RX ファミリ
積和演算の組み込み関数の活用方法

R01AN0255JJ0100
Rev.1.00
2011.03.14

要旨

この文書は、RX ファミリの DSP 機能命令の組み込み関数の使用方法を説明します。

動作確認デバイス

RX ファミリ

目次

1. はじめに

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 ビットを超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 ソフトウェアマニュアル」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」（R01AN0254JJ0100）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。

本アプリケーションノートでは、RX ファミリ C/C++コンパイラ（以下コンパイラと略）の拡張機能である積和演算の組み込み関数（コンパイラ V1.01 からサポートされます）の使い方について説明します。組み込み関数は C 言語の関数呼び出しの形式で記述することができ、コンパイラによってインライン展開されます。組み込み関数を使うことで、アセンブリ言語を使わずに効率のよい積和演算をアプリケーションから容易に利用できます。

[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. 組み込み関数の種類

コンパイラの拡張機能として次の 3 種類の積和演算の組み込み関数が用意されています。

- macw1
- macw2
- macl

macw1 と macw2 は固定小数点データ向きの組み込み関数です。macw1 と macw2 は、16 ビット符号付きデータ同士の積和演算の結果を RACW 命令で丸め処理をしてから 16 ビット符号付きデータとして返します。macw1 と macw2 の違いは、丸め処理を「RACW #1」命令で行うか、それとも「RACW #2」命令で行うかという点です。これに対して、macl は整数データ向きの組み込み関数になっています。macl は、16 ビット符号付きデータ同士の積和演算の結果を 32 ビット符号付きデータとして返します。

2.1　macw1

組み込み関数 macw1 は固定小数点データ向きの積和演算を行います。macw1 は 16 ビットの符号付きデータ同士の積和演算を実行し、結果を 16 ビットの符号付きデータで返します。ただし、演算結果は「RACW #1」命令で丸め処理を行います。

組み込み関数 macw1 の呼び出し形式は次のとおりです。

```
#include <machine.h>
short macw1(short* data1, short* data2, unsigned long count);
```

引数 data1 と data2 にはそれぞれ 16 ビット符号付きデータ配列の先頭アドレスを指定します。引数 count には配列の長さ（要素数）を指定します。

組み込み関数 macw1 の呼び出しは次の演算の流れにインライン展開されます。

1. アキュムレータの初期化（0 を設定）
2. data1 と data2 の積和演算の繰り返し処理ループ（MULLO、MULHI、MACLO、MACHI 命令）
3. 演算結果（アキュムレータ）の丸め処理（「RACW #1」命令）
4. アキュムレータからの演算結果の取り出し（MVFACHI 命令）

この演算の流れを図 1 に示します。積和演算はアキュムレータ（ACC）を使って 48 ビット長の精度で実行します。演算結果は「RACW #1」命令で丸め処理をした後、MVFACHI 命令でアキュムレータの上位 32 ビットを取り出して戻り値にします。

