RX ファミリ
DSP 機能命令を活用した画像フィルタプログラム

R01AN0228JJ0100
Rev.1.00
2011.03.14

要旨

この文書は、RX ファミリの DSP 機能命令を使用した画像フィルタプログラム例を説明します。

動作確認デバイス

RX ファミリ

目次

1. はじめに

RX ファミリ CPU コア (以下 RX と略) は 16 ビット × 16 ビットの積和器を搭載しています。乗算を用いた演算式やアドレス計算で通常に用いる 32 ビット × 32 ビットの整数乗算命令 (MUL 命令) は、32 ビット × 32 ビットの演算結果 64 ビットのうち下位 32 ビットをその演算結果とします。つまり、MUL 命令の使用にあたっては、演算結果が 32 ビットを越えないという前提があります。ところが、数値データを固定小数点表現 (例えば、[1]を参照ください) で表す場合、乗算あるいは積和演算の結果のうち、有効なデータは上位側に位置するのが普通です。そのため、固定小数点表現を用いた数値データの乗算あるいは積和演算の場合に MUL 命令を用いていたのでは、演算結果が 32 ビット以内に納まる場合しか用いることができず、狭い範囲の数値データしか表現できなくなるという問題が発生します。この問題を解決するために、RX は 48 ビットのアキュムレータによる積和演算命令 (または乗算命令)、アキュムレータに格納された値の丸め演算を実行する命令、およびアキュムレータと汎用レジスタ間のデータの転送命令をサポートしています。これらの積和演算命令や丸め命令等を組み合わせることで、固定小数点表現を用いた数値データの種々の演算を高速に実現でき、DSP に匹敵するデータ処理能力を実現することができます。RX の積和演算命令の詳細は「RX ファミリ ユーザーズマニュアル ソフトウェア編 (RJJ09B0465)」を参照ください。アプリケーションノート「積和演算命令の活用方法」(R01AN0254JJ) では、これらの積和演算命令や丸め命令の使い方を説明しています。また、アプリケーションノート「積和演算の組み込み関数活用方法」(R01AN0255JJ) では、これらの積和演算命令や丸め命令を RX ファミリ C/C++コンパイラ (以下コンパイラと略) の拡張機能である組み込み関数（コンパイラ V1.01 からサポートされます）から活用する方法を説明しています。

以下では、RX の積和演算命令を活用した画像フィルタプログラムについて説明します (画像フィルタの理論的な詳細については、例えば[2]のようなテキストを参照ください)。なお、サンプルプログラムでは積和演算命令をコンパイラの組み込み関数から活用する方法をとります。

【注】 [1] 森、名取、鳥居; "岩波講座　情報科学-18　数値計算", pp.1-27, 岩波書店, (1982)
[2] R. C. Gonzalez et al.; "Digital Image Processing", 3rd edition, Pearson Education International, (2008)

2. 画像フィルタ

本アプリケーションノートでは、次の図 1 に示すように、画像 f 上のピクセル f(x, y)とその周囲の合計 9 個のピクセルに対して 3 × 3 のフィルタマスクの係数を乗算する形の画像フィルタについて考えます。

