

## 第 19 章

タッチ式お絵描きソフト  
を作る

— 試しながらグラフィックス表示の基本をマスタ

本章では、付属 CD-ROM に収録されたタッチ式お絵描きソフトウェアを動かしてみます。実際に描画を手で試しながら、お絵描きソフトウェアで実際に行っているグラフィックス描画処理を解説します。

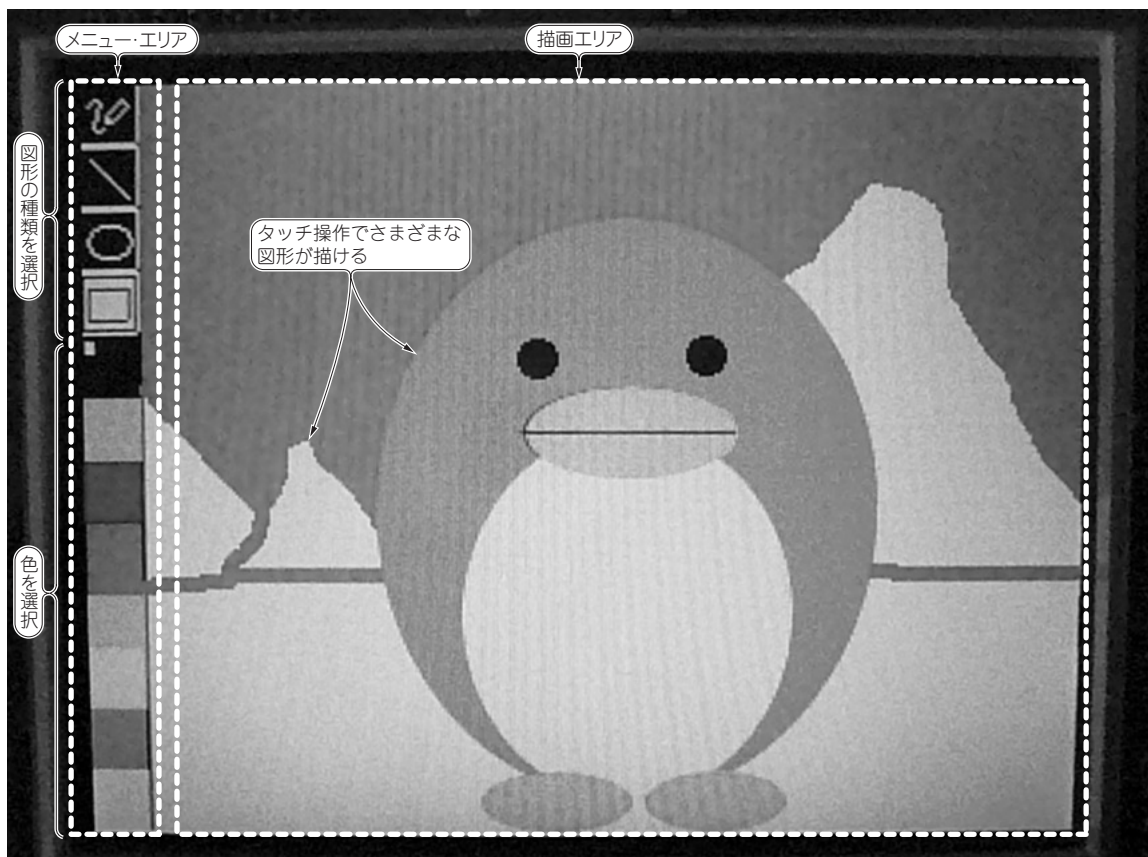

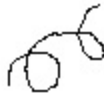



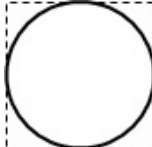








図 1 付属 CD-ROM に収録されているタッチ式お絵描きソフトウェア

表 1 描画できる図形

機能	選択するアイコン	描画する図形	解説
フリーハンド			ペンをタッチしている間指定した色でドットを描きます。
直線			最初にペンをタッチした位置から現在のペン位置まで直線を描画します。タッチしたままペンの位置を移動すると追従し描画します。ペンをスクリーンから離すとその位置で確定します。
楕円(円)			最初にペンをタッチした位置から現在のペン位置までの2点を頂点に持つ長方形に内接する楕円を描画します。タッチしたままペンの位置を移動すると追従し描画します。ペンをスクリーンから離すとその位置で確定します。
塗りつぶし楕円(円)			最初にペンをタッチした位置から現在のペン位置までの2点を頂点に持つ長方形に内接する楕円を描画します。タッチしたままペンの位置を移動すると追従し描画します。ペンをスクリーンから離すとその位置で確定し楕円内を外周と同じ色で塗りつぶします。
長方形(正方形)			最初にペンをタッチした位置から現在のペン位置までの2点を頂点に持つ長方形を描画します。タッチしたままペンの位置を移動すると追従し描画します。ペンをスクリーンから離すとその位置で確定します。
塗りつぶし長方形(正方形)			最初にペンをタッチした位置から現在のペン位置までの2点を頂点に持つ長方形を描画します。タッチしたままペンの位置を移動すると追従し描画します。ペンをスクリーンから離すとその位置で確定し長方形内を外周と同じ色で塗りつぶします。

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

### タッチ式お絵描きソフトでできること

Workspace¥paint フォルダには、図 1 のようなタッチ式お絵描きソフトウェアが収録されています。タッチ操作で以下のことができます。ハードウェア構成を写真 1 に示します。

