RX ファミリ
DSP 機能命令を活用した複素数演算プログラム

R01AN0229JJ0100
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複素数演算の方法を説明します。

動作確認デバイス

RX ファミリ

目次

1. はじめに 2
2. 複素数のデータ表現 3
3. 複素数の四則演算 4
4. サンプルプログラム: 等角写像 7

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複素数の演算プログラムについて説明します。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. 複素数のデータ表現

複素数は、実部 (real part) と虚部 (imaginary part) からなる数です。RX の積和演算命令の演算対象は符号付き 16 ビット整数ですので、本アプリケーションノートでは、複素数を 2 つの 16 ビット符号付き整数をメンバとして持つ構造体で表現することにします。実部と虚部のデータは 16 ビットの符号付き整数ないしは固定小数点データと見なすことができます。なお、構造体のサイズが 32 ビットに収まるので、関数呼び出しではレジスタで受け渡しされることが期待できます。

上述の複素数のデータ表現と後述する演算プログラムのインタフェースを一つのヘッダファイルにまとめて定義します。次にヘッダファイル ("RXComplex.h") の内容を示します。

```
#ifndef  _RXCOMPLEX_H
#define  _RXCOMPLEX_H

#include <stdint.h>

/* complex number */
typedef struct {
   int16_t re;   /* real part */
   int16_t im;   /* imaginary part */
} RXComplex;

/* function(s) */
RXComplex complex_add(RXComplex a, RXComplex b);
RXComplex complex_sub(RXComplex a, RXComplex b);
RXComplex complex_mul(RXComplex a, RXComplex b);
RXComplex complex_div(RXComplex a, RXComplex b);

#endif /* ! _RXCOMPLEX_H */
```

3. 複素数の四則演算

本章では、複素数の四則演算について説明します。

3.1 加減算

本節では複素数の加減算について説明します。最初に、複素数の加算の定義を次に示します。複素数の加算では実部と虚部のそれぞれの値を加えます。

$$(a+bi)+(c+di)=(a+c)+(b+d)i$$

複素数の加算関数 complex_add のプログラムを次に示します。この関数は複素数 a と複素数 b の和を返します。加算結果の小数点位置は変化しません (入力を固定小数点データと見なした場合)。

```
/*
   複素数の加算
   複素数 a と複素数 b の和を返す
 */
RXComplex complex_add(RXComplex a, RXComplex b)
{
    RXComplex res;

    res.re = (int16_t)(a.re + b.re);
    res.im = (int16_t)(a.im + b.im);
    return res;
}
```

次に、減算について説明します。複素数の減算の定義は次のとおりです。複素数の減算では実部と虚部のそれぞれの値を引き算します。

$$(a+bi)-(c+di)=(a-c)+(b-d)i$$

複素数の減算関数 complex_sub のプログラムを次に示します。この関数は複素数 a と複素数 b の差を返します。減算結果の小数点位置は変化しません (入力を固定小数点データと見なした場合)。

```
/*
   複素数の減算
   複素数 a と複素数 b の差を返す
 */
RXComplex complex_sub(RXComplex a, RXComplex b)
{
    RXComplex res;

    res.re = (int16_t)(a.re - b.re);
    res.im = (int16_t)(a.im - b.im);
    return res;
}
```

3.2　乗算

本節では乗算について説明します。複素数の乗算の定義は次のとおりです。

$$(a+bi)\times(c+di)=(ac-bd)+(bc+ad)i$$

積和演算の組み込み関数 macl を使って作成した乗算関数 complex_mul のプログラムを次に示します。この関数は、複素数 a と複素数 b の積を返します。

```
#include <machine.h>

/*
   複素数の乗算
   複素数 a と複素数 b の積を返す
   注意：結果の値の小数点位置は 2 倍 (a と b の両方) のビット数だけ右に移動する
 */
RXComplex complex_mul(RXComplex a, RXComplex b)
{
    RXComplex res;
    int16_t t;

    t = b.im;
    b.im = (int16_t)(-t);
    /* res.re <-- (a.re * b.re) - (a.im * b.im) */
    res.re = (int16_t)macl(&a.re, &b.re, 2);
    b.im = b.re;
    b.re = t;
    /* res.im <-- (a.re * b.im) + (a.im * b.re) */
    res.im = (int16_t)macl(&a.re, &b.re, 2);
    return res;
}
```