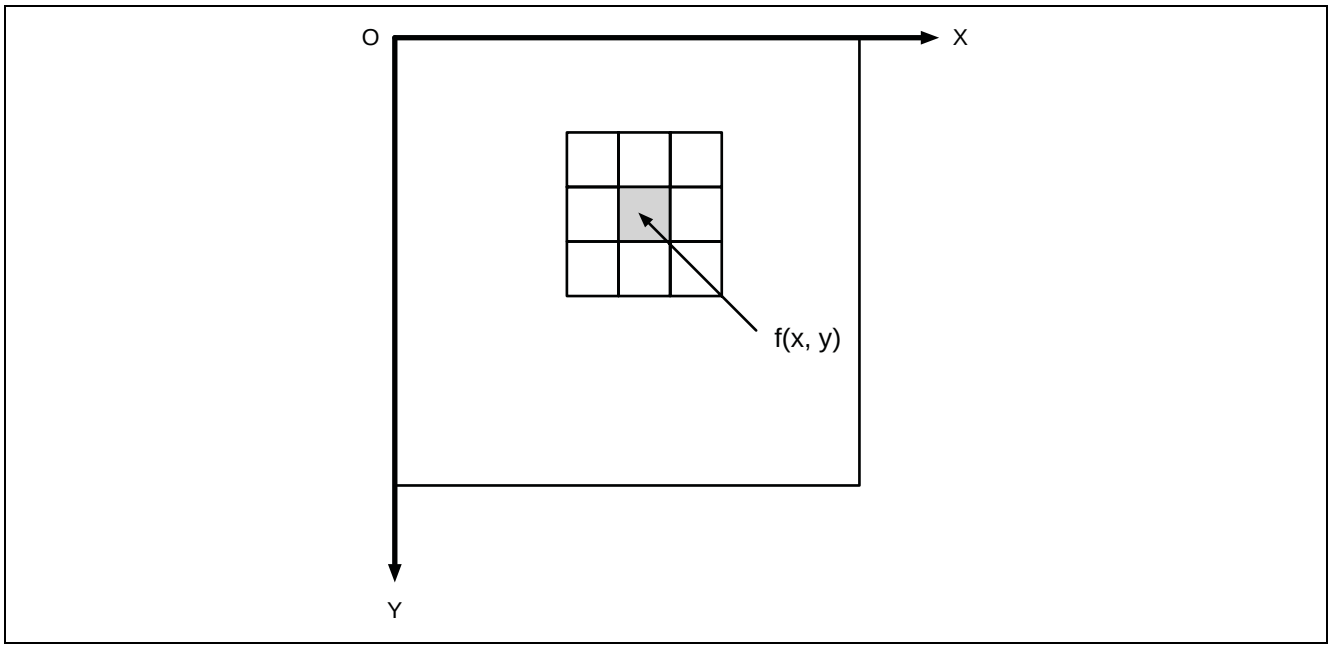

図 1　画像 f への 3 × 3 のフィルタマスクの適用

次の図 2 に、画像 f 上のピクセルと 3 × 3 のフィルタマスク w の係数との対応関係を示します。

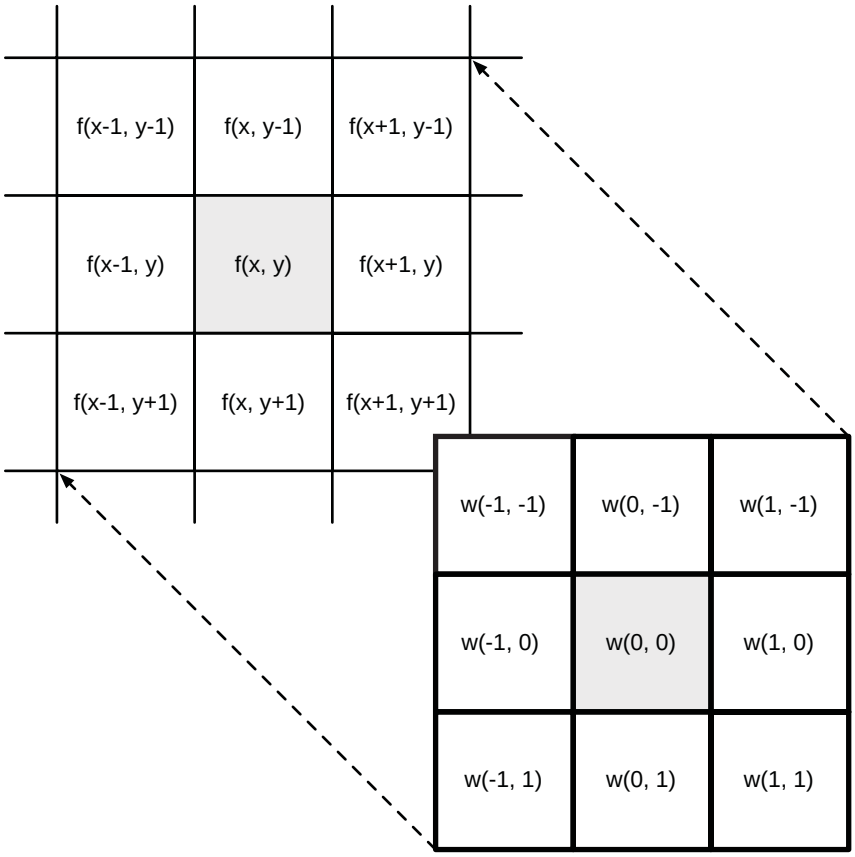

図 2　画像 f のピクセル f(x,y)と 3 × 3 のフィルタマスクの係数 w(s,t)の対応関係

入力画像のピクセルを f(x, y)、3 × 3 のフィルタマスクの係数を w(s, t)とすると、フィルタ処理の出力画像のピクセル g(x, y)は次のように表すことができます。

$$g(x,y) = \sum_{s=-1}^{1} \sum_{t=-1}^{1} w(s,t) \times f(x+s, y+t)$$

これは、簡単にいえば、ピクセルとマスクの係数を順番に掛けて合計するだけの処理ですから、RX の積和演算命令を活用することができます。この章では、上述のメカニズムを使って実現できる画像フィルタの例をいくつか説明します。なお、フィルタの説明では図 3 に示すサンプル画像を使用します。

図 3　サンプル画像

2.1　平滑化フィルタ

本節では、画像の階調を滑らかにする平滑化フィルタについて説明します。平滑化フィルタは、周囲のピクセル値との平均をとることで画像の階調を滑らかにします。平滑化フィルタは、画像をぼかす、ノイズを取り除くなどの目的に使用されます。