- (1) 表 1 に示す図形の描画
- (2) 図 2 に示すような塗りつぶし色の選択

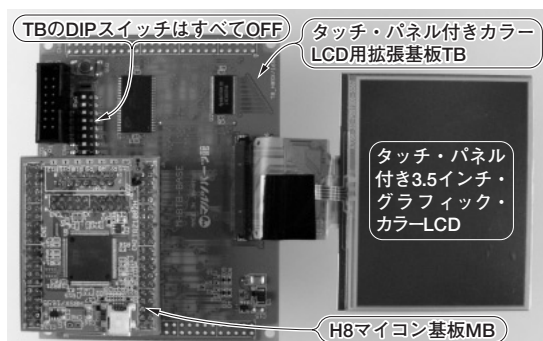


写真 1 タッチ式お絵描きソフトとタッチ式もぐらたたきゲーム(第 20 章)を動かすハードウェア構成

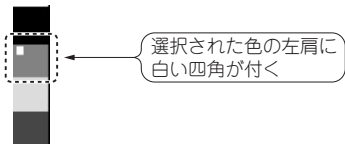
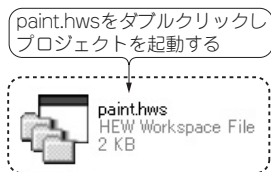


図2 8色(黒, 赤, 緑, 青, 黄, 紫, 水, 白)から塗りつぶす色を選択できる

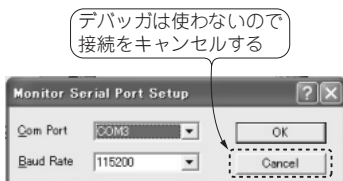
## 実験の手順

まずタッチ式お絵描きソフトウェアをH8マイコン基板(MB)に書き込んで動かしてみます。手順を説明します。

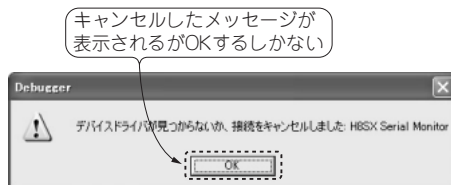
- (1) paint.hwsをダブルクリックし、HEWのプロジェクト・ワークスペースを開きます(図3)。



(a) HEWの起動



(b) 起動後のシリアル・デバッガ接続の設定はキャンセル



(c) シリアル・デバッガをキャンセルしたアラート

図3 プロジェクト・ワークスペース・ファイルでHEWを起動  
(b)は表示されないこともある

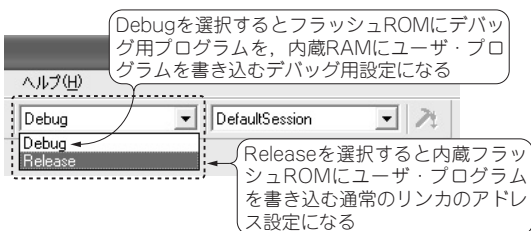


図4 Releaseコンフィギュレーションの設定

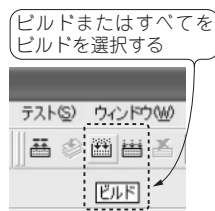


図5 ビルドの実行



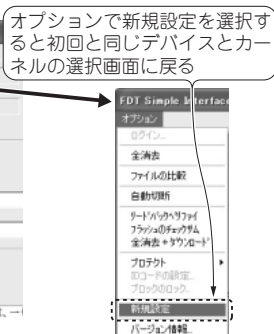
図6 FDTの起動(Basicモード)



(a) 初回の起動時はデバイスとカーネルの選択が表示される



(b) 2回目以降の起動時は前回の画面が表示される



(c) 新規設定オプションでデバイス選択へ戻る

図7 FDTの初期設定

デバッガ(詳細は Appendix 4 参照)が起動してMBへの接続メニューが表示されます。通常のフラッシュ・メモリへのプログラム書き込みの場合は「キャンセル」を選択してください。Appendix 4のシリアル・デバッガを利用する場合は「接続」します。

- (2) 自律動作のコンフィギュレーション Releaseを選択します(図4)。

もう一度シリアル・デバッガが起動し、MBへの接続メニューが表示された場合は「キャンセル」してください。

- (3) 図5のようにビルド(コンパイル, アセンブル, リンク)します。

- (4) プログラムをフラッシュ・メモリに書き込むツールFDTをBasicモードで起動します(図6)。

初めての場合は設定画面から起動します。2回目以降は前回の設定が残った状態で起動します。2回目以

(a) デバイスはH8SX, TypeはGeneric BOOT Deviceを選択

(b) USB Directを選択

(c) H8SXをブート・モードで起動

(d) 表示されるH8SXのUSBを選択

(e) FDTがH8SXと通信し情報を収集

(f) H8SXのクロック・モード設定を入力

(g) 情報収集終了

(h) MBのクロック入力を設定

(i) 書き込みオプション

(j) 設定終了

フィルタにH8SXと入力すると, TypeにGeneric BOOT Deviceが表示されるので, 選択し次へ進む

Select portはUSB Directを選択する

H8SX/1655をUSBでパソコンに接続し, ブート・モードで起動しなさいというメッセージ

FDT will now attempt to connect to your generic device. Please ensure the board is connected, powered and in Boot mode.