上のプログラムでは、積和演算を使うために、実部の計算の前に b.im (b の虚部) の符号の反転、および、虚部の計算の前に b.re (b の実部) と b.im (b の虚部) の入れ替えを行っています。

なお、入力が固定小数点データの場合には乗算結果の小数点位置が右に移動することに注意してください。例えば、bit15 と bit14 の間に小数点がある 16 ビットの符号付き固定小数点データを入力した場合は、乗算結果の小数点の位置は bit14 と bit13 の間に移動します。必要ならば、上のプログラムで使用している積和演算の組み込み関数 macl を固定小数点用の macw1 または macw2 に置き換えて使用してください。

3.3　除算

本節では除算について説明します。複素数の除算の定義は次のとおりです。

$$(a+bi)\div(c+di)=\left(\frac{ac+bd}{c^2+d^2}\right)+\left(\frac{bc-ad}{c^2+d^2}\right)i$$

積和演算の組み込み関数 macl を使って作成した除算関数 complex_div のプログラムを次に示します。この関数は、複素数 a を複素数 b で除した結果を返します。ただし、割り算の被除数が除数よりも小さくなる入力データについては正しい結果が得られないことに注意してください。なお、除算結果の小数点位置は変化しません (入力を固定小数点データと見なした場合)。

```
#include <machine.h>

/*
   複素数の除算
   複素数 a を複素数 b で除した結果を返す
   注意：被除数が除数より小さくなる場合は正しい結果にならない
 */
RXComplex complex_div(RXComplex a, RXComplex b)
{
    RXComplex res;
    int16_t d, t;

    /* d <-- (b.re * b.re + b.im * b.im) */
    d = (int16_t)macl(&b.re, &b.re, 2);
    /* res.re <-- (a.re * b.re + a.im * b.im) / d */
    res.re = (int16_t)(macl(&a.re, &b.re, 2) / d);
    /* res.im <-- (a.im * b.re - a.re * b.im) / d */
    t = a.re;
    a.re = a.im;
    a.im = (int16_t)(-t);
    res.im = (int16_t)(macl(&a.re, &b.re, 2) / d);
    return res;
}
```

上のプログラムでは、積和演算を使うために、虚部の計算の前に a.re (a の実部) と a.im (a の虚部) を入れ替え、同時に a.re の符号を反転しています。

4. サンプルプログラム: 等角写像

複素数の演算を使ったサンプルプログラム として等角写像 (conformal map) を取り上げます。等角写像とは、局所的な線分がなす角度が保存されるような変換です。多くの場合、等角写像は複素平面 (z 平面) 上の写像として、複素数関数

$$w = f(z)$$

で表されます。上の関数は、元の座標系を z として、写像後の座標系を w に対応させます (複素数の実部を横軸に、虚部を縦軸に対応させます)。本章では、次の z の二乗の等角写像

$$w = z^2$$

をグラフ化するプログラムについて説明します。

4.1 ビットマップとカラーパレット

本節ではグラフを描画するビットマップについて説明します。

ビットマップは幅と高さをもつ四角形の領域で、ピクセルの配列として表現します。ピクセルの深さ (データサイズ) は 8 ビットとし、256 色のカラーパレットのインデックスを格納します。ビットマップ上のピクセルは、四角形の右上隅を原点として右から左に向かって水平方向に広がる x 軸と、上から下に向かって垂直方向に広がる y 軸からなる座標系で指定されます。水平方向のピクセルの並びをスキャンラインと呼び、スキャンライン上のピクセルは左から右に昇順に並んでいるものとします。また、ビットマップの中でスキャンラインは上から下に昇順で並んでいるものとします。

ビットマップから参照されるカラーパレットは、サイズが 256 の符号無し 32 ビット整数の配列で表現します。カラーパレットで指定するピクセルの色は RGB888 形式で指定します。ちなみに、今回は白と赤と緑の三色を定義して使います。

以上を次のプログラムに示します。

```
/* constant(s) */
#define  WIDTH   202
#define  HEIGHT  202
#define  WHITE   0
#define  RED     1
#define  GREEN   2

/* bitmap and palette */
uint8_t bitmap[WIDTH * HEIGHT];
uint32_t palette[256] = {
    0xffffff,   /* [0] --> white */
    0xff0000,   /* [1] --> red */
    0x00ff00,   /* [2] --> green */
};
```