図 4 に平滑化フィルタのマスクの代表例を二つ示します。左側のフィルタは、単純に 3 × 3 のマスク内部の相加平均をとるものです。これに対して、右側のフィルタは中心に近いピクセルの値に重みを付けて平均をとる一例です。どちらのフィルタマスクも、平均を計算するために、フィルタ処理の結果を係数の合計で割算する必要があります。

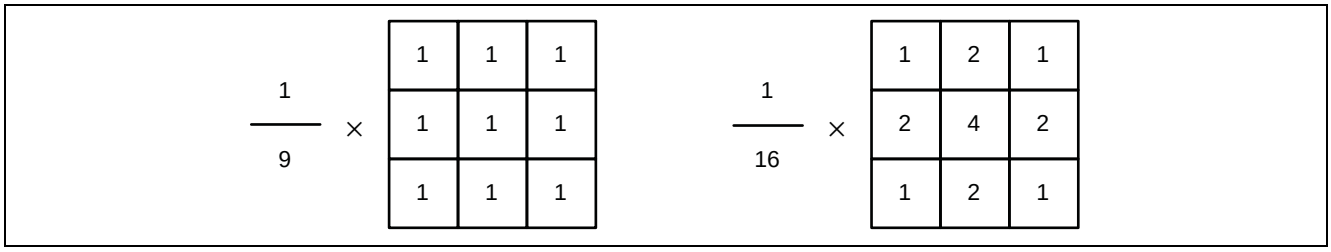

図 4　平滑化フィルタのマスク (左: 相加平均、右: 重み付き平均の例)

上の平滑化フィルタをサンプル画像に適用した例を図 5 に示します。

図 5　平滑化フィルタの出力例 (左: 相加平均、右: 重み付き平均の例)

2.2　エッジ検出フィルタ

本節では、画像のエッジを検出するためのフィルタとして、Sobel と Prewitt の二種類のフィルタについて説明します。エッジ検出とは、簡単にいえば、画像の階調の勾配が急な部分を強調/抽出する画像処理に相当します。

最初に、Sobel として知られるフィルタマスクを図 6 に示します。マスクは水平方向のエッジを検出するものと、垂直方向のエッジを検出するものの二種類があります。

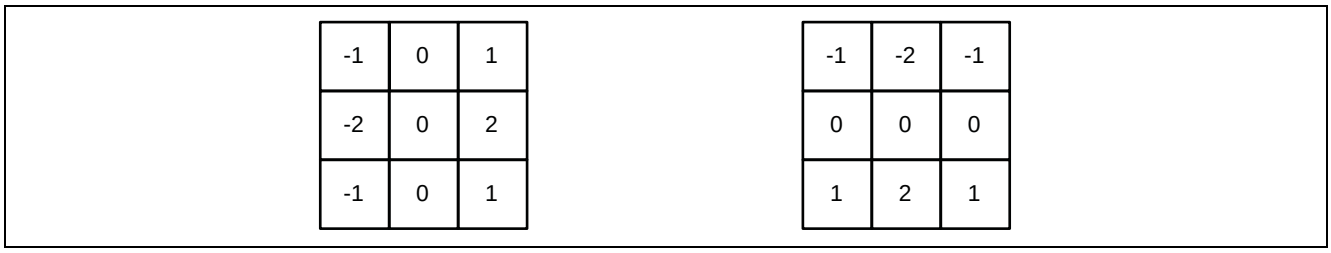

図 6　Sobel フィルタのマスク (左: 水平方向、右: 垂直方向)