1 USB device located

VID: 04E8B:PID: 0005: 6441916ca8382

汎用デバイスの確認

Booting Device  
Sending Supported Devices Inquiry...  
Selecting Device  
Sending Clock Mode Inquiry...  
Selecting Clock Mode  
Sending Other Inquiries

H8SX/1655と通信し確認後, 各種の設定を行う

MBとの通信で自動的にH8SX/1655(製品型番のR5F61655)が設定される. クロック・モードは手動で選択した0で, OKして次へ進む

入力クロックはMBボードに実装されている12MHz, メイン・クロックのてい倍比は4を選択する(つまりCPUは12MHz × 4 = 48MHzで動作しているという設定). 間違えるとフラッシュROMに書き込む時間が適正ではなくなり, 書けなかったりダメージが大きく壊れたりする可能性がある

MBの状況であるクロックモード0を設定する

Select a clock 0

汎用デバイスの確認

Booting Device  
Sending Supported Devices Inquiry...  
Selecting Device  
Sending Clock Mode Inquiry...  
Selecting Clock Mode  
Sending Other Inquiries

R5F61655

0

汎用デバイスの確認

デバイス設定用の値を入力してください

[R5F61655] using [Protocol 0]

外部クロックまたは内部クロックを選択し [External Clock] でください

入力クロック 12.000 Mhz

クロックモード 0

メインクロックの倍比 (CKM) 4

周辺クロックの倍比 (CKP) 1

この画面で変更する項目は特にない

Protection  
Automatic Interactive None

書き込み済みブロックへの書き込みを認識した場合は書き込み前に自動的にブロックを消去します。

出力メッセージレベルはどれにしますか?  
Messaging  
Standard Advanced

詳細なメッセージを出力します。

書き込み完了後, リードバックペリファイを実行しますか?  
Readback Verification  
Yes No

ダウンロードとはフラッシュROMに書き込むこと. 場所はUser/Data Area. User Boot Areaは通常使わない

基本の設定が表示されている

FDT Simple Interface (Unsupported Firmware Version)

オプション

BASIC FILE PROGRAMMING

Device: R5F61655 Port: USB Direct

File Selection

Download File

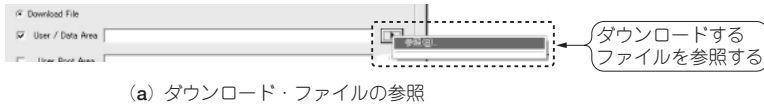
User / Data Area

User Boot Area

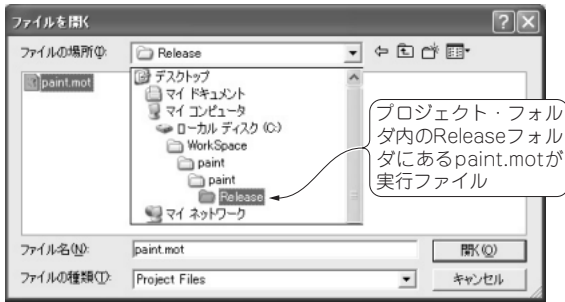
スタート

デバイスの切断

図8 FDTの書き込みオプション設定



(a) ダウンロード・ファイルの参照



(b) ダウンロード・ファイルの指定



(c) 設定完了

図9 ダウンロード(書き込み)ファイルの設定



図10 書き込みと完了

書き込みに失敗する場合は、本書で紹介しているバージョンより新しいバージョンを使うと成功することがある

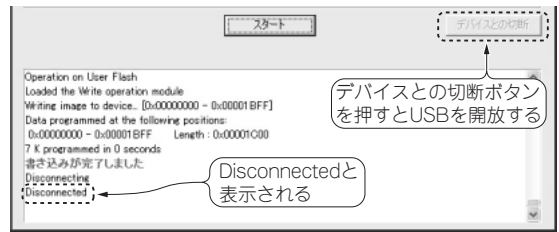


図11 書き込み後の切断

降の起動で、新しく設定しなおしたいなら、「オプション」で「新規設定」を選択します。初めての起動と同じ設定画面が表示されます(図7)。

(5) MBのJ<sub>1</sub>をショート状態にし、H8SX/1655をブート・モードに設定します。電源(USBでもOK)を接続します。

(6) FDTをメニューに従って設定します(図8)。

接続：USB Direct  
 CPU：H8SX/1655で「Generic BOOT Device」  
 モード：0  
 クロック周波数：12MHz  
 周波数倍率：4

(7) FDTの設定でUSER/DATA Areaにpaint.motファイルを選択します(図9)。

(8) 書き込みを開始し、完了を待ちます(図10)。1秒とかかりません。

(9) FDTを終了するかまたは接続を解除します(図11)。

(10) MB電源を切ってジャンパJ<sub>1</sub>をオープンにします。

(11) MBの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)。図1のようなタッチ式お絵描きソフトウェアの画面が表示されます。

## グラフィックス描画プログラミング

### ● 「点」を描く：すべての図形の基本

液晶ディスプレイの画面は2次元配列で管理します。それぞれのドット・データを格納するメモリ・アドレスは図12に示すようになっています。数学上は(x, y)の順ですが配列のインデックス指定は(y, x)の順です。こうすることで画面の横方向をメモリ上、順に利用することになります。横方向に連続アドレスとなっているため、横方向の直線は高速に書き込みで

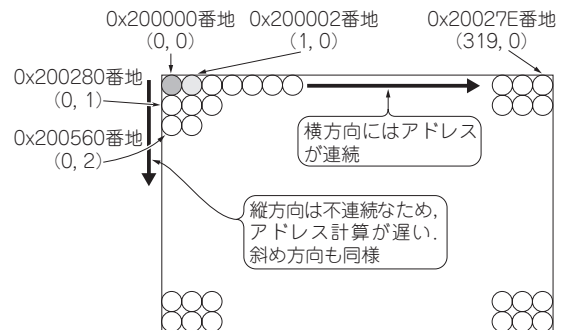


図12 液晶ディスプレイの表示画面とドット・データを格納するアドレスの関係



リスト1 「点」 描画プログラム (lcd\_lib.c)

```

/*
点描画プログラム
  入力：描画点(x, y), 色 color [16ビット(R:G:B=5:5:6)], 描画モード opm(0:置換, 1:AND, 2:OR, 3:EOR)
  出力：なし
*/
void dot(int x, int y, unsigned short color, int opm)
{
    switch(opm)
    {
        case 0:
            global_lcd_framebuffer[y][x] = rgb16_color[color]; break;
        case 1:
            global_lcd_framebuffer[y][x] &= rgb16_color[color]; break;
        case 2:
            global_lcd_framebuffer[y][x] |= rgb16_color[color]; break;
        case 3:
            global_lcd_framebuffer[y][x] ^= rgb16_color[color]; break;
        default:
            ;
    }
}