4.2　描画サブルーチン

本節ではグラフの描画に使用する描画サブルーチンを説明します。まず、次に示す関数 clear は、ビットマップ全体を白色で塗り潰して初期化します。

```
/* bitmap を初期化 (白で塗り潰し)  */
void clear(void)
{
    memset(bitmap, WHITE, sizeof bitmap);
}
```

次に示す関数 set_pixel は、ビットマップの上の座標 (x, y) に色 c の点を描画します。set_pixel は直線を描画する関数 draw_line から呼び出されます。

```
/* bitmap 上の(x, y)のピクセルの値を c に設定する */
void set_pixel(int x, int y, int c)
{
    if (x >= 0 && x < WIDTH && y >= 0 && y < HEIGHT) {
        bitmap[x + y * WIDTH] = c;
     }
}
```

次に示す関数 draw_line は、ビットマップの上の二つの点 (x1, y1) と (x2, y2) の間に色 c で直線を描画します。

```
/* bitmap 上の(x1, y1)-(x2, y2)にピクセル値 c の直線を引く */
void draw_line(int x1, int y1, int x2, int y2, int c)
{
    int inc = 1;
    float d;
    float x = x1;
    float y = y1;

    if ((x1 < 0 && x2 < 0) || (y1 < 0 && y2 < 0) ||
        (x1 > WIDTH && x1 > WIDTH) ||
        (y1 > HEIGHT && x2 > HEIGHT)) {
        return; /* 領域外：なにもしない */
    }
    if (x1 == x2 && y1 == y2) {
        /* 点 */
        set_pixel(x1, y1, c);
        return;
    }
    if (abs(x1 - x2) > abs(y1 - y2)) {
        /* 傾きが 45 度より小 (＝水平方向に長い)  */
        d = (float)(y2 - y1) / (float)(x2 - x1);
        if (x1 > x2) {
            inc = -1;
            d = -d;
        }
        while (x1 != x2) {
            set_pixel(x1, (int)y, c);
            x1 += inc;
            y += d;
        }
    }
```

```
    else {
        /* 傾きが 45 度より大 (＝垂直方向に長い)  */
        d = (float)(x2 - x1) / (float)(y2 - y1);
        if (y1 > y2) {
            inc = -1;
            d = -d;
        }
        while (y1 != y2) {
            set_pixel((int)x, y1, c);
            y1 += inc;
            x += d;
        }
    }
}
```

4.3　z 平面のグラフ

次に示す関数 plot_plain は、等角写像の元になる複素平面 (z 平面) をグリッドとしてグラフ表示します。

```
/* 変換前の平面グリッドをグラフ表示 */
void plot_plain(void)
{
    int i, j, x, y, x0, y0;

    clear();
    for (j = -100; j <= 100; j += 10) {
        for (i = -100; i <= 100; i += 10) {
            x = i + WIDTH / 2;
            y = j + HEIGHT / 2;
            if (i != -100) {
                draw_line(x0, y0, x, y, RED);
            }
            x0 = x;
            y0 = y;
        }
    }
    for (i = -100; i <= 100; i += 10) {
        for (j = -100; j <= 100; j += 10) {
            x = i + WIDTH / 2;
            y = j + HEIGHT / 2;
            if (j != -100) {
                draw_line(x0, y0, x, y, GREEN);
            }
            x0 = x;
            y0 = y;
        }
    }
}
```

