第20章 タッチ式もぐらたたきゲームを作る

―― 動画表示の基本と高速描画テクニックをマスタ

本章では，付属 CD-ROM に収録されたタッチ式もぐらたたきゲームを動かしてみます．動画表示の基本や高速描画テクニックを解説します．

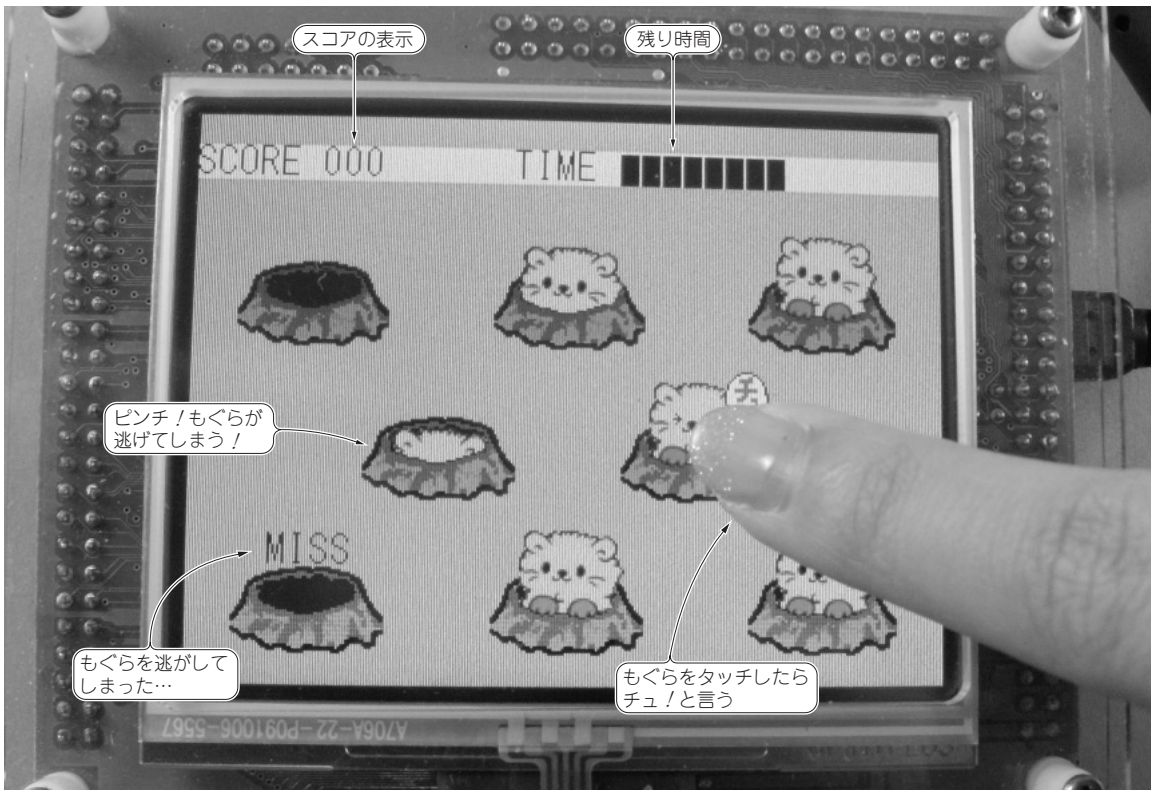


写真1　タッチ式もぐらたたきゲーム

本章では，付属 CD-ROM の add¥Workspace にある mole_rom フォルダを C:¥Workspace にコピーして使います．

タッチ式もぐらたたきゲームとは

Workspace¥mole_rom フォルダには**写真1**のようなタッチ式もぐらたたきゲームが収録されています．

タッチ式もぐらたたきゲームを動かしている動画を本書サポート・ページ（http://toragi.cqpub.co.jp/tabid/284/Default.aspx）に公開しています．

10 秒間でどれだけ多くのもぐらをたたけるか（タッチできるか）を競うゲームです．ハードウェア構成は第 19 章**写真 1**(p.159)と同じです．

表示が，

「SCORE 000　PLEASE TOUCH PANEL」

の状態でスクリーンをタッチすると，

「SCORE xxx　TIME ■■■■■■■■■■」

という表示になり，ゲームが始まります．SCORE はもぐらをたたくと 10 点加算されます．ミスは減点しません．

ゲームの残り時間は「■」1 個が約 1 秒で表示されています．表示個数が 0 になるとゲーム終了です．

ゲームが終了すると，

「SCORE xxx　GAME END　RETRY」

という表示になります．

RETRY をタッチするとゲームを再開できます．

実験の手順

ゲーム書き込みの手順を説明します．第 19 章のお絵描きソフトウェアの場合とほぼ同じですから，異なる部分のみ説明します．

(1) mole_rom.hws をダブルクリックし，HEW を起動してプロジェクト・ワークスペースを開きます(**図 1**)．
(2) Release コンフィグレーションを選択します．
(3) **図 2** のようにビルド(コンパイル，アセンブル，リンク，ファイル合成)します．