上の Sobel フィルタをサンプル画像に適用した例を図 7 に示します。

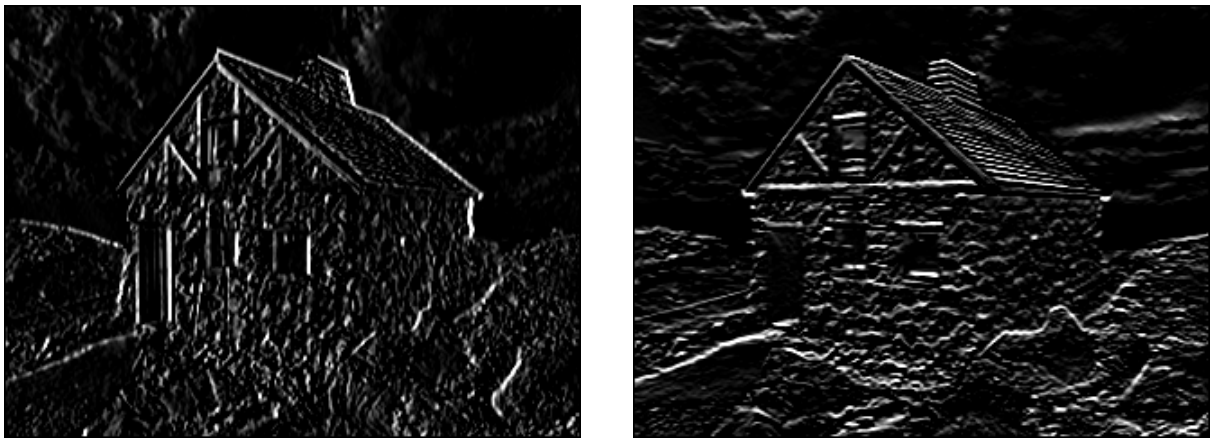

図 7　Sobel フィルタの出力例 (左: 水平方向、右: 垂直方向)

次に、Prewitt として知られるフィルタマスクを図 8 に示します。Sobel と同様に、マスクは水平方向のエッジを検出するものと、垂直方向のエッジを検出するものの二種類があります。

-1	0	1
-1	0	1
-1	0	1

-1	-1	-1
0	0	0
1	1	1

図 8　Prewitt フィルタのマスク (左: 水平方向、右: 垂直方向)

上の Prewitt フィルタをサンプル画像に適用した例を図 9 に示します。

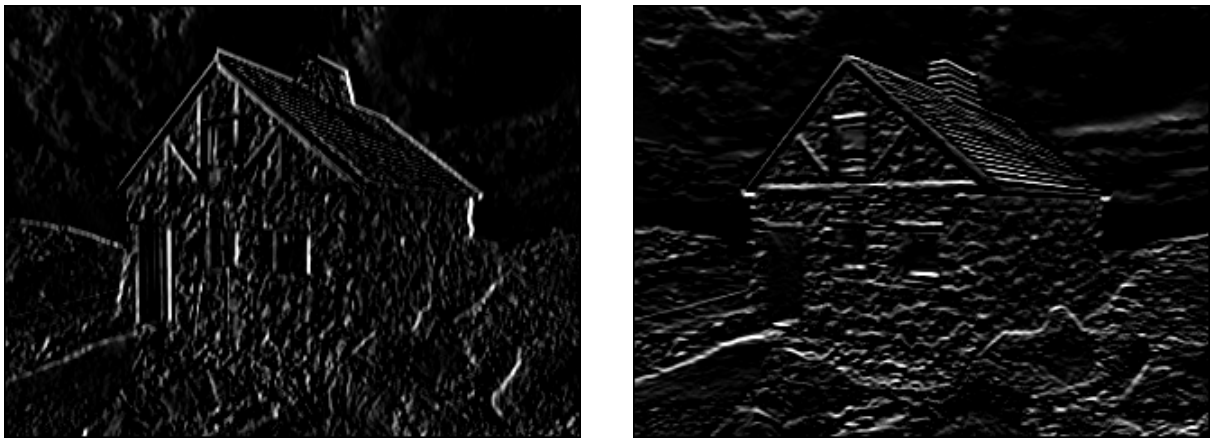

図 9　Prewitt フィルタの出力例 (左: 水平方向、右: 垂直方向)

2.3　鮮鋭化処理 (ラプラシアン・フィルタ)

本節では、ラプラシアン・フィルタ (the Laplacian filter) を使った画像の鮮鋭化処理 (unsharp masking) について説明します。まず、ラプラシアン・フィルタのマスクの一例を図 10 に示します。

-1	-1	-1
-1	8	-1
-1	-1	-1

図 10　ラプラシアン・フィルタのマスク

上のラプラシアン・フィルタをサンプル画像に適用した例を図 11 に示します。ただし、図に示した画像はピクセル値を適宜スケーリングしてあります。

図 11　ラプラシアン・フィルタの出力例 (スケーリング処理後)

ラプラシアン・フィルタは、簡単にいえば、元画像のピクセル値から周辺ピクセルの平均値 (平滑化した結果に等しい) 引くことにより階調の変化 (輪郭) を取り出します。ラプラシアン・フィルタによって取り出した輪郭を元の画像に加えることにより、画像の鮮鋭度を高めることができます。ただし、実際のプログラムの作成に当たっては、ラプラシアン・フィルタの出力は元画像のピクセル値の最大値・最小値の範囲を超える可能性があること、また、ラプラシアン・フィルタの出力を元の画像に加える場合にはピクセル値の最大値を超える可能性があることに留意する必要があります。

サンプル画像にラプラシアン・フィルタによる鮮鋭化処理を適用した例を図 12 に示します。

図 12　先鋭化処理 の例 (左: 元の画像、右: 処理後の画像)