この関数を呼び出すと図 1 に示すグラフをビットマップに描画します。グラフでは、z 平面の横軸 (実部) を赤色で、縦軸 (虚部) を緑色で描画しています。

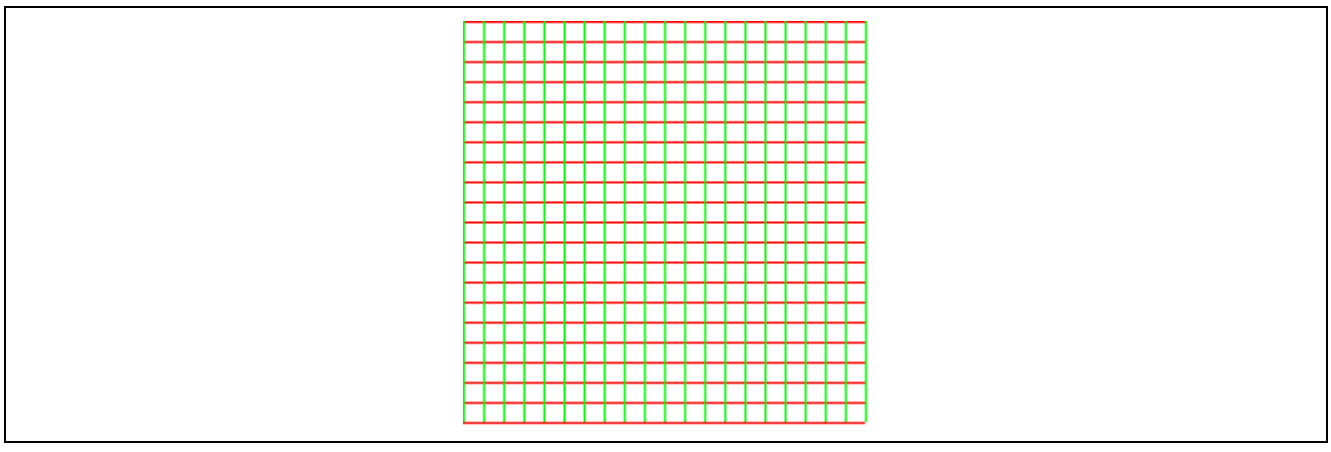

図 1　z 平面のグラフ (赤が横軸、緑が縦軸)

4.4　z の二乗の等角写像のグラフ

次に示す関数 plot_square は、複素平面 (z 平面) が z の二乗の等角写像

$$w = z^2$$

によってどのように写像されるかをグリッド表示でグラフ化します。

```
/* 変換式(z^2)で変換した平面グリッドをグラフ表示 */
void plot_square(void)
{
    int i, j, x, y, x0, y0;
    RXComplex z;
    const float scale_x = 0.005;
    const float scale_y = 0.005;
    const int translate_x = 100;
    const int translate_y = 100;

    clear();
    for (j = -100; j <= 100; j += 10) {
        for (i = -100; i <= 100; i++) {
            z.re = i;
            z.im = j;
            z = complex_mul(z, z);
            x = translate_x + z.re * scale_x;
            y = translate_y + z.im * scale_y;
            if (i != -100) {
                draw_line(x0, y0, x, y, RED);
            }
            x0 = x;
            y0 = y;
        }
    }
    for (i = -100; i <= 100; i += 10) {
        for (j = -100; j <= 100; j += 1) {
            z.re = i;
            z.im = j;
            z = complex_mul(z, z);
            x = translate_x + z.re * scale_x;
            y = translate_y + z.im * scale_y;
            if (j != -100) {
                draw_line(x0, y0, x, y, GREEN);
            }
            x0 = x;
            y0 = y;
        }
    }
}
```

この関数を呼び出すと図 2 に示すグラフをビットマップに描画します。グラフでは、z 平面の横軸 (実部) の変換先を赤色で、縦軸 (虚部) の変換先を緑色で描画します。

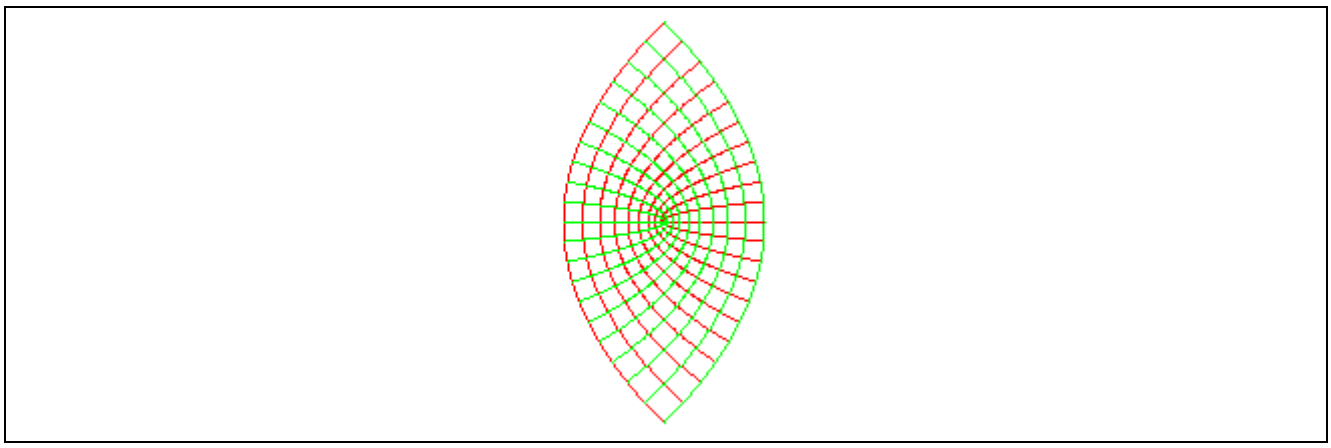

図 2　z の二乗の等角写像のグラフ (赤が横軸、緑が縦軸)

