RX ファミリ
ソフトウェアによるウェイト処理のコーディング例

R01AN1852JJ0100
Rev.1.00
2014.02.03

要旨

本アプリケーションノートでは、ソフトウェアによるウェイト処理(以下、ウェイト処理)のコーディング例について説明します。

対象デバイス

・RX600 シリーズ	RX610 グループ、RX62N/621 グループ、RX62T グループ RX62G グループ、RX630 グループ、RX63N/631 グループ RX63T グループ
・RX200 シリーズ	RX210 グループ、RX220 グループ、RX21A グループ
・RX100 シリーズ	RX110 グループ、RX111 グループ

本アプリケーションノートを他のマイコンへ適用する場合、そのマイコンの仕様にあわせて変更し、十分評価してください。

目次

1.　ウェイト処理について

Do-while 文などの繰り返し処理を使用してウェイト処理を行う場合、以下の理由によりウェイト時間が意図した時間と一致しない場合があります。

- C 言語では、コンパイラの最適化オプションやバージョンの違いにより、出力される命令の種類や数が変化し、実行サイクル数が変化してしまう。
- アセンブリ言語では、アライメントをまたいで命令が配置されると、命令フェッチの回数が増え、実行サイクル数が変化してしまう。

これらの理由により、実行サイクル数が変化しないウェイト処理を記述するには、アセンブリ言語を用い、かつ、命令がアライメントに依存しないようにする必要があります。

以下に実行サイクル数が変化しないウェイト処理のコーディング例を示します。

2.　動作確認条件

本アプリケーションノートのサンプルコードは、下記の条件で動作を確認しています。

表2.1　動作確認条件 (High-performance Embedded Workshop)

項目	内容
使用マイコン	R5F563NBDDFC（RX63N グループ）
統合開発環境	ルネサスエレクトロニクス製 High-performance Embedded Workshop Version 4.09.01
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler Package for RX Family V.1.02 Release 01
	オプション [コンパイラ] -cpu=rx600 -output=obj="$(CONFIGDIR)¥$(FILELEAF).obj" -debug -nologo [アセンブラ] -cpu=rx600 -output="$(CONFIGDIR)¥$(FILELEAF).obj" –debug -nologo [リンカ] -noprelink -rom=D=R,D_1=R_1,D_2=R_2 –nomessage -list="$(CONFIGDIR)¥$(PROJECTNAME).map" -nooptimize -start=B_1,R_1,B_2,R_2,B,R,SU,SI/04,PResetPRG/0FFFF8000,C_1,C_2,C,C$*,D_1,D_2,D,P,PIntPRG,W*,L/0FFFF8100,FIXEDVECT/0FFFFFFD0 -nologo -output="$(CONFIGDIR)¥$(PROJECTNAME).abs" –end -input="$(CONFIGDIR)¥$(PROJECTNAME).abs" -form=stype -output="$(CONFIGDIR)¥$(PROJECTNAME).mot" -exit
エンディアン	リトルエンディアン
サンプルコードのバージョン	Version 1.00

表2.2　動作確認条件 (e2 studio)

項目	内容
使用マイコン	R5F563NBDDFC（RX63N グループ）
統合開発環境	ルネサスエレクトロニクス製 e2 studio Version 2.2.0.13
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler Package for RX Family V.2.01
	オプション [コンパイラ] -cpu=rx600 -include="${TCINSTALL}¥include" –debug –nologo -change_message=warning -define=__RX [アセンブラ] -cpu=rx600 -nolistfile -debug -nologo [リンカ] -library="${CONFIGDIR}¥${ProjName}.lib" -noprelink -list="${ProjName}.map" -show -nooptimize -nomessage -nologo -output="${CONFIGDIR}¥${ProjName}.abs" -rom=D=R -rom=D_1=R_1 -rom=D_2=R_2
エンディアン	リトルエンディアン
サンプルコードのバージョン	Version 1.00

3.　ソフトウェア説明

本サンプルコードでは、2 種類のウェイト処理を用意しています。

・ループ回数を指定するインライン関数

・実行時間を指定する関数

3.1　動作概要

- ループ回数を指定するインライン関数

指定したループ回数のループを行います。

本処理は、アセンブラ埋め込みインライン関数になっており、1 ループを 5 サイクルで実行します。

ループ処理に入る前の分岐命令は、CPU の命令キューをクリアし、1 ループ目の実行サイクル数を 2 ループ目と合わせるためのものです。分岐命令を実行すると命令キューはクリアされ、CPU は分岐先から命令フェッチを開始します。

分岐先の NOP 命令は、アライメントに関係なく固定のサイクルにするためのものです。NOP 命令を実行している間に後続命令の命令フェッチを行うことで、ループ処理の命令実行サイクル数を同じにしています。