統合開発環境 HEW の評価版は期限に制約があり，60 日を経過した後は 64 K バイト以下のプログラムしかビルドできません．そこで，64 K バイト以上のもぐらたたきゲームがビルドできるように 64 K バイト以下のサイズで分割ビルドしています．その結果，H8SX/1655 に書き込むファイルが三つ(bmp_font_ROM.mot，bmp_mole_ROM.mot，mole_ROM.mot)生成されます．

ただしプログラム書き込みツール FDT は三つのファイルを指定できないので，ファイルを一つにまとめるプログラム file_merge.exe を用意しました．

(3)でビルドを実行すると，三つの mot ファイルを作成し，自動的に file_merge.exe を実行して三つのファイルを合成し，mole.mot を作成します．

(4) フラッシュ書き込みツールの FDT を起動します．
(5) MB のジャンパ J_1 をショート状態にし，ブート・モードを選択します．
(6) FDT を設定します．
(7) FDT の設定で USER/DATA Area にフォルダ mole_rom の直下にある mole.mot ファイルを選択します．
(8) 書き込みを開始し，完了を待ちます(1 秒程度)．
(9) FDT を終了するか，または接続を解除します．
(10) MB のジャンパ J_1 をオープンにします．
(11) 電源を再投入します(USB の場合はいったん抜いてから再度差し込む)．もぐらたたきゲームの初期画面が表示されます．

もぐらの動画表示プログラミング

LCD 表示とタッチ操作で実現できる「もぐらたたきゲーム」の処理を紹介します．

まず，パソコンで作成したもぐらや文字のビットマップ・データを LCD に表示するためフォーマット変換をし，もぐらを動画表示します．もぐらを増やすと書き換え情報が多くなるため動きが鈍くなりますが，ソフトウェアで処理を高速化します．