```

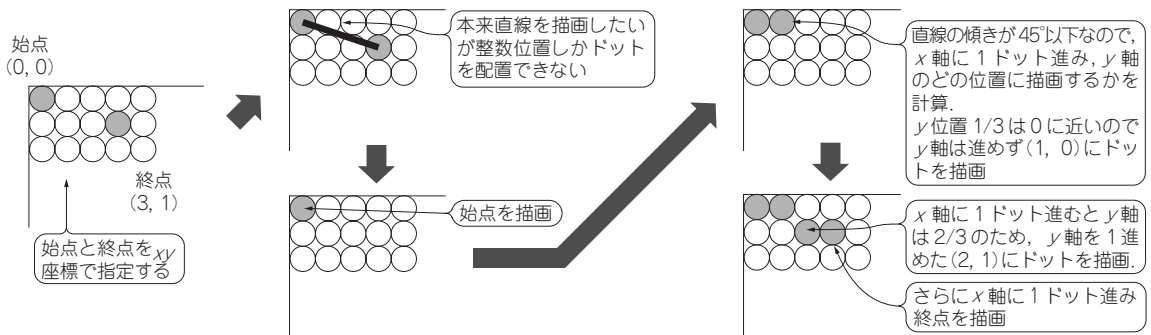


図13 直線描画で塗りつぶすドット

きます。逆に縦方向や斜め方向ではアドレスが連続せず、描画に時間がかかります。

ドット(点)を描画するにはリスト1に示すように、配列のインデックス(要素)に色データを代入します。これで色の置換ができますが、置換前の色にANDやORなどの論理演算もできるようにしました。なぜかは後で説明します。

●「直線」の描画：中間値は四捨五入してどちらかのドットに表示させる

直線は開始点(x<sub>s</sub>, y<sub>s</sub>)と終点(x<sub>e</sub>, y<sub>e</sub>)を指定することでその間を点(ドット)で結びます。とはいえ、ドットの配列は整数ですから、点と点の間には描画できません。

図13のように、例えば開始点(0, 0)から終点(3, 1)の描画では、(0, 0)に描画、次は(1, 1/3)、その次は(2, 2/3)、最後に(3, 1)に描画したいところです。しかし1/3や2/3の位置にはドットがなく描画できません。

解決策としては二つ近傍の整数位置に描画するかまたは近傍の2点に描画色と背景色を距離に合わせて混

ぜた色を描画する方法です。近傍の整数位置に描画する方法は、1/3は0に近い、2/3は1に近いので、(0, 0), (1, 0), (2, 1), (3, 1)と描画します。

この処理をリスト2に示します。

● 小数の演算や割り算はできるだけ避ける

直線の描画演算は1未満の数値を扱いますから浮動小数点(float)で行うと素直です。しかし浮動小数点演算ユニットFPUをもたないH8SX/1655ではとても時間がかかります。

H8SX/1655で高速に処理できるように、整数で演算するプログラムに直したのがリスト3です。

H8SX/1655マイコンで、浮動小数点で行うのと整数演算で行うのはどれほど実行時間に差が出るでしょ

表2 H8SX/1655マイコンは浮動小数点演算ユニットをもたないので整数演算を行う方が6倍早い (0, 0)-(3, 1)の直線の描画演算の例

演算の種類	クロック数	50MHz換算での時間
浮動小数点	3125	62.5μs
整数	471	9.42μs

## リスト2 浮動小数点演算による直線描画プログラム(lcd\_lib.c)

```

void line(int x1,int y1,int x2,int y2,unsigned short color,int opm)
{
    int i;
    int dx,dy;
    float sx,sy,cp;
    //座標 (0, 0) - (3, 1) に直線を描画する例
    dx = ( x2 > x1 ) ? x2 - x1 : x1 - x2; // 始点と終点の相対距離を求める dx=3
    dy = ( y2 > y1 ) ? y2 - y1 : y1 - y2; // 始点と終点の相対距離を求める dy=1
    if( dx > dy ){ //傾きが45度未満 ← こちらを利用
        sy = (float)dy / dx;
        cp = y1;
        for(i=0 ; i<=dx ; i++){
            dot(x1,y1,color,opm);
            x1++;
            cp += sy;
            y1 = cp + 0.5;
        }
    }else{ //傾きが45度以上
        sx = (float)dx / dy;
        cp = x1;
        for(i=0 ; i<=dy ; i++){
            dot(x1,y1,color,opm);
            y1++;
            cp += sx;
            x1 = cp + 0.5;
        }
    }
}