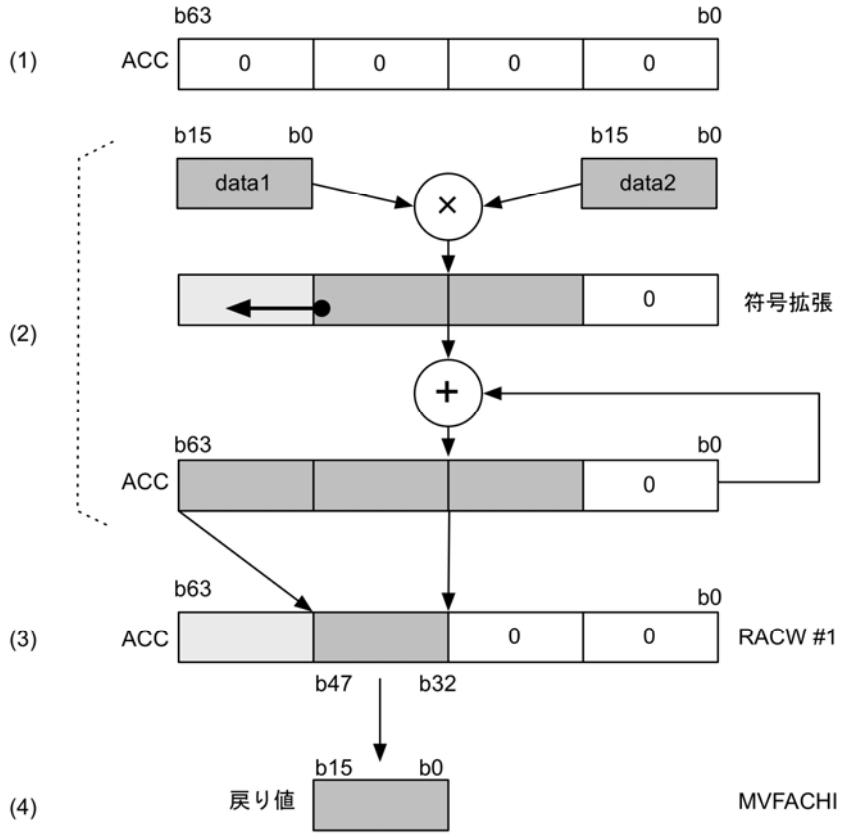

図 1　組み込み関数 macw1 の演算の流れ

また、この演算の流れを C 言語の擬似コードで表現すると次のようになります。

```
short macw1(short* data1, short* data2, unsigned long count)
{
    short ret;

    /* (1) */
    アキュムレータの初期化;
    /* (2) */
    while (count--) {
        積和演算(*data1, *data2);  // MULLO、MACLO、MACHI 命令
        data1++;
        data2++;
    }
    /* (3) */
    丸め処理;  // 「RACW #1」命令
    /* (4) */
    ret = アキュムレータからの演算結果の取り出し;  // MVFACHI 命令
    return ret;
}
```

2.2　macw2

組み込み関数 macw2 は、固定小数点データ向きの積和演算を行います。macw2 は 16 ビットの符号付き整数データ同士の積和演算を実行し、結果を 16 ビットの符号付き整数データで返します。ただし、演算結果は「RACW #2」命令で丸め処理をします。

組み込み関数 macw2 の呼び出し形式は次のとおりです。

```
#include <machine.h>
short macw2(short* data1, short* data2, unsigned long count);
```

引数 data1 と data2 にはそれぞれ 16 ビット符号付きデータ配列の先頭アドレスを指定します。引数 count には配列の長さ（要素数）を指定します。

組み込み関数 macw2 の呼び出しは次の演算の流れにインライン展開されます。

(1) アキュムレータの初期化（0 を設定）

(2) data1 と data2 の積和演算の繰り返し処理ループ（MULLO、MULHI、MACLO、MACHI 命令）

(3) 演算結果（アキュムレータ）の丸め処理（「RACW #2」命令）

(4) アキュムレータからの演算結果の取り出し（MVFACHI 命令）

この演算の流れを図 2 に示します。積和演算はアキュムレータ（ACC）を使って 48 ビット長の精度で実行します。演算結果は「RACW #2」命令で丸め処理をした後、MVFACHI 命令でアキュムレータの上位 32 ビットを取り出して戻り値にします。