3. 画像フィルタプログラム

本章では、RX の積和演算命令を活用した画像フィルタプログラムについて説明します。

3.1 画像データの構造

本画像フィルタプログラムが扱う画像は、説明を簡単にするために、幅 320 ピクセル、高さ 240 ピクセルのグレイスケール画像とします。画像のピクセルは、RX の積和演算命令を活用するために 16 ビットの符号付き整数で表現し、0 から 255 までのピクセル値 (0 が黒色、255 が白色です) をとるものとします。次に画像サイズを定義しているプログラム部分を示します。

```
/* constant(s) */
#define  WIDTH   320    /* image width */
#define  HEIGHT  240    /* image height */
```

3.2 フィルタマスクの構造

フィルタマスクは、画像データと同様に RX の積和演算命令を活用するために符号付き 16 ビットデータの配列で表現します。フィルタマスクは 3 × 3 の大きさですから、配列に格納すべき係数の数は全部で 9 個です。しかし、同じ行に並んでいる係数データの先頭を 32 ビット境界に配置するため、各行の最後に「埋め草」の要素を置くことにします。これを次の擬似コードで示します。

```
int16_t w[12] = {
   w(-1,-1),  w(0,-1),  w(1,-1), /* padding */ 0,
   w(-1, 0),  w(0, 0),  w(1, 0), /* padding */ 0,
   w(-1, 1),  w(0, 1),  w(1, 1), /* padding */ 0,
};
```

なお、係数は 16 ビットの符号付きの整数あるいは固定小数点データと見なすことができます。この二つはフィルタの種類によって使い分けます。固定小数点データは、図 13 に示すように、b15 と b14 の間に小数点がある符号付き 16 ビットデータで表現します。この場合は、–1.0 から 1.0 までの範囲の値を係数に設定できます。

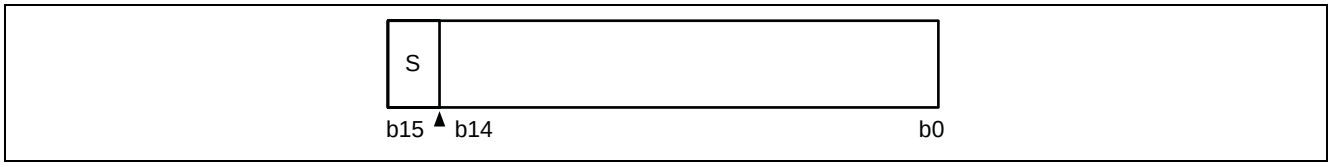

図 13　固定小数点データの形式

3.3 フィルタ処理関数

フィルタ処理関数は、入力画像とフィルタマスクを引数として受け取り、フィルタマスクを適用した結果を別の画像として出力します。以下では、フィルタマスクの係数が整数の場合と固定小数点の場合を分けて考えます。

最初に、固定小数点データの係数マスクを使うフィルタ処理関数 filter_macw1 のプログラムを示します。この関数は、画像 f にマスク mask を適用して得られた画像を g に出力します。

```
/* 画像 f にフィルタ mask を適用した結果を画像 g に出力 (固定小数点係数)  */
#include <machine.h>

void filter_macw1(int16_t *f, int16_t *g, int16_t mask[12])
{
    int x, y;
    int16_t *r1 = f;
    int16_t *r2 = r1 + WIDTH;
    int16_t *r3 = r2 + WIDTH;
    int16_t *ro = g  + WIDTH;

    /* 注意：画像の端 (四辺) のピクセルにはフィルタをかけない */
    for (y = 0; y < HEIGHT - 2; y++) {
        for (x = 0; x < WIDTH - 2; x++) {
            ro[x + 1] = (int16_t) (macw1(r1 + x, mask,     3)
                                 + macw1(r2 + x, mask + 4, 3)
                                 + macw1(r3 + x, mask + 8, 3));
        }
        r1 += WIDTH;
        r2 += WIDTH;
        r3 += WIDTH;
        ro += WIDTH;
    }
}
```

次に整数係数のマスクを使うフィルタ処理関数 filter_macl のプログラムを示します。filter_macl は、filter_macw1 と同様に、画像 f にマスク mask を適用して得られた画像を g に出力します。

```
/* 画像 f にフィルタ mask を適用した結果を画像 g に出力 (整数係数)  */
void filter_macl(int16_t *f, int16_t *g, int16_t mask[12])
{
    int x, y;
    int16_t *r1 = f;
    int16_t *r2 = r1 + WIDTH;
    int16_t *r3 = r2 + WIDTH;
    int16_t *ro = g  + WIDTH;

    /* 注意：画像の端 (四辺) のピクセルにはフィルタをかけない */
    for (y = 0; y < HEIGHT - 2; y++) {
        for (x = 0; x < WIDTH - 2; x++) {
            ro[x + 1] = (int16_t) (macl(r1 + x, mask,     3)
                                 + macl(r2 + x, mask + 4, 3)
                                 + macl(r3 + x, mask + 8, 3));
        }
        r1 += WIDTH;
        r2 += WIDTH;
        r3 += WIDTH;
        ro += WIDTH;
    }
}
```

