = 0

        wsw(4, i) = 0       ' ステート0にジャンプ
        wsw(12, i) = 1      ' ディシジョンポイント
        wsw(20, i) = 0
        wsw(28, i) = 0
    Next
    '値が0の時
    wsw(0, 0) = 0           'ステート0にジャンプ
    wsw(8, 0) = 1
    wsw(16, 0) = 0
    wsw(24, 0) = 0
    '値が32の時
    wsw(0, 32) = 0           'ステート0にジャンプ
    wsw(8, 32) = 1
    wsw(16, 32) = &H3F
    wsw(24, 32) = 0
    Call EZ_Open
    sts = EZ_SetPortConfig(2, 2, 1, 1, 1, 1, 0)
    Speed = 0
    Call Motor_ON
    Enable = False
    VScrl_Speed.Max = 32
    VScrl_Speed.Min = 0
    VScrl_Speed.value = VScrl_Speed.Max
    Speed = VScrl_Speed.Max - VScrl_Speed.value
    Text_Speed = Str$(Speed)
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Call Motor_OFF
    Call EZ_Close
End Sub

Private Sub VScrl_Speed_Change()
    Speed = VScrl_Speed.Max - VScrl_Speed.value
    Text_Speed = Str$(Speed)
    If Enable = True Then
        Call Motor_ON
    End If
End Sub
```

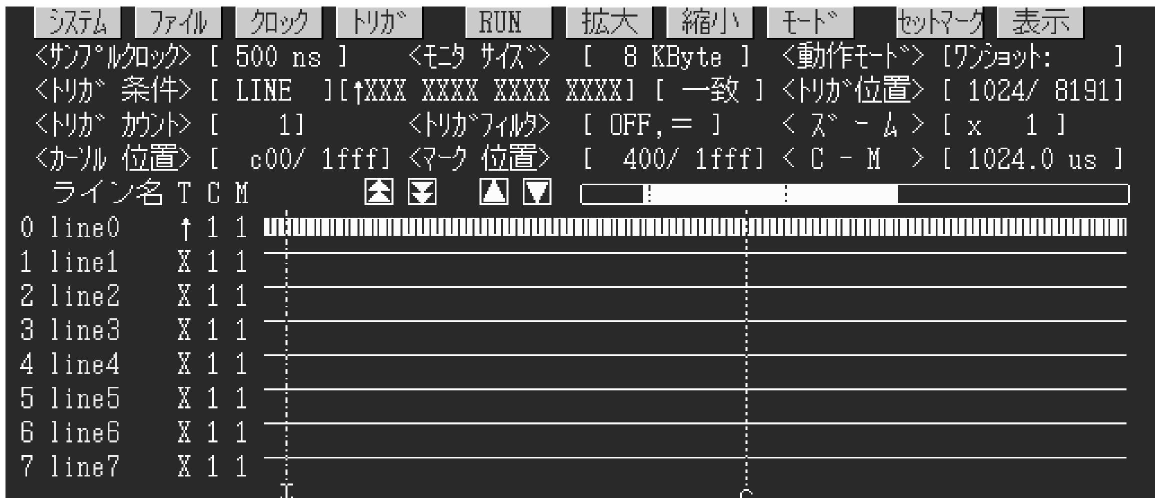

図19　GPIFのデータ転送を利用したモードと同じサンプリング周期で測定したようす