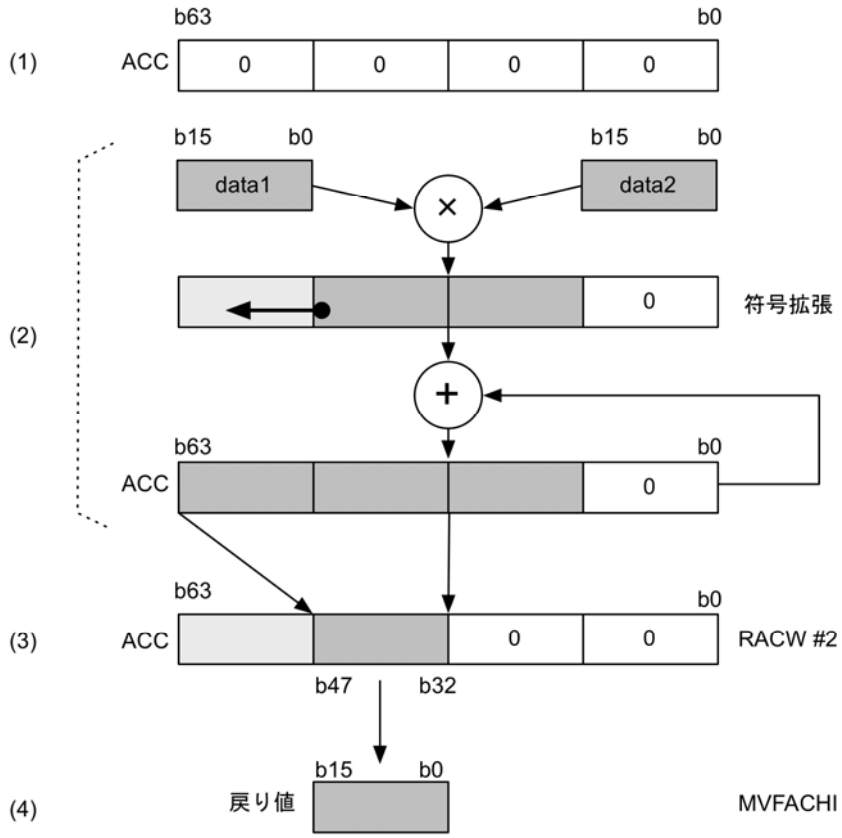

図 2　組み込み関数 macw2 の演算の流れ

また、この演算の流れを C 言語の擬似コードで表現すると次のようになります。

```
short macw2(short* data1, short* data2, unsigned long count)
{
    short ret;

    /* (1) */
    アキュムレータの初期化;
    /* (2) */
    while (count--) {
        積和演算(*data1, *data2);  // MULLO、MACLO、MACHI 命令
        data1++;
        data2++;
    }
    /* (3) */
    丸め処理;  //「RACW #2」命令
    /* (4) */
    ret = アキュムレータからの演算結果の取り出し;  // MVFACHI 命令
    return ret;
}
```

2.3　macl

組み込み関数 macl は整数データ向きの積和演算を行います。macl は 16 ビットの符号付きデータ同士の積和演算を実行し、結果を 32 ビットの符号付きデータで返します。

組み込み関数 macl の呼び出し形式は次のとおりです。

```
#include <machine.h>
long mac1(short* data1, short* data2, unsigned long count);
```

引数の data1 と data2 にはそれぞれ 16 ビット符号付きデータ配列の先頭アドレスを指定します。引数 count には配列の長さ（要素数）を指定します。

組み込み関数 macl の呼び出しは次の演算の流れにインライン展開されます。

1. アキュムレータの初期化（0 を設定）
2. data1 と data2 の積和演算の繰り返し処理ループ（MULLO、MULHI、MACLO、MACHI 命令）
3. アキュムレータからの演算結果の取り出し（MVFACHI 命令）

この演算の流れを図 3 に示します。積和演算はアキュムレータ（ACC）を使って 48 ビット長の精度で実行します。演算結果は MVFACMI 命令でアキュムレータ中央の 32 ビットを取り出して戻り値にします。