```

## リスト3 整数演算による直線描画プログラム(lcd\_lib.c)

```

void line(int x1,int y1,int x2,int y2,unsigned short color,int opm)
{
    int i;
    int dx,dy,sx,sy,E;
    // 座標 (0, 0) - (3, 1) に直線を描画する例
    dx = ( x2 > x1 ) ? x2 - x1 : x1 - x2; // 始点と終点の相対距離を求める dx=3
    dy = ( y2 > y1 ) ? y2 - y1 : y1 - y2; // 始点と終点の相対距離を求める dy=1
    sx = ( x2 > x1 ) ? 1 : -1; //描画方向を決定 sx=1
    sy = ( y2 > y1 ) ? 1 : -1; // sy=1, 右肩上がり
    if( dx > dy ){ //傾きが45度未満 ← こちらを利用
        E = -dx; //初期偏差をEとする E=-3
        for(i=0 ; i<=dx ; i++){ //相対距離だけ点描画する繰り返し ← 0~3まで4回繰り返し
            dot(x1,y1,color,opm); //点描画
            x1 += sx; //描画点をxに+1移動 x1=0+1=1
            E += 2 * dy; //ドット位置誤差を計算 E=-3+2*1=-1
            if( E >= 0 ){ //yに+1移動するかを確認
                y1 += sy; //移動
                E -= 2 * dx; //移動したら偏差を修正
            }
        }
    }else{ //傾きが45度以上
        E = -dy;
        for(i=0 ; i<=dy ; i++){
            dot(x1,y1,color,opm);
            y1 += sy;
            E += 2 * dx;
            if( E >= 0 ){
                x1 += sx;
                E -= 2 * dy;
            }
        }
    }
}

```

うか？

表2に実験結果を示します。浮動小数点演算より整数演算の方が3125/471 = 6.6倍も高速になります。

この他の処理もできるだけFPUのないH8SXでは整数演算になるようアルゴリズムを考えましょう。あつ、もう一つ高速化において大事なルールがありま

## リスト4 円(楕円)描画プログラム(lcd\_lib.c)

```
void crcl(int xs,int ys,int xe,int ye,unsigned short color,int opm)
{
    int x, y, a, b, xc, yc;
    int d;
    int F, H, r;
    xc = xs ; yc = ys ; //中心点のx座標を求める
    xc = (xe - xc)/2 + xs ;
    a = (xe > xc) ? xe - xc : xc - xe ; //aを求める
    yc = (ye - yc)/2 + ys ; //中心点のy座標を求める
    b = (ye > yc) ? ye - yc : yc - ye ; //bを求める
    r = a * b;
    x = a; //描画開始点
    y = 0;
    d = b * r;
    F = -2 * d + b * b + 2 * a * a;
    H = -4 * d + 2 * b * b + a * a;
    //上下左右対称のため楕円の1/4を演算し、4点を描画
    while( x >= 0 ){
        dot(xc + x,yc + y,color,opm); //4点を描画
        dot(xc - x,yc + y,color,opm);
        dot(xc + x,yc - y,color,opm);
        dot(xc - x,yc - y,color,opm);
        if( F >= 0 ){
            x--;
            F -= 4 * b * b * x;
            H -= 4 * b * b * x - 2 * b * b;
        }
        if( H < 0 ){
            y++;
            F += 4 * a * a * y + 2 * a * a;
            H += 4 * a * a * y;
        }
    }
}
```

## リスト5 塗りつぶしの円(楕円)プログラム(lcd\_lib.c)

```
void forcl(int xs,int ys,int xe,int ye,unsigned short color,int opm)
{
    int x, y, a, b, xc, yc;
    int d;
    int F, H, r;
    xc = xs ; yc = ys ; //中心点のx座標を求める
    xc = (xe - xc)/2 + xs ;
    a = (xe > xc) ? xe - xc : xc - xe ; //aを求める
    yc = (ye - yc)/2 + ys ; //中心点のy座標を求める
    b = (ye > yc) ? ye - yc : yc - ye ; //bを求める
    r = a * b;
    x = a; //描画開始点
    y = 0;
    d = b * r;
    F = -2 * d + b * b + 2 * a * a;
    H = -4 * d + 2 * b * b + a * a;
    //上下左右対称のため楕円の1/4を演算し、直線で結ぶ
    while( x >= 0 ){
        line(xc - x,yc + y,xc + x,yc + y,color,opm); //塗りつぶしのため線描画
        line(xc - x,yc - y,xc + x,yc - y,color,opm);
        if( F >= 0 ){
            x--;
            F -= 4 * b * b * x;
            H -= 4 * b * b * x - 2 * b * b;
        }
        if( H < 0 ){
            y++;
            F += 4 * a * a * y + 2 * a * a;
            H += 4 * a * a * y;
        }
    }
}
```



す。それは、除算は低速だ、ということです。除算アルゴリズムは乗算アルゴリズムに変換できないかを検討しましょう。どのコンピュータも除算は得意ではありませんから。

### ● 楕円(円)の描画と塗りつぶし

楕円(円)の描画と塗りつぶしについて紹介します。楕円は方程式があります。しかしこれも浮動小数点演算となり遅いですから、整数で演算します。プログラムをリスト4に示します。

楕円の周囲の点分かれば、その間を直線で塗ること塗りつぶし演算ができます(リスト5)。

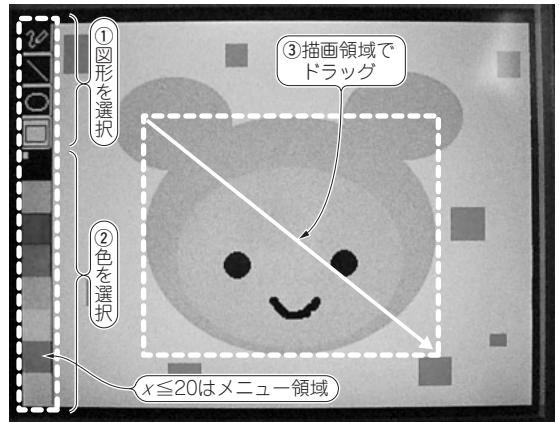


図14 図形の種類と色を選択してキャンバス領域でドラッグすると図形が描ける

## タッチ式お絵描きソフト本体のプログラミング

タッチ式お絵描きソフトウェア本体の処理を解説します。図14に示すように、左に描画図形選択と描画色の選択のメニューを表示しています。図形と色を選んでからキャンバス領域でタッチ&ドラッグすることで描画します。