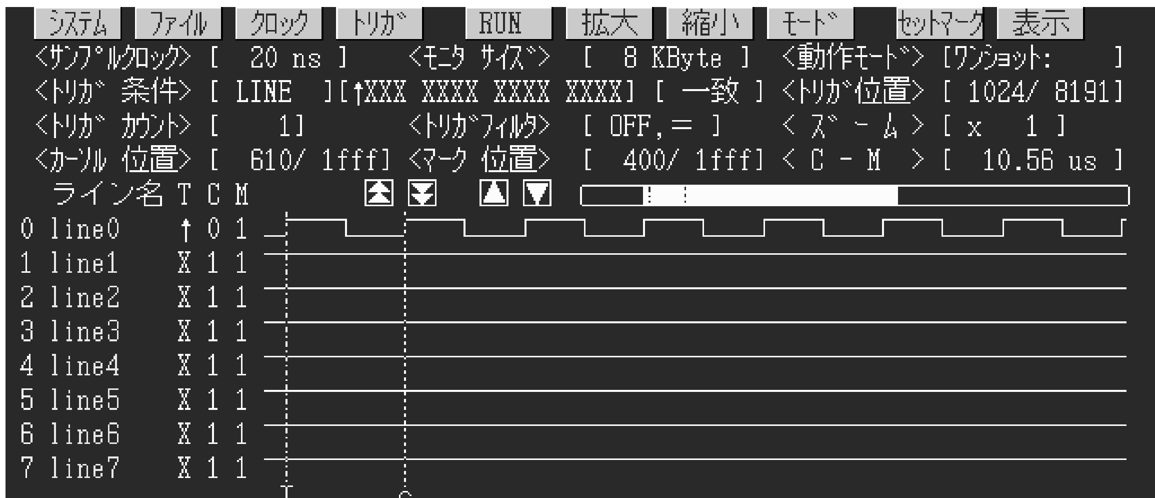

図20　設定値が「8」のときの出力パルス

GPIFの動作完了を待たずに戻ってきますので（`EZ_SglWt()`は書き込み動作完了を待つ）今回のような使いかたには適しています．

●実行例

　プログラムができたら動かしてみましょう．例によってスクロール・バーは真ん中より少し上あたりにセットしてONにします．動き出したらスクロール・バーを上下してみましょう．PIO制御のときのようなガタガタし

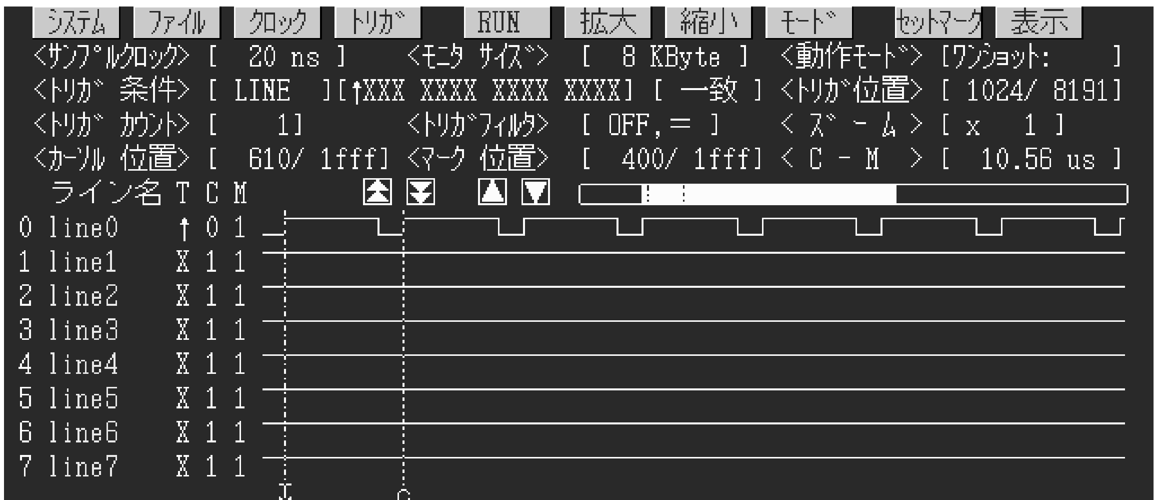

図21　設定値が「20」のときの出力パルス

た感じがないのはもちろん，GPIFデータ転送で行ったときのような「ピー」という音もなく，アナログ的な制御をしているようなスムーズさです．

先ほどのGPIFのデータ転送を利用したモードと同じサンプリング周期（500 ns）で波形をとったのが**図19**です．一段と細かくなったおかげで，なんだかよくわかりませんので，サンプリング周期を20nsまで細かくして取ったのが**図20**です．平方根を使ったので設定値は「8」ですが，デューティはほぼ50％になっています．

設定値を「20」にしたときのものが**図21**です．単純PWMのときと同様に幅が変化していることがわかります．同じ単純PWMでも周期が非常に短くなっていますので，PIOモードで行ったときのようながたつきは感じられません．

また，当然のことですがパルス生成はGPIFが独立して行っていますので，ホスト側のCPUにいくら負荷がかかったとしても，扇風機の回転やランプの明るさには変化がないというのは大きな利点といえるでしょう．速度切り替えが頻繁に発生する場合にはウェーブフォーム・ディスクリプタの更新の時間が無視できなくなりますが，今回のような使いかたであれば問題ないと言ってよいと思います．

●まとめ

扇風機とランプのコントロールということで，簡単なPIO制御，GPIFによるデータ転送の使いかた，そしてGPIFのちょっとトリッキーな使い方をそれぞれ試してみました．プログラム自体は見てのとおりの単純さです．今までUSBで四苦八苦していたのは何だったのかと思うようなものでしょう．ぜひこの実験をベースに面白い応用を考えてみてください．