● ソフトウェアの基本構造

図 3 に main 関数と周期割り込みの主な処理を示します．もぐらたたきゲームでは，main 関数と周期割

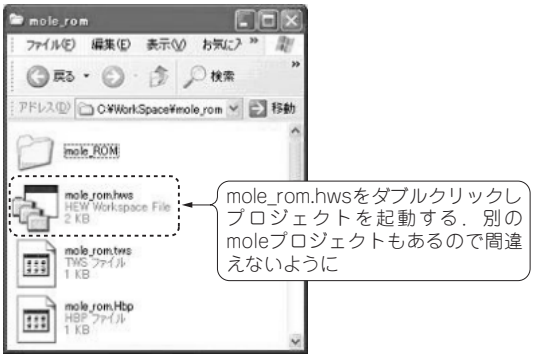


図 1　もぐらたたきの起動

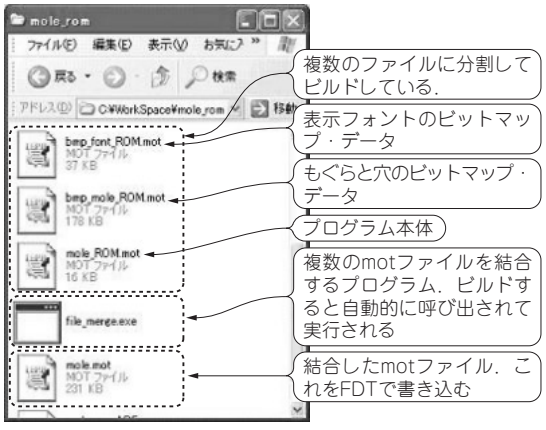
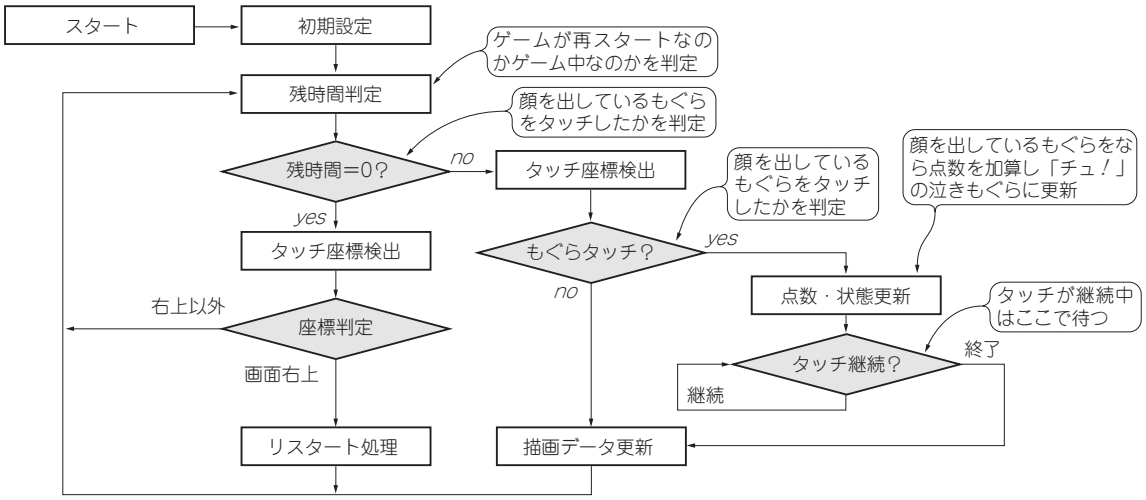
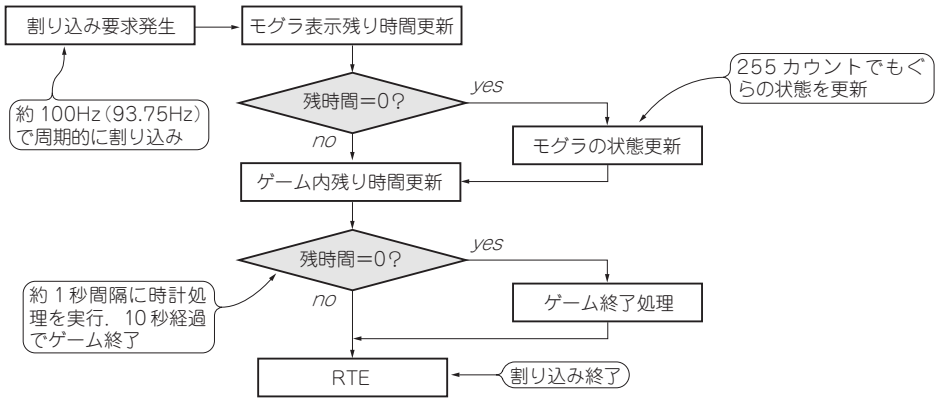


図 2　ロード・モジュールとファイルの結合



（**a**）メイン・ルーチン



（**b**）時間測定用の割り込み処理

図 3　もぐらたたきゲームの主な処理

り込み以外は，LCD 表示とタッチ・パネル処理のみに単純化しています．

main 関数は，ゲーム開始/停止の管理と，タッチあり/なしによる動作制御を行います．周期割り込みでは，8 体のもぐらの状態更新と残り時間の管理を行います．

● もぐらをピョコピョコと動かす処理

もぐらの状態は**図 4** に示すように 8 体それぞれを構造体変数で管理しています．draw メンバに書き換え要/不要を設定しています．

もぐらには**図 5** にあるように顔を出した，たたかれたなど六つの状態を用意しました．CPU の計算量を少なくするため，すべての状態の画像データを内蔵フラッシュ・メモリにあらかじめ記憶し，ゲームの進行に合わせて表示領域にコピーすることで画面を更新します．ただしデバッグ時（Appendix4 で紹介する mole プロジェクトの debug コンフィグレーション）は外部 SRAM に保存します．