タッチ位置を取得する考え方は「第9章 タッチ入力付きグラフィック・カラー液晶パネル基板を作る」に紹介されています。ここではペンがタッチされている間は「試し描き」、離れたら「確定」する動作の実現方法を紹介します。

### ● メイン・フロー：まずメニューか描画かを判断

メイン側の動作フローを図15に示します。

タッチを検出すると、メニュー領域かキャンバス領域かをx座標で判定します。

メニュー領域ならメニューを決定し、タッチが終了するまで待ちます。

キャンバス領域なら図形を描画しますが、下地の色

と描画する色をEXORして図形を仮描画します。タッチが継続しかつ位置が変更されたら図形を同じ色でEXORすることで下地の色に戻して消します。タッチが終了したら図形を本来の色で上書きし確定します。動作の詳細を以下に説明します。

### ■ 「仮の描画」を実現する方法

直線で動作を説明しましょう。タッチした最初の位置が(100, 100)で、タッチしたまま(200, 200)にペンが移動した場合、試し描きで(100, 100)-(200, 200)に直線を描画します。タッチしたままの状態でペン位置が(200, 210)に変わったら(100, 100)-(200, 200)を元の下絵に戻し、(100, 100)-(200, 210)の描画をします。

この場合(100, 100)-(200, 200)の直線を描画する前にあった絵を戻す必要があります。これには図16に示す二つの方法があります。一つは(100, 100)-(200, 200)の方形領域を別のメモリ領域に退避し保管