4.5　メインプログラム

メインプログラム (main 関数) 部分を次に示します。メインプログラムは、グラフを表示する関数を順番に呼び出しているだけです。RX の統合開発環境 (High-performance Embedded Workshop) でサンプルプログラムを実行する場合は、関数 pause にブレークポイントを設定しておくと、グラフを表示した直後にプログラムを一時停止させることができます。

```
/* この関数にブレークを設定してグラフ表示を確認する */
void pause(void)
{
}

void main(void)
{
    plot_plain();
    pause();
    plot_square();
    pause();
}
```

4.6　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を RX の 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 3 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 202 (ビットマップの幅) を設定します。
8. ダイアログの「高さ」202 (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。

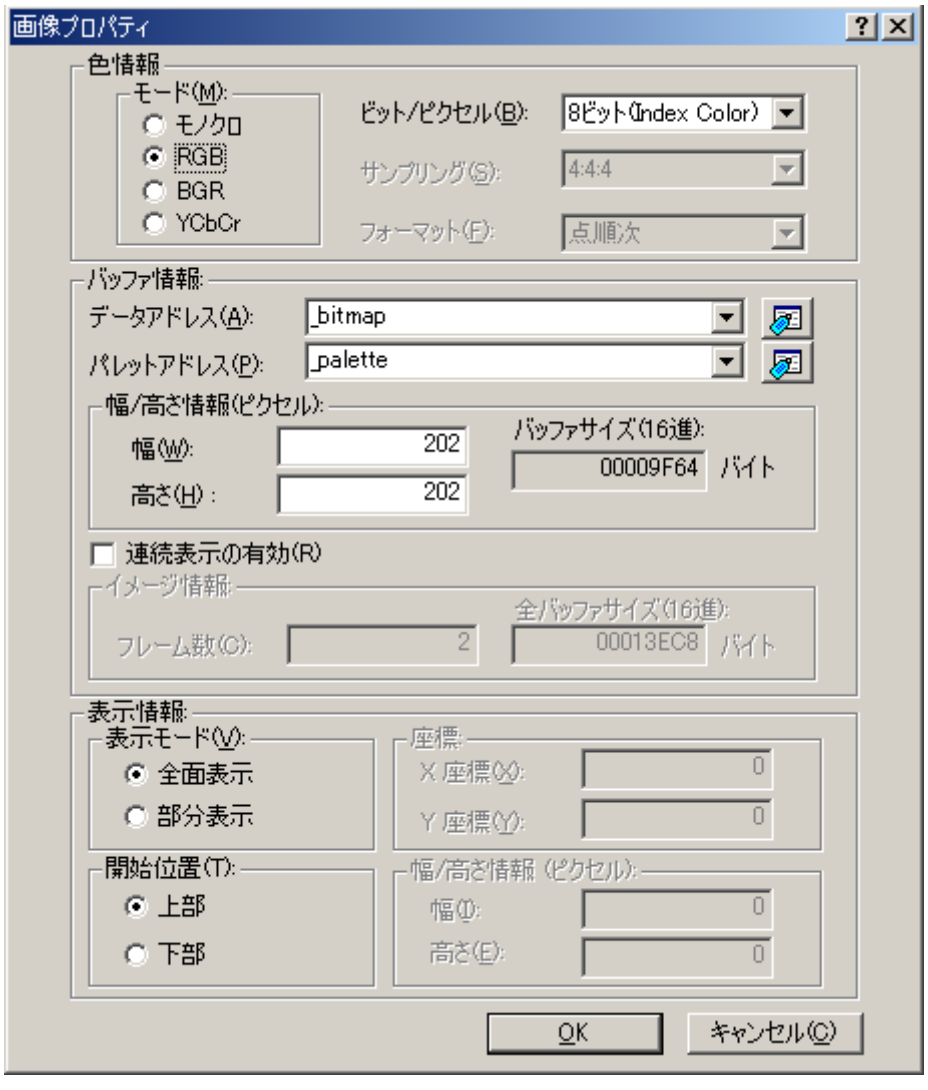

図 3　「画像プロパティ」ダイアログ

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