なお、上に示したフィルタ処理関数は、いずれも入力画像の四辺 (外周) の上にあるピクセルにはフィルタ処理をしません。四辺上のピクセルについては、欠けている周辺ピクセルを何らかの方法で補ってやらないとフィルタ処理ができないからです。プログラムが複雑になるのを避けるため、単純化した仕様になっています。

3.4　ピクセル値のスケーリング関数

画像のピクセル値を適切にスケーリングする関数 equalize を次に示します。この関数は、画像の鮮鋭化処理で使用します。

```
/* 画像 image のピクセル値を 0..255 の範囲にスケーリング */
void equalize(int16_t *image)
{
    int16_t *p;
    int min, max;

    min = max = image[0];
    for (p = image + 1; p < image + WIDTH * HEIGHT; p++) {
        if (*p < min) {
            min = *p;
        }
        if (*p > max) {
            max = *p;
        }
    }
    for (p = image; p < image + WIDTH * HEIGHT; p++) {
      *p = (int16_t) (((*p - min) * 255) / (max - min));
    }
}
```

3.5　画像データの加算関数

2 つの画像のピクセル値を加算する関数 image_add を次に示します。この関数は、前節で説明した関数 equalize 同様、画像の鮮鋭化処理で使用します。

```
/* 画像 f のピクセル値に画像 g のピクセル値を加算 */
void image_add(int16_t *f, const int16_t *g)
{
    int i;

    for (i = 0; i < WIDTH * HEIGHT; i++) {
        *f++ += *g++;
    }
}
```

4. サンプルプログラム

本章では、前章で説明した画像フィルタプログラムのサンプルプログラムを示します。

4.1 サンプルプログラムの実行環境

次にサンプルプログラムの実行環境を示します。

```
/* local variable(s) */
static int16_t buf[WIDTH * HEIGHT];
static int16_t test_image[WIDTH * HEIGHT] = {
#include "image.h"
};

/* bitmap and palette */
uint8_t bitmap[WIDTH * HEIGHT];
const uint32_t palette[256] = {
#include "palette.h"
};

/* 画像 img を bitmap に出力 */
void put_image(const int16_t *img)
{
    int i, c;

    for (i = 0; i < WIDTH * HEIGHT; i++) {
        c = img[i];
        if (c < 0) {
          c = 0;
        } else if (c > 255) {
          c = 255;
        }
        bitmap[i] = (int16_t) c;
    }
}
```

変数 buf と test_image はいずれも画像データを格納する変数です。変数 buf は、作業用の画像バッファとして使用し、主に画像フィルタの出力結果を格納するために使用します。変数 test_image は、サンプル画像のデータを格納しています (初期化データはヘッダファイル"image.h"に定義されています)。

変数 bitmap と palette は、統合開発環境 (High-performance Embedded Workshop) の中で画像を表示するためのビットマップ (8 ビット・インデックス・カラー) です。サンプルプログラムから関数 put_image を呼び出すことにより、指定された画像データを変換して bitmap に書き込みます。変数 palette には、0 から 255 のインデックスに対応したグレイスケールの色情報が RGB888 形式で格納されています (初期化データはヘッダファイル"palette.h"に定義されています)。

bitmap に書き込まれている画像を High-performance Embedded Workshop で表示する方法については次の節を参照してください。

4.2　ビットマップ画像の表示方法

次の方法でビットマップ画像 (bitmap) を High-performance Embedded Workshop のウィンドウに表示できます。

1. High-performance Embedded Workshop のメニューバーから「画面」>「グラフィック」>「画像」メニューを選択します。
2. 「画像プロパティ」ダイアログが開きます (図 14 を参照) 。
3. ダイアログの「色情報」の「モード」から「RGB」を選択します。
4. ダイアログの「ビット/ピクセル」から「8 ビット(Index Color)」を選択します。
5. ダイアログの「データアドレス」にシンボル「_bitmap」を設定します。
6. ダイアログの「パレットアドレス」にシンボル「_palette」を設定します。
7. ダイアログの「幅」に 320 (ビットマップの幅) を設定します。
8. ダイアログの「高さ」240 (ビットマップの高さ) を設定します。
9. 「OK」ボタンをクリックしてダイアログを閉じます。
10. 画像を表示するための High-performance Embedded Workshop のウィンドウが開きます。