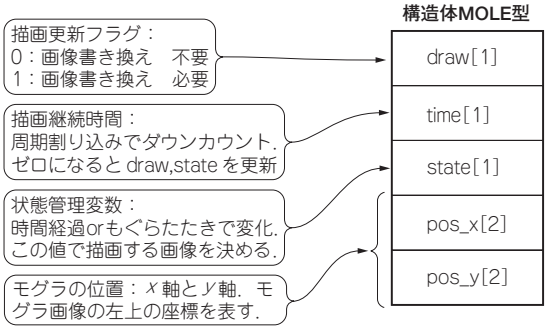
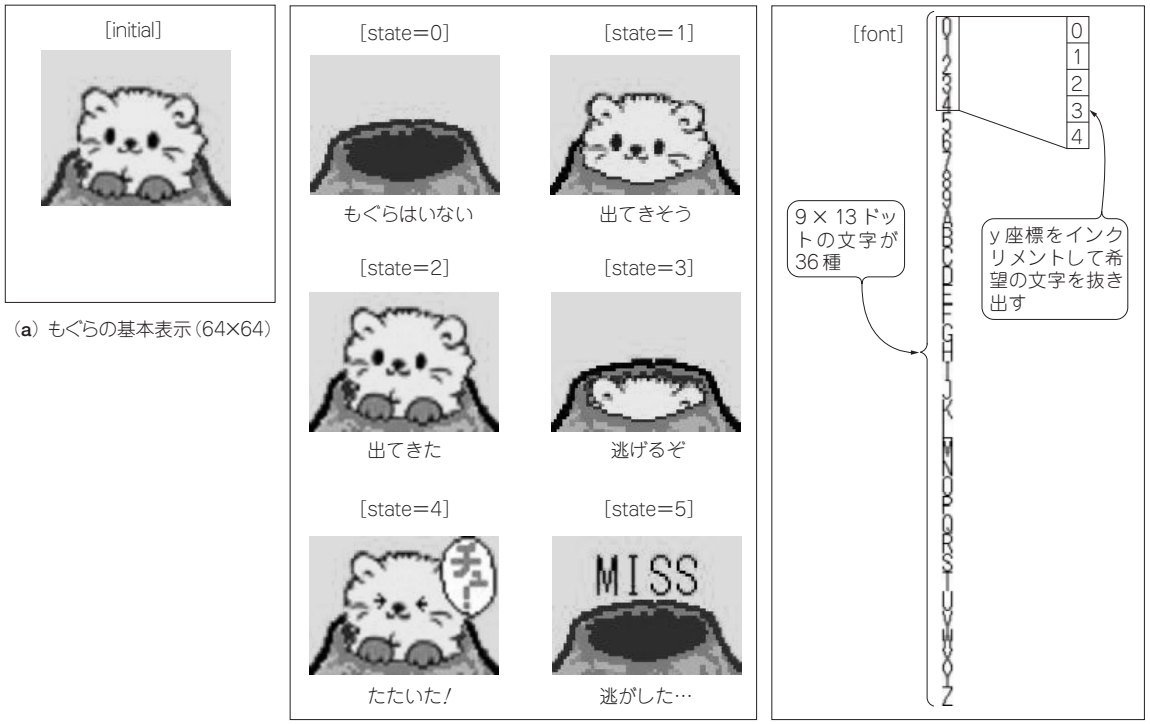
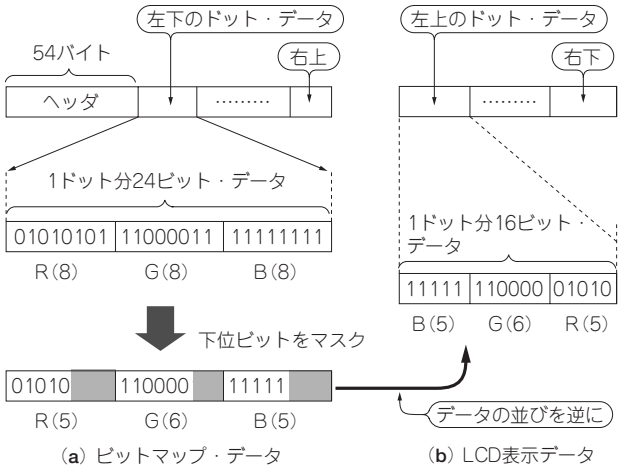


図 4　もぐらの状態を管理している構造体変数（1 体分）



(**a**) もぐらの基本表示(64×64)　(**b**) もぐらの動画データ(64×44)　(**c**) 文字(9×468)

図5　もぐらがとる六つの状態と表示の関係



(**a**) ビットマップ・データ　(**b**) LCD表示データ

図6　ビットマップとLCD表示フォーマットの違い

● パソコンで作成したビットマップのもぐらをLCDに表示させる

もぐらの絵はパソコンで編集します．パソコンで作成したもぐらはビットマップ・データとしてファイルに保管します．このファイルをH8マイコン基板MBに転送して利用しますが一つ問題があります．パソコンのビットマップと，LCD(拡張基板TB)で表示するためにSRAMに記憶するフォーマットが異なるのです．つまりフォーマット変換が必要です．パソコン側でもMB側でも変換できますが，今回はMBで変換することにしました．**図6**にパソコンで作成するビットマップとLCDの表示フォーマットの違いを示します．