- 実行時間を指定する関数

実行時間とシステムクロック(ICLK)の周波数を引数とし、指定した実行時間を待つ処理です。引数で指定した実行時間(μs)とシステムクロックの周波数(kHz)からループ回数を算出して、ループ回数を指定するインライン関数を呼び出します。

関数の呼び出しと復帰、およびループ回数の算出に 20 サイクル程度必要なため、そのオーバヘッドを考慮してループ回数を計算しています。

3.2　ウェイト処理のコーディング例

ループ回数を指定するインライン関数のコーディング例を図 3.1に、実行時間を指定する関数のコーディング例を図 3.2に示します。

■ Cソースコード

```
void main(void)
{
      :
   R_DELAY(LOOP_COUNT);                    ← ループ回数(LOOP_COUNT)を設定。
      :
}

#pragma inline_asm R_DELAY
static void R_DELAY (unsigned long loop_cnt)
{
   BRA    ?+
   NOP
?:
   NOP                                     } アライメントに関係なく固定サイクルとなる。
   SUB    #01H,R1
   BNE    ?-
}
```

図3.1　ループ回数を指定するインライン関数のコーディング例

R_DELAY 関数を呼び出している処理では、引数にループ回数(LOOP_COUNT)を指定しています。

R_DELAY 関数では、R1 に格納されたループ回数を 1 ループごとに減算し、R1 が “0” になったときにループを抜けます。

ループ回数(LOOP_COUNT)を 5 回と仮定すると、R_DELAY 関数の実行サイクルは

5 回ループ × 5 サイクル = 25 サイクル

となります。

■ Cソースコード

```
#include “r_delay.h”                          ← ヘッダファイルのインクルード。

void main(void)
{
      :
   R_DELAY_Us(WAIT_TIME_US, BSP_ICLK_HZ);     ← 実行時間(WAIT_TIME_US)と
      :                                          システムクロック(BSP_ICLK_HZ) を設定。
}
```

■ Cソースコード (r_delay.c)

```
#pragma inline_asm R_DELAY
static void R_DELAY (unsigned long loop_cnt)
{
   BRA    ?+
   NOP
?:
   NOP
   SUB    #01H,R1
   BNE    ?-

}

void R_DELAY_Us (unsigned long us, unsigned long khz)
{

   signed long loop_cnt;

   loop_cnt = us * khz;
   loop_cnt = ( loop_cnt / 5000 );            } ループ回数を算出している。
   loop_cnt = loop_cnt - 4;

   if( loop_cnt > 0 )
   {
      R_DELAY((unsigned long)loop_cnt);
   }
}
```

図3.2　実行時間を指定する関数のコーディング例

R_DELAY_Us 関数を呼び出している処理では、引数に実行時間(WAIT_TIME_US)とシステムクロック(BSP_ICLK_HZ)を指定しています。

R_DELAY_Us 関数では、まずループ回数を算出し、得られた結果を引数にして R_DELAY 関数を実行します。

ループ回数の算出式は以下のとおりです。

ループ回数 = 実行時間(μs) × システムクロック(kHz) / 5000 [1 ループの実行サイクル×1000]
- 4 ループ[オーバヘッド 20 サイクル]

例えば、実行時間を 100μs、システムクロックを 10000kHz(10MHz)と仮定すると、

ループ回数は、

100 × 10000 / 5000 - 4 = 196 回ループ

実行サイクルは、

196 回ループ × 5 サイクル = 980 サイクル

実行時間(μs)は、

10000kHz (100ns) × (980 サイクル + 20 サイクル[オーバヘッド]) = 100μs

となります。

3.3　使用上の注意事項

以下に、各関数の使用上の注意事項を示します。

- ループ回数を指定するインライン関数
  - ループ回数に“0”を指定することは禁止です。
  - 外付けのメモリで実行する場合は、1 ループのサイクル数が 5 サイクルになりません。

- 実行時間を指定する関数
  - 実行時間とシステムクロック(ICLK)に“0”を指定することは禁止です。
  - 実行時間とシステムクロック(ICLK)は整数で指定してください。
  - オーバヘッドの 20 サイクルは、実行時間(μs)とシステムクロックの周波数(kHz)の値により、増加する場合があります。
  - ループ回数の計算結果の小数点以下は、切り捨てられますので、切り捨てを考慮した実行時間を指定してください。