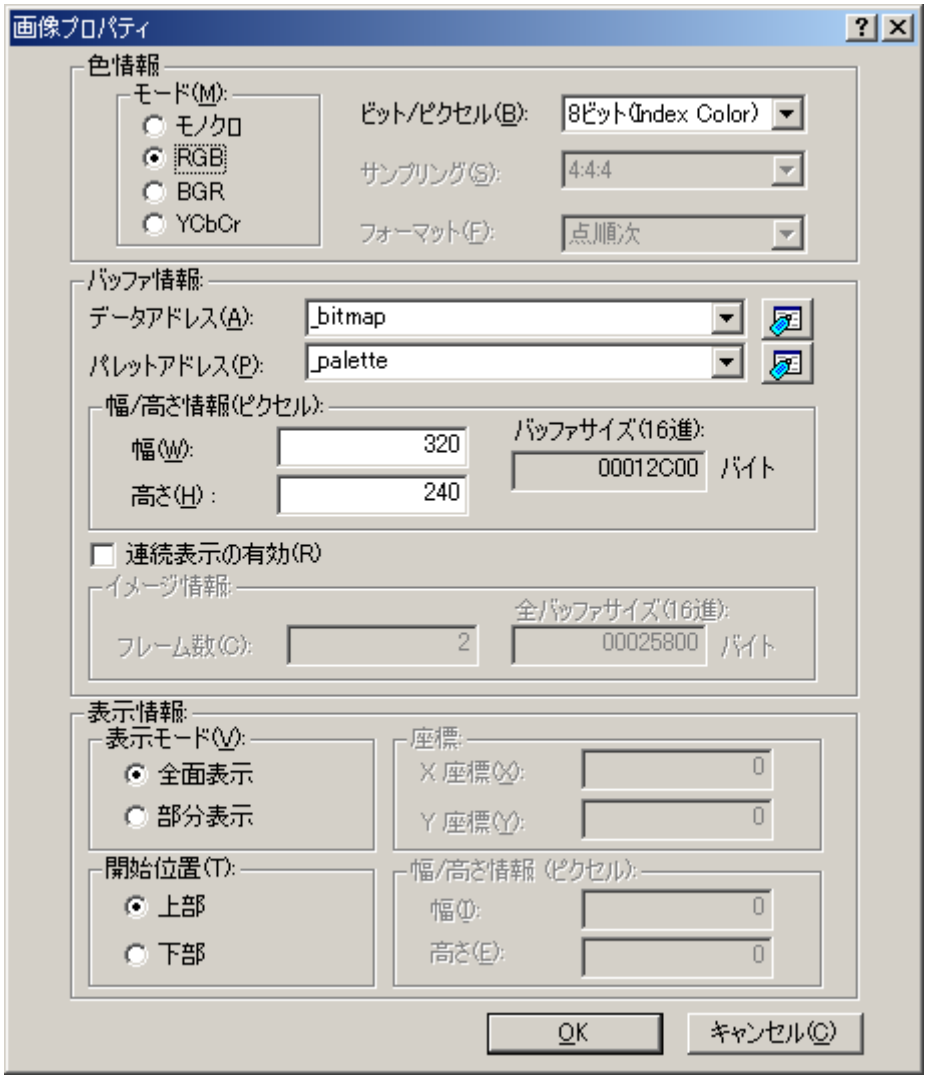

図 14　「画像プロパティ」ダイアログ

4.3　平滑化フィルタのサンプル

次に平滑化フィルタのマスクのプログラムを示します。smoothing_mask_1 が相加平均、smoothing_mask_2 が重み付き平均の例に対応します。

```
int16_t smoothing_mask_1[12] = {
    3640, 3640, 3640, /* padding */ 0,
    3640, 3640, 3640, /* padding */ 0,
    3640, 3640, 3640, /* padding */ 0,
};

int16_t smoothing_mask_2[12] = {
    2048, 4096, 2048, /* padding */ 0,
    4096, 8192, 4096, /* padding */ 0,
    2048, 4096, 2048, /* padding */ 0,
};
```

これらのマスクの係数は、符号付き 16 ビットの固定小数点データで表現されています。平滑化フィルタでは、平均をとるために係数の合計で割算する必要がありますが、割算は前もってそれぞれの係数に分配して計算してあります。

上の平滑化フィルタマスクを使ったサンプルプログラムを次に示します。平滑化フィルタマスクは固定小数点の係数を格納しているので、固定小数点用のフィルタ処理関数 filter_macw1 を使用します。

```
    /* テスト画像の平滑化(1) */
    memset(buf, 0, sizeof buf);
    filter_macw1(test_image, buf, smoothing_mask_1);
    put_image(buf);

    /* テスト画像の平滑化(2) */
    memset(buf, 0, sizeof buf);
    filter_macw1(test_image, buf, smoothing_mask_2);
    put_image(buf);
```