まずビットマップ・ファイルからヘッダを除いたカ

リスト1　ビットマップ・データのもぐらをLCDに描画するプログラム(graphic.c)

```
void draw_mole(unsigned char mole_num)
{
    int i,j;
    int draw_point_y = mole[mole_num].pos_y +63 -12;
    int draw_point_x = mole[mole_num].pos_x;
    unsigned short mixed_rgb16;
    volatile unsigned char *p_bmp;
    switch(mole[mole_num].state)
    {
      case 0:  p_bmp = (volatile unsigned char *)&D_BMP_STATE0 + 54;  break;
      case 1:  p_bmp = (volatile unsigned char *)&D_BMP_STATE1 + 54;  break;
      case 2:  p_bmp = (volatile unsigned char *)&D_BMP_STATE2 + 54;  break;
      case 3:  p_bmp = (volatile unsigned char *)&D_BMP_STATE3 + 54;  break;
      case 4:  p_bmp = (volatile unsigned char *)&D_BMP_HIT + 54;  break;
      case 5:  p_bmp = (volatile unsigned char *)&D_BMP_MISS + 54;  break;
      default : break;
    }
    for(j=44; j>0; j--)
    {
      for(i=0; i<64; i++,p_bmp+=3){
        mixed_rgb16 = ( ((*(p_bmp+2)) & (MASK_RED))  << (SHIFT_RED))
                    + ( ((*(p_bmp+1)) & (MASK_GREEN))<< (SHIFT_GREEN))
                    + ( ((*(p_bmp+0)) & (MASK_BLUE)) >> (SHIFT_BLUE));
        global_lcd_framebuffer[draw_point_y][draw_point_x] = mixed_rgb16;
        draw_point_x++;
      }
      draw_point_y--;
      draw_point_x = mole[mole_num].pos_x;
    }
}
```

ラー・データのみを抜き出します．座標を上下逆順に格納しつつ，24ビット/ドットのデータを16ビット/ドットに圧縮しながら表示領域にコピーします．

実際にビットマップのもぐらをLCDに描画する関数を**リスト1**に紹介します．

どうですか？ 単純なゲームですが結構遊べますよね．もぐらたたきができるとその考え方や処理方法はいろいろな用途に応用できると思います．ぜひ遊びながらマイコンや処理の考え方を覚えてください．

応用のヒント：グラフィックス描画の高速化

■ もぐらの動きを速くする：描画用にブランキング期間を増やす

H8マイコン基板(MB)でもぐらを描画してもそこそこ動くのですが，やっぱり遅い感じがします．そこで，改善にチャレンジしてみましょう．

描画を速くして表示の更新を速くすればもぐらは素早く動きます．描画を速くする方法としては，一つは描画処理アルゴリズムの見直し，もう一つは描画に使える時間のねん出があります．

描画アルゴリズムでは，描画するたびに行っているもぐらのフォーマット変換を事前に行っておけば，変換時間をなしにできます．

● 描画に使えるブランキング期間にもぐらを約35回書き換えられる

次に描画時間のねん出です．もともと，描画時間はどのくらいあるのでしょうか？

H8SX/1655マイコンを使ったダイレクトLCD表示では，表示データを転送していないブランキング期間は自由にCPUがアクセスできます．つまりブランキング期間に描画できます．

描画可能期間(ブランキング期間)は，ドット・クロックDCLK=6 MHzの場合，**図7**に示すように約28.2 %の時間です．

ブランキング期間すべてを描画処理に充てることができたとすると1フレーム表示あたり408 × 22 + 88 × 240 = 30096ドットの処理ができ，フレーム周波数は　6 MHz/408/262 = 56.1 Hz　ですから1秒間で1,688,385ドットの処理ができます．

実際の描画の量はどれくらいかというと，ゲーム中，もぐらの出現によって書き換えが必要となるのは64 × 44ドット(2816ドット)です．8体分すべてを書き換えると22528ドット，結構な量です．フォントは9 × 13ドットのサイズですから，経過時間やスコア表示に4文字変更したとして486ドット，計22996ドット，約23000ドットの書き換え量です．すなわち，1秒間に73.4面分の書き換えができる計算になります．実際は内部で演算している時間もあるため，約半