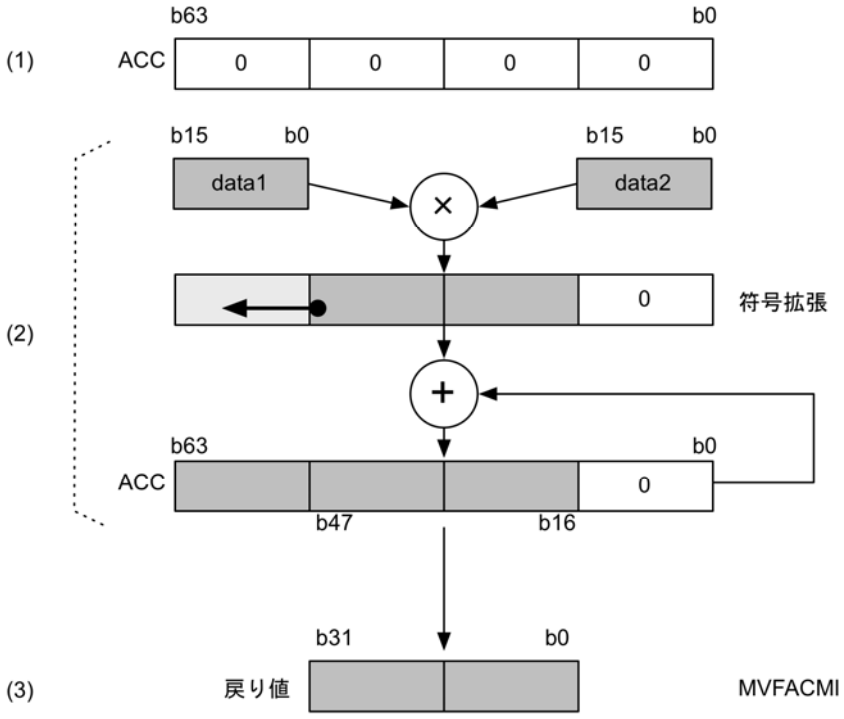

図 3　組み込み関数 macl の演算の流れ

また、この演算の流れを C 言語の擬似コードで表現すると次のようになります。

```
long macl(short* data1, short* data2, unsigned long count)
{
    long ret;

    /* (1) */
    アキュムレータの初期化;
    /* (2) */
    while (count--) {
        積和演算(*data1, *data2);  // MULLO、MACLO、MACHI 命令
        data1++;
        data2++;
    }
    /* (3) */
    ret = アキュムレータからの演算結果の取り出し;  // MVFACMI 命令
    return ret;
}
```

3.　組み込み関数の使用上の注意点

この章では組み込み関数（macw1, macw2, macl）を使用する上での注意点について説明します。

3.1　　データのアドレス境界整合

この節では、C 言語プログラムで積和演算のデータを 4 バイトのアドレス境界に配置する方法について説明します。

組み込み関数（macw1, macw2, macl）は、積和演算を行うデータ数が 2 個以上ある場合には、2 個の 16 ビットデータを 32 ビット（4 バイト長）データにパックしてレジスタにロードする処理に展開されます。RX は任意のアドレス境界から 32 ビットデータを読み出すことができるので、データをメモリ上の任意のアドレスに配置することが可能です。しかし、4 バイトのアドレス境界に整合するようにデータを配置すれば、RX は 1 回のメモリアクセスで 32 ビットデータをロードできるので実行速度の面で有利です。

C 言語で任意のデータを 4 バイト境界に整合させるためには、そのデータをダミーの long 整数メンバと同じ一つの共用体（union）にラッピングするテクニックを使用します。例えば次のように記述します。

```
/* データ u1.data を 4 バイトのアドレス境界に整合させる例 */
union { int16_t data[10]; int32_t dummy; } u1;
```

この例では short 型の配列データ（u1.data[10]）の先頭アドレスが 4 バイト境界に整合されることがコンパイラによって保証されます。

なお、ANSI 標準 C 言語では共用体の先頭メンバに対する初期化が可能ですので、次のように初期化データも問題なく定義できます。

```
/* データ u2.data を 4 バイト境界に整合させ初期値を設定する例 */
const union { int16_t data[10]; int32_t dummy; } u2 = {
    { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }
};
```

上の例では、初期値列が二重の括弧（カーリーブラケット）に囲まれていますが、これは両方とも必要です。外側の括弧は共用体に対応し、内側の括弧は short 型の配列に対応しています。

3.2　積和演算の繰り返しカウント

組み込み関数（macw1, macw2, macl）の第 3 引数には積和演算の繰り返しカウント数を指定します。カウント数には、変数を含む式と定数式のどちらでも記述することができます。ところが、引数のカウント数が変数を含むか定数式であるのかによって、コンパイラは組み込み関数の呼び出しを異なるパターンの命令列に展開します。

他の条件が同じであれば、一般に、カウント数が定数式である方が変数を含む式である場合よりも短くて速いコードが生成されます。したがって、積和演算の繰り返し回数があらかじめわかっているときはカウント数を定数式で指定することで、より効率のよいコード生成が期待できます。