3.4　ファイル構成

表 3.1にサンプルコードで使用するファイルを示します。

表3.1　サンプルコードで使用するファイル

ファイル名	概要	備考
r_delay.c	ソフトウェアによるウェイト処理	
r_delay.h	r_delay.c のヘッダファイル	

3.5　関数一覧

表 3.2に関数を示します。

表3.2　関数

関数名	概要
R_DELAY	ループ回数を指定するインライン関数
R_DELAY_Us	実行時間を指定する関数

3.6　関数仕様

サンプルコードの関数仕様を示します。

R_DELAY	
概　要	ループ回数を指定するインライン関数
ヘッダ	なし
宣　言	static void R_DELAY　(unsigned long loop_cnt)
説　明	5 サイクル固定でループするウェイト処理
引　数	loop_cnt:　　ループ回数
リターン値	なし
備考	本関数はアセンブラ埋め込みインライン関数です。使用する場合は、使用するソースファイル内に関数を記載してください。 ループ処理の先頭に NOP 命令を追加することで 1 ループのサイクル数を調整することができます。

R_DELAY_Us	
概　要	実行時間を指定する関数
ヘッダ	r_delay.h
宣　言	void R_DELAY_Us (unsigned long us, unsigned long khz)
説　明	実行時間(μs)とシステムクロック(ICLK)の周波数をもとにループ回数を計算し、ループ回数を指定するインライン関数を呼び出す
引　数	us:　　実行時間 khz:　　関数呼び出し時のシステムクロック(ICLK)の周波数
リターン値	なし