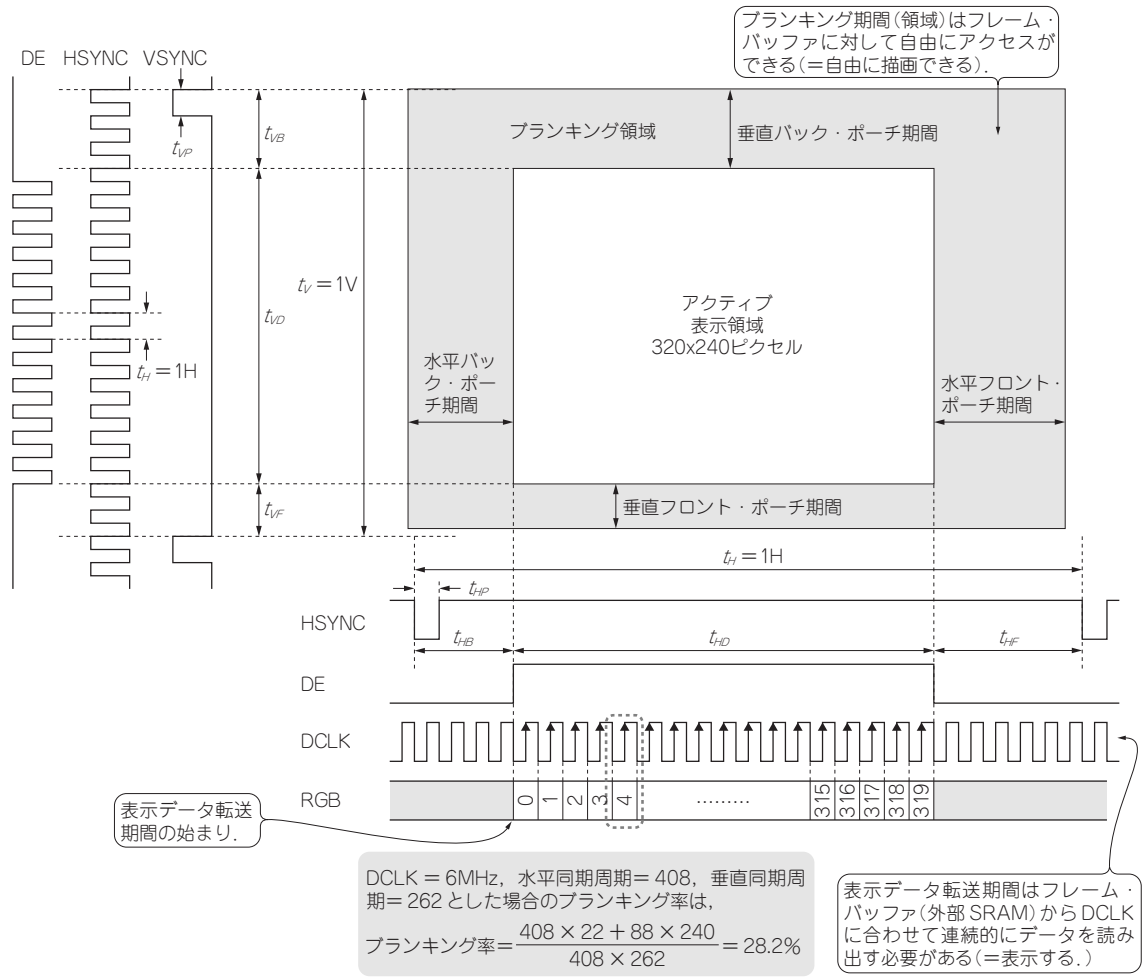


図 7　描画可能なブランキング期間の割合（改善前）

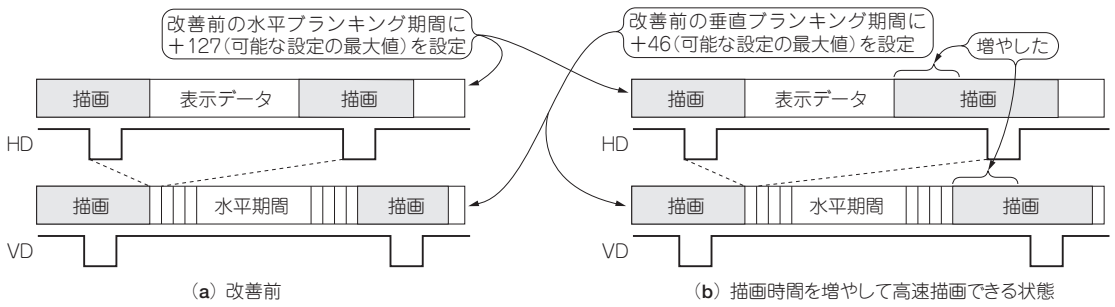


図 8　ブランキング率の向上の原理

分程度の実力といったところでしょう.