4.4　エッジ検出フィルタのサンプル

最初に、Sobel フィルタのマスクのプログラムを次に示します。sobel_h_mask が水平方向用のフィルタマスク、sobel_v_mask が 垂直方向のフィルタマスクに対応します。

```
/* Sobel (水平方向) エッジ検出フィルタ */
int16_t sobel_h_mask[12] = {
    -1,  0,  1, /* padding */ 0,
    -2,  0,  2, /* padding */ 0,
    -1,  0,  1, /* padding */ 0,
};

/* Sobel (垂直方向) エッジ検出フィルタ */
int16_t sobel_v_mask[12] = {
    -1, -2, -1, /* padding */ 0,
     0,  0,  0, /* padding */ 0,
     1,  2,  1, /* padding */ 0,
};
```

上のフィルタマスクを使ったサンプルプログラムを次に示します。これらのフィルタマスクは整数係数ですので、整数用のフィルタ処理関数 filter_macl を使用します。

```
    /* テスト画像のエッジ検出 (Sobel、水平方向)  */
    memset(buf, 0, sizeof buf);
    filter_macl(test_image, buf, sobel_h_mask);
    put_image(buf);

    /* テスト画像のエッジ検出 (Sobel、垂直方向)  */
    memset(buf, 0, sizeof buf);
    filter_macl(test_image, buf, sobel_v_mask);
    put_image(buf);
```

次に Prewitt フィルタのマスクのプログラムを示します。prewitt_h_mask が水平方向用のフィルタマスク、prewitt_v_mask が 垂直方向のフィルタマスクに対応します。

```
/* Prewitt (水平方向) エッジ検出フィルタ */
int16_t prewitt_h_mask[12] = {
    -1,  0,  1, /* padding */ 0,
    -1,  0,  1, /* padding */ 0,
    -1,  0,  1, /* padding */ 0,
};

/* Prewitt (垂直方向) エッジ検出フィルタ */
int16_t prewitt_v_mask[12] = {
    -1, -1, -1, /* padding */ 0,
     0,  0,  0, /* padding */ 0,
     1,  1,  1, /* padding */ 0,
};
```

上のフィルタマスクを使ったサンプルプログラムを次に示します。これらのフィルタマスクは整数係数ですので、整数用のフィルタ処理関数 filter_macl を使用します。

```
    /* テスト画像のエッジ検出 (Prewitt、水平方向)  */
    memset(buf, 0, sizeof buf);
    filter_macl(test_image, buf, prewitt_h_mask);
    put_image(buf);

    /* テスト画像のエッジ検出 (Prewitt、垂直方向)  */
    memset(buf, 0, sizeof buf);
    filter_macl(test_image, buf, prewitt_v_mask);
    put_image(buf);
```

4.5　鮮鋭化処理 (ラプラシアン・フィルタ) のサンプル

次にラプラシアン・フィルタのマスクのプログラムを示します。

```
/* ラブラシアン・フィルタ */
int16_t laplacian_mask[12] = {
    -1, -1, -1, /* padding */ 0,
    -1,  8, -1, /* padding */ 0,
    -1, -1, -1, /* padding */ 0,
};
```

上のフィルタマスクを使った鮮鋭化処理のサンプルプログラムを次に示します。ラプラシアン・フィルタのマスクは整数係数ですので、整数用のフィルタ処理関数 filter_macl を使用します。サンプルプログラムでは、フィルタの出力をスケーリングした後、元の画像を加えてからもう一度スケーリング処理をしています。

```
    /* テスト画像の鮮鋭化（ラプラシアン・フィルタ） */
    memset(buf, 0, sizeof buf);
    filter_macl(test_image, buf, laplacian_mask);
    equalize(buf);
    put_image(buf);
    image_add(buf, test_image);
    equalize(buf);
    put_image(buf);
```

ホームページとサポート窓口

ルネサス エレクトロニクスホームページ
http://japan.renesas.com/

お問合せ先
http://japan.renesas.com/inquiry

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2011.03.14	—	初版発行

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 未使用端子の処理
   【注意】未使用端子は、本文の「未使用端子の処理」に従って処理してください。
   CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。
2. 電源投入時の処置
   【注意】電源投入時は，製品の状態は不定です。
   電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。
   外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。
   同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. リザーブアドレス（予約領域）のアクセス禁止
   【注意】リザーブアドレス（予約領域）のアクセスを禁止します。
   アドレス領域には、将来の機能拡張用に割り付けられているリザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
4. クロックについて
   【注意】リセット時は、クロックが安定した後、リセットを解除してください。
   プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。
   リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
5. 製品間の相違について
   【注意】型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。
   同じグループのマイコンでも型名が違うと、内部 ROM、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ輻射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA機器、通信機器、計測機器、AV機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制するRoHS指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注2. 本資料において使用されている「当社製品」とは、注1において定義された当社の開発、製造製品をいいます。

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口 http://www.renesas.com

※営業お問合せ窓口の住所・電話番号は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス販売株式会社　〒100-0004　千代田区大手町2-6-2（日本ビル） (03)5201-5307

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：http://japan.renesas.com/inquiry

Colophon 1.0