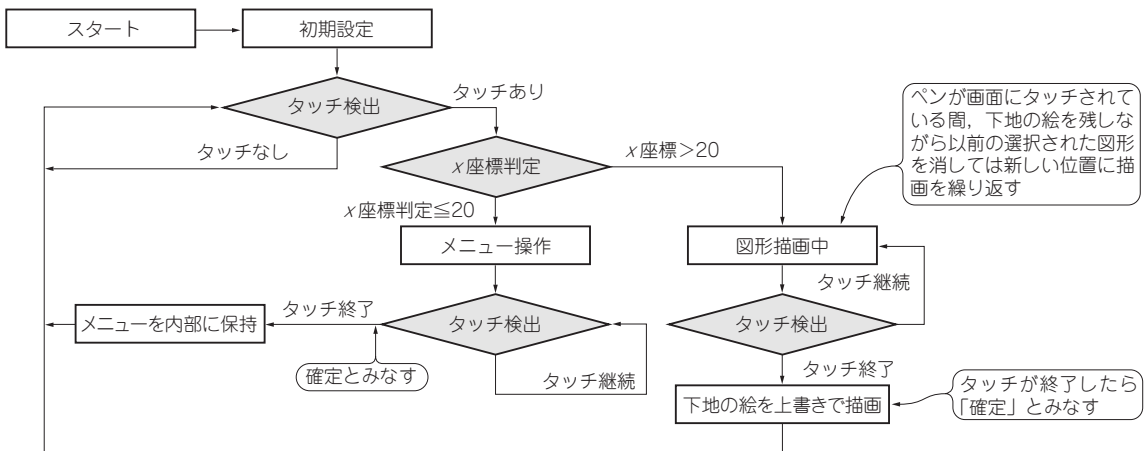


図15 H8プログラムのメイン・フロー

付属 CD-ROM に表 A に示す描画ライブラリ関数を使える graphic プロジェクトを収録しています。関数はグラフィックス描画に使用します。main

関数には呼び出しサンプルが記述されているので参考にしてください。

表 A 用意した描画ライブラリ

機能	関数	引き数	返値	使用例
点描画	dot(int x,int y,unsigned short color,int opm)	int x:座標x軸0～319, int y:座標y軸0～239, unsigned short color:描画の色指定65535色(以下はdefine済み 0:黒, 1:赤, 2:緑, 3:青, 4:黄, 5:紫, 6:薄青, 7:白), int opm:描画方式(0:上書き, 1:AND, 2:OR, 3:EOR)	なし	dot(100,150,0,0); ←座標(100, 150)に黒い点を描画
指定点の色取得	get_dot(int x, int y)	int x:座標x軸0～319, int y:座標y軸0～239	(unsigned short) RGBデータ	x = get_dot(100, 150); ←座標(100, 150)の色を取得して変数xに格納
線描画	line(int x1, int y1, int x2, int y2, unsigned short color, int opm)	int x1:始点x0～319, int y1:始点y0～239, int x2:終点x0～319, int y2:終点y0～239, unsigned short color:描画の色指定65535色(以下はdefine済み 0:黒, 1:赤, 2:緑, 3:青, 4:黄, 5:紫, 6:薄青, 7:白), int opm:描画方式(0:上書き, 1:AND, 2:OR, 3:EOR)	なし	line(50,60,150,160,0,0); ←始点(50, 60)と終点(150, 160)を結ぶ線を描画
円描画	crcl(int x1, int y1, int x2, int y2, unsigned short color, int opm)	int x1:始点x0～319, int y1:始点y0～239, int x2:終点x0～319, int y2:終点y0～239, unsigned short color:描画の色指定65535色(以下はdefine済み 0:黒, 1:赤, 2:緑, 3:青, 4:黄, 5:紫, 6:薄青, 7:白), int opm:描画方式(0:上書き, 1:AND, 2:OR, 3:EOR)	なし	crcl(50,60,150,160,0,0); ←始点(50, 60)と終点(150, 160)からなる四角形に内接する楕円を描画
塗り潰し円描画	fcrcrcl(int x1, int y1, int x2, int y2, unsigned short color, int opm)	int x1:始点x0～319, int y1:始点y0～239, int x2:終点x0～319, int y2:終点y0～239, unsigned short color:描画の色指定65535色(以下はdefine済み 0:黒, 1:赤, 2:緑, 3:青, 4:黄, 5:紫, 6:薄青, 7:白), int opm:描画方式(0:上書き, 1:AND, 2:OR, 3:EOR)	なし	fcrcrcl(50,60,150,160,0,0); ←始点(50, 60)と終点(150, 160)からなる四角形に内接する楕円を描画+塗り潰し
四角形描画	rect(int x1, int y1, int x2, int y2, unsigned short color, int opm)	int x1:始点x0～319, int y1:始点y0～239, int x2:終点x0～319, int y2:終点y0～239, unsigned short color:描画の色指定65535色(以下はdefine済み 0:黒, 1:赤, 2:緑, 3:青, 4:黄, 5:紫, 6:薄青, 7:白), int opm:描画方式(0:上書き, 1:AND, 2:OR, 3:EOR)	なし	rect(50,60,150,160,0,0); ←始点(50, 60)と終点(150, 160)からなる四角形描画
塗り潰し四角形描画	frct(int x1, int y1, int x2, int y2, unsigned short color, int opm)	int x1:始点x0～319, int y1:始点y0～239, int x2:終点x0～319, int y2:終点y0～239, unsigned short color:描画の色指定65535色(以下はdefine済み 0:黒, 1:赤, 2:緑, 3:青, 4:黄, 5:紫, 6:薄青, 7:白), int opm:描画方式(0:上書き, 1:AND, 2:OR, 3:EOR)	なし	frct(50,60,150,160,0,0); ←始点(50, 60)と終点(150, 160)からなる四角形描画+塗り潰し

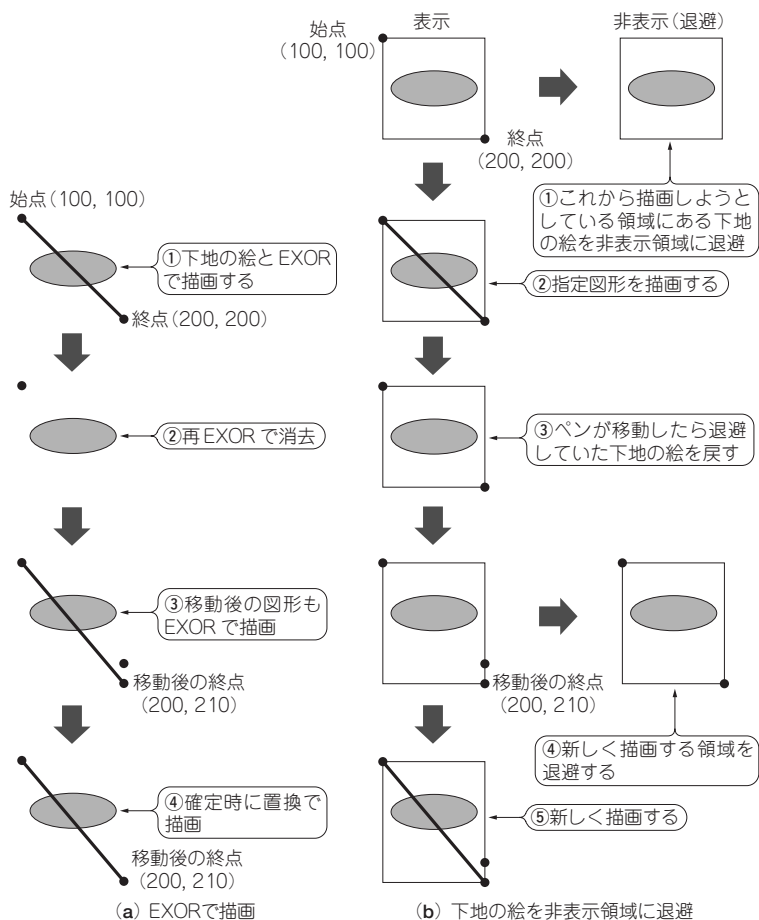


図 16 試し描きと元に戻す方法

します。もう一つの方法は下絵に対して描画したい色を EXOR で試し描き、元に戻すときはもう一度描画したい色で EXOR します。

● もともとの図形を覚えておく方法は時間がかかる

方形領域を扱う方法は処理時間がかかります。例えば退避する領域のドット数が  $300 \times 200 = 60,000$ 、外部 SRAM のバスが 6MHz とした場合、 $6\text{MHz}/60,000$  ドット = 100Hz、10ms でリード、退避する領域も外部 SRAM ならライトも 10ms で合計 20ms の時間です。描画に利用できる時間(プランキング期間など)が全体の 10% (1/10) とした場合で 200ms、1 秒間で 5 回の書き直ししかできず、かなりごちなくなります。

● EXOR で試し描きして 2 度目の EXOR で元に戻す

もう一つの EXOR で試し描きする方法は 100 ドットの直線なら 100 ドットのリードとライトの計 200 ドットのメモリ・アクセスで済みますから、バスが 6MHz、描画期間が 10% と想定した場合にも