● ブランキング期間が長いほど，もぐらは速く動く

さらに描画時間を確保するにはブランキング時間を長くし，ブランキング率を上げます．**図 8** のように水平期間（HD）と垂直期間（VD）も長くし，描画時間を増やします.

しかし，ブランキング率を上げるとフレーム周波数が下がります．あまり下がりすぎると画面がちらついたり，表示の更新が遅くなったりと問題が発生する可能性があります．十分に検討して試してください.

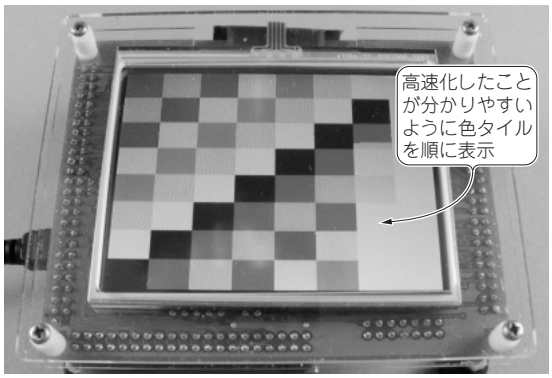


写真2　高速化実験で表示させるタイル
speedup.mot を書き込むと表示される

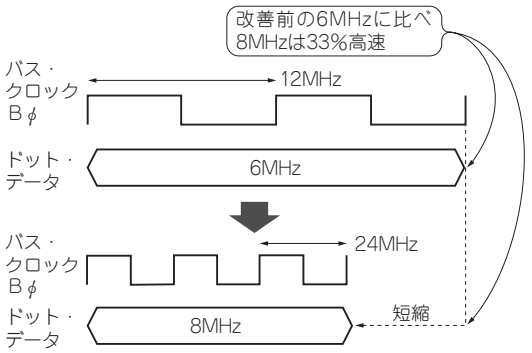


図9　バス・クロックの高速化

● 高速化の実験

add¥Workspace に収録した speedup フォルダを C:¥Workspace にコピーして，描画高速化を体験する実験を行ってみます．ビルドして生成された実行ファイル speedup.mot を MB に書き込んで実行すると**写真2**のようにタイルを描画します．

build.h ファイルにある，

```
#define  BACK_PORCH  OFF
```

が HD，VD ともにバックポーチ期間を Typ に設定した状態です．これを，

```
#define  BACK_PORCH  ON
```

に変更し，ビルド・書き込み・実行を行ってみてください．

水平バックポーチ期間を 38 から 165 へ＋ 127，垂直バックポーチ期間を 18 から 64 へ +46 増やしています[注]．1 フレーム表示あたり 535 × 149 + 134 × 240 = 111875 ドットの処理が行えます．フレーム周波数は 6 MHz/535/308 = 36.4 Hz ですから 1 秒間で 4,073,613 ドットの処理能力です．改善前の 1,688,385 に比べ 2.4 倍の描画時間が確保できました．このときのブランキング率は 67.9 %です．

どうですか？ 表示はかなり速くなると思います．やはり描画時間の確保は効きます．

■ もっと速くもぐらを動かす：ドット・クロックを高速化

さらに描画を高速化してみましょう．次に考えるのは表示データを転送するドット・クロック（バス・クロック）の周波数アップです．

LCD パネルの仕様ではドット・クロックの最大値は 11 MHz ですが今は 6 MHz です．

タイマ・パルス・ユニット TPU を使うと，動作クロック（Pφ = 24 MHz）の整数分の 1 を生成できます．ドット・クロック 6 MHz（24 MHz/4）から 8 MHz（24 MHz/3）に高速化できます．

具体的には**図9**に示すように，バス・クロック（Bφ）を 24 MHz とし，バス・サイクルのステート数を 3 クロック固定にすることで実現できます．ドット・クロックの高速化に伴って描画が約 33 %高速になります．

● フレーム周波数が速すぎても描画が間に合わないので意味がない

ドット・クロックを高速化するとどのような問題が発生するのでしょうか？ ドット・クロックのみを 8 MHz に高速化するとフレーム周波数が上がり，74.1 Hz と 70 Hz を超えます．1 秒間に「もぐら」の画面は 35 回しか更新できないのでフレーム周波数 70 Hz は速すぎます．表示フレーム・バッファ（SRAM）の内容が更新されていないにもかかわらず LCD へ表示のために転送しているという状態で，はっきりいって無駄です．フレーム周波数を 30 Hz 程度まで下げても問題ないでしょう．LCD がちらつかないかを確かめながらトライしてください．