3.7　フローチャート

3.7.1　ループ回数を指定する関数

図 3.3にループ回数を指定する関数のフローチャートを示します。

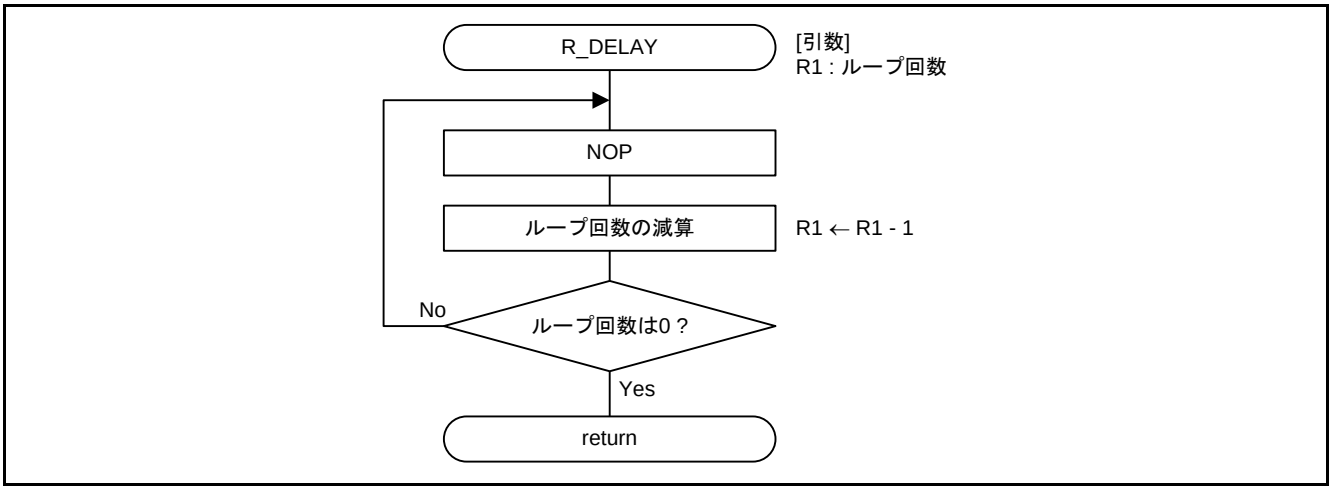

図3.3　ループ回数を指定する関数

3.7.2　実行時間を指定する関数

図 3.4に実行時間を指定する関数のフローチャートを示します。

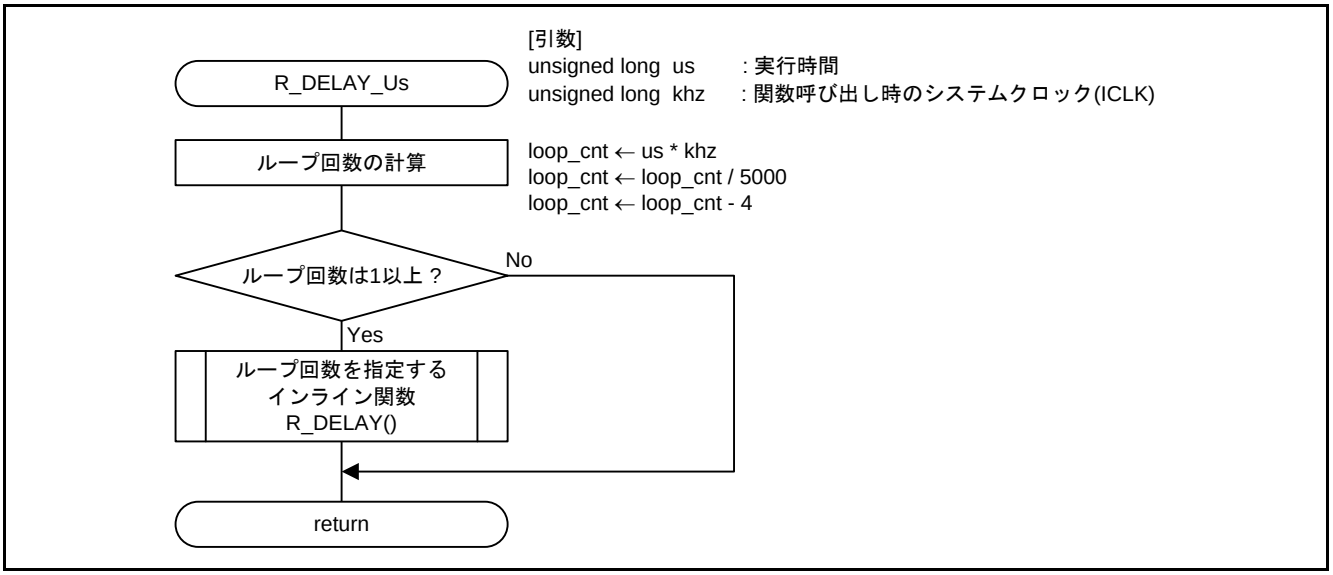

図3.4　実行時間を指定する関数

4.　参考

以下に、実行サイクル数が変化する要因を説明します。

4.1　最適化オプションによる命令コードへの影響

C 言語を使用してウェイト処理を記述する場合、コンパイル時の最適化オプションやコンパイラのバージョンが変わると、出力される命令の種類や数が変わり、ウェイト処理の実行サイクル数も変わってしまいます。

表 4.1に最適化オプションと命令の出力例を示します。

表4.1　最適化オプションと命令の出力例

<table>
<tr><td colspan="3">C 言語ソース</td></tr>
<tr><td>Do-while 文
を用いたウェイト処理</td><td colspan="2">void Software_delay (unsigned long count)
{
　　do
　　{
　　　　count--;
　　} while (count);
}</td></tr>
<tr><td colspan="3">コンパイル結果例</td></tr>
<tr><td>最適化レベル:“0”
1 ループ:6 命令</td><td>最適化レベル:“1”
最適化のタイプ:size 優先
1 ループ:4 命令</td><td>最適化レベル:“2”
最適化のタイプ:size 優先
1 ループ:2 命令</td></tr>
<tr><td>_Software_delay:
　　.STACK
　　_Software_delay=8
　　SUB #04H, R0
　　MOV.L R1, [R0]
L1:
　　MOV.L [R0], R1
　　SUB #01H, R1
　　MOV.L R1, [R0]
L2:
　　MOV.L [R0], R1
　　CMP #00H, R1
　　BNE L1
L3:
　　RTSD #04H</td><td>_Software_delay:
　　.STACK
　　_Software_delay=4
L1:
　　ADD 0FFFFFFFFH, R1, R14
　　CMP #01H, R1
　　MOV.L R14, R1
　　BNE L1
L2:
　　RTS</td><td>_Software_delay:
　　.STACK
　　_Software_delay=4
L1:
　　SUB #01H, R1
　　BNE L1
L2:
　　RTS</td></tr>
</table>

4.2　命令の配置アドレスによる命令実行サイクル数への影響

命令コードのコードサイズが 2 バイト以上で、命令コードの配置アドレスがアライメントをまたぐ場合、その命令フェッチが 2 回行われるため、実行サイクル数が 1 サイクル増加することがあります。

下記のサンプルコードを例に、ウェイト処理において実行サイクル数が増加するメカニズムを説明します。

図 4.1では、サンプルコードをコンパイルすると、右のように命令コードが出力されるものとします。

SUB 命令のアドレスが、図 4.2の「A.実行サイクル数が増加しないパターン」のようにアライメントをまたがないアドレスに配置されている場合、命令フェッチは 1 回で済むため、SUB 命令の実行サイクル数は 1 サイクルになります。

図 4.2の「B.実行サイクル数が増加するパターン」のようにアライメントをまたぐアドレスに配置されている場合、命令フェッチが 2 回実行されるため、SUB 命令の実行サイクル数は 2 サイクルになります。

■ サンプルコード

```
count = XX;
do
 {
        count -- ;
 } while (count);
```

(XX : ループ回数)

■ コンパイル結果 (アセンブラ)

```
アドレス          命令コード
FFFFXXXX                      Label:
FFFFXXXX          6011              SUB #01H, R1
FFFFXXXX+2        21rr              BNE Label
```

図4.1　サンプルコードのコンパイル結果例