6MHz/200 ドット/10 = 3kHz、座標演算時間を考慮しても 1kHz 以上で十分な速度が得られます。

応用のヒント：「UNDO」と「SAVE」を作る

■ 「UNDO」を実現するには

お絵描きソフトウェアで遊んでいると、あっ開始点の位置を間違えた、あ～やっちゃったよ、「元に戻す」(UNDO, CTRL + Z)があれば便利なのに、と思いますよね。ではその「元に戻す」を追加する方法を紹介します。UNDO は、最後に実行した描画処理をなかったことにする、という役目を持ち、二つの方法が考えられます。

● 前の画像を退避しておく

一つ目の方法は、描画処理を実行する前の状態を非表示領域にバックアップして保存し、UNDO を要求されたら保存した状態を新たに描画します。バック

リスト6 UNDO プログラム

```
//描画領域全体をバックアップするには 140K バイトの領域が必要
short i, j;
//バックアップ
for( j = 0; j < 240;j++ )
{
    for( i = 20; i < 320; i++)
        bakup_buffer[j][i] = global_lcd_framebuffer[j][i];
}
//リストア
for( j = 0; j < 240;j++ )
{
    for( i = 20; i < 320; i++)
        global_lcd_framebuffer [j][i] = bakup_buffer[j][i];
}
```

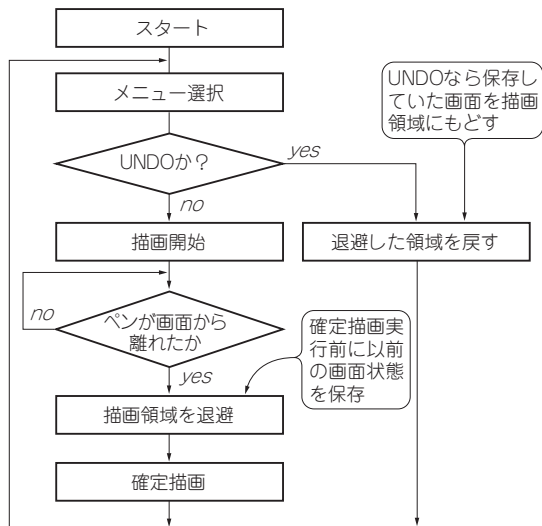


図 17 一つ前の状態を退避しておいて UNDO を実現できる

アップするタイミングは、最後にタッチを離れた後で、確定描画する前です(図 17)。

バックアップする領域は、描画領域全体か、描画の開始点から終了点までの方形領域とします。プログラムをリスト6に紹介します。

描画した方形領域だけをバックアップするなら、領域の情報も併せて保存します。ここで紹介したのは1回のUNDOですが、バックアップ領域を増やせば何回でもできます(メモリの使用量からTBでは現実的ではない)。

● 前の描画処理そのものを覚えておく

二つ目の方法は、描画処理そのものを保存する方法です。処理の番号と処理のパラメータ(引き数)をバックアップします。この方法の長所はバックアップ量が少ないことです(図 18)。コマンド数が少ない場合はH8SX/1655の内蔵RAMも使えます。短所は、すべての描画処理を記憶する必要があるため、描画回数が増えると最初から実行するため、描画時間がかかること

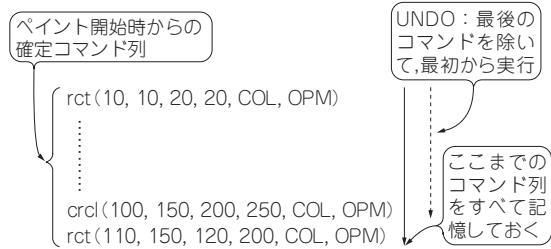


図 18 最初から最後までのコマンド列をすべて覚えておく

リスト7 UNDO で描画処理を覚えておくために必要なバッファ

```
struct cmd_buf{
    short command;
    short param1;
    :
    short param7;
};
struct cmd_buf command_buffer[10];
```

です。

10個の描画処理を使えるようにするには、リスト7のように10個のバッファを準備します。

■ 「SAVE」を実現するには

SAVEを実現するなら、拡張基板SBに取り付けられるSDカードに保存します。H8SX内蔵フラッシュ・メモリでも構いませんが、保存中はフラッシュ・メモリを読み出せないため、RAMでプログラムを実行するなど工夫が必要なためです。やり方はUNDOと同じく二つの方法が考えられます。一つは画面そのものをハードコピーする方法、もう一つは描画処理そのものを保存しておく方法です。

\*

その他にWindows標準で付いているアクセサリの「ペイント」では、ペイント缶による閉領域の塗りつぶしやスプレーなどのツールがあり、これらも実装できます。チャレンジしてみたいはいかがでしょうか。

〈藤澤 幸穂〉

# 本書サポート・ウェブ・サイト

http://toragi.cqpub.co.jp/tabid/284/Default.aspx

トランジスタ技術のウェブ・サイト  
(http://toragi.cqpub.co.jp/)の記事サポートから本書のサポート・ページを見つけられます

増補版の情報を更新していきます

目次や見本ページ、表紙が確認できます

関連企画：  
トランジスタ技術2010年4月号  
特集「はじめてのH8マイコン」

重要なお知らせ

付属 H8 マイコン基板で  
できることが動画や記事  
でわかります

編集部からのお知らせを  
掲載しています

さらに興味を広げたい場合  
●他の H8 関連書籍  
●他の USB に挿するだけマイコン  
●タッチ・パネルの使い方  
●上位機種 SH マイコン

ウェブ・サイトの内容は予告なく変わることがあります。