● 高速化の実験

付属 CD-ROM の speedup プロジェクトの build.h ファイルにある，

```
#define  BUS_CLOCK  OFF
```

が 6MHz バス・クロックです．これを，

```
#define  BUS_CLOCK  ON
```

に変更すると 8 MHz になります．前述の mole_rom プロジェクトと同様に C:¥ ドライブにコピーして，ビルド・書き込み・実行を行ってみてください．

図10は高速化の対策イメージです．また対策によって得られた効果を**表1**に示します．効果は約 5 倍です．ちょっとした工夫ですが，読者の皆さんの参考になれば幸いです．

注：バックポーチ期間の変更は LCD パネルに SPI を通して設定を書き込みます．この設定はサンプル・プログラムが行っています．フロントポーチ期間を変更する場合は設定不要です．

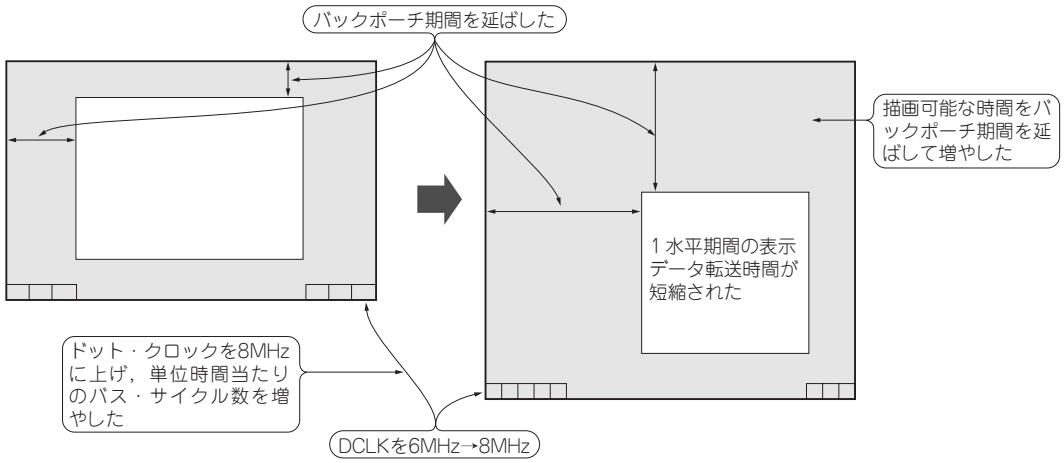


図10　ドット・クロックによる高速化

表1　高速化の対策と描画性能

BACK_PORCH	BUS_CLOCK	VD	HD	フレーム周波数	描画性能［サイクル/秒］	相対比
OFF	OFF	262	408	56.1 Hz	1,688,385	1
ON	OFF	308	535	36.4 Hz	4,073,613	2.42
OFF	ON	262	408	74.8 Hz	2,252,385	1.33
ON	ON	308	535	48.5 Hz	8,372,725	4.95

表2　利用したタイマ・パルス・ユニット TPU の機能

CH	目　的	端　子	動作モード	説　明
0	表示乱れ対策	—	通常	TGRA：水平周期，EXDMAC起動直前に周期割り込みを発生し外部バスをEXDMACに開放することで表示乱れを対策
1	未使用	—	—	—
2	水平周期	—	通常，マスタ	TGRA：水平周期， TGRB：割り込み発生（TPU3，4の設定，VCOUNTの計算）
3	HSYNC，VSYNC	TIOCA3，TIOCC3	PWM1，同期クリア	TGRA/TGRB：HSYNC， TGRC/TGRD：VSYNC（VCOUNT=0と1のみ動作）
4	DE，EDREQ	TIOCA4，TIOCB4	PWM2，同期クリア	アクティブ領域のみ，TGRA：DE（コンペアマッチでH）， TGRB：EDREQ（コンペアマッチでL）
5	DCLK	TIOCA5	PWM2	TGRA：パルス幅，TGRB：周期（6 MHz）
6〜11	未使用	—	—	—

● 描画と表示が重ならないように割り込みを要求

なお，EXDMAC が表示データの転送を開始するタイミング（水平表示期間開始時点）で CPU が描画していることがあるため，TPU からの同期信号と EXDMAC の表示データの転送タイミングがずれることがあります．対策として，TPU-ch0 をインターバル・タイマに設定し，水平表示期間開始の直前に割り込みを要求しました．これで CPU は水平表示期間開始時点で外部バスを使わないため表示が乱れることはなくなります．**表 2** に使用したタイマ・パルス・ユニット TPU をまとめました．

〈藤澤 幸穂〉