3.3　割り込み関数でのアキュムレータの退避と復帰

組み込み関数（macw1, macw2, macl）を使用するアプリケーションは、割り込み関数の中でアキュムレータ（ACC）の内容が書き換えられないことを保証しなければなりません。つまり、割り込み関数でアキュムレータの内容が書き換えられる可能性がある場合は、割り込み関数の入口でアキュムレータの内容を退避し、出口で復帰する必要があります。

割り込み関数でアキュムレータを退避／復帰する機能の詳細については「RX ファミリ C/C++コンパイラ、アセンブラ、最適化リンケージエディタ コンパイラパッケージ ユーザーズマニュアル」のコンパイルオプションの「save_acc」の説明（コンパイルオプション「save_acc」はコンパイラ V1.01 からサポートされます）、あるいは、拡張言語仕様の「#pragma interrupt」の説明を参照してください。

4.　サンプルプログラム

この章では組み込み関数を使ったプログラム例を座標の回転を用いて説明します。

4.1　座標の回転（座標変換）

図 4 に示すように、原点を中心として 2 次元の座標 (x, y) を回転する変換を考えます。

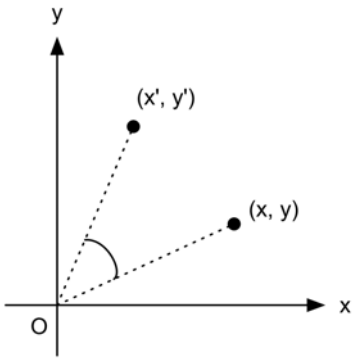

図 4　座標の回転（座標変換）

回転する角度をθ、回転した先の座標を(x',y')とすると、この座標変換は次の行列の乗算で表現できます。

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix}$$

この例では回転する角度θを具体的に 75 度として、次の変換式を使うことにします。

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0.25882 & -0.96593 \\ 0.96593 & 0.25882 \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix}$$

上の式の 4 行 4 列の正方行列（以下変換行列と呼びます）の要素の値は-1.0 から 1.0 の間のデータですので、bit15 と bit14 の間に小数点がある 16 ビット長の固定小数点データで表現できます。このデータ表現では bit14 から bit0 が小数点部分になるので、例えば 0.25882 は

$$0.25882 \times 2^{15} = 8481$$

つまり 16 進表現で 0x2121 となります。このようにして、変換行列の要素を 16 ビットデータとすることで積和演算を使用する準備が整います。

4.2　データ構造と組み込み関数の選択

まず、変換行列の 16 ビット固定小数点データ

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

は一次元の配列で表現し、次の順番（左から右）で各要素を配列に格納します。

$$a,b,c,d$$

次に、二次元の座標(x, y)は 16 ビットの符号付き整数の配列（要素数が 2）で表現し、先頭の要素を x 座標、2 番目の要素を y 座標と約束します。

なお、この例の場合、入力に固定小数点データを含むので、演算結果を 16 ビットの符号付き整数にするために小数点位置を調整する必要があります。具体的には、変換行列の要素の値について bit15 と bit14 の間に小数点があると考えたので、丸め処理をする前に演算結果を左へ 1 ビットだけシフトしなければなりません。そこで積和演算の組み込み関数の macw1 を選択します。

4.3　サンプルプログラム

4.1 節および 4.2 節で説明した考え方で作成したプログラム例を次に示します。

```
#include <machine.h>

/*
  75 度の回転変換
  座標(xy_in[0], xy_in[1])を 75 度回転した先の座標を
  (xy_out[0], xy_out[1])に格納して返す

 */
void rotate75(int16_t *xy_in, short *xy_out)
{
    static union { short matrix[4]; int32_t dummy; } u = {
        {
            0x2121,  0x845d,  /* 0.25882, -0.96593 */
            0x7ba3,  0x2121,  /* 0.96593,  0.25882 */
        }
    };

    xy_out[0] = (int16_t)macw1(xy_in, u.matrix, 2);
    xy_out[1] = (int16_t)macw1(xy_in, u.matrix + 2, 2);
}

void main(void)
{
    union { int16_t coord[2]; int32_t dummy; } u1, u2;

    /* 座標(128, 64)を 75 度回転 */
    u1.coord[0] = 128;
    u1.coord[1] = 64;
    rotate75(u1.coord, u2.coord);  /* 結果は u2.coord に入る */
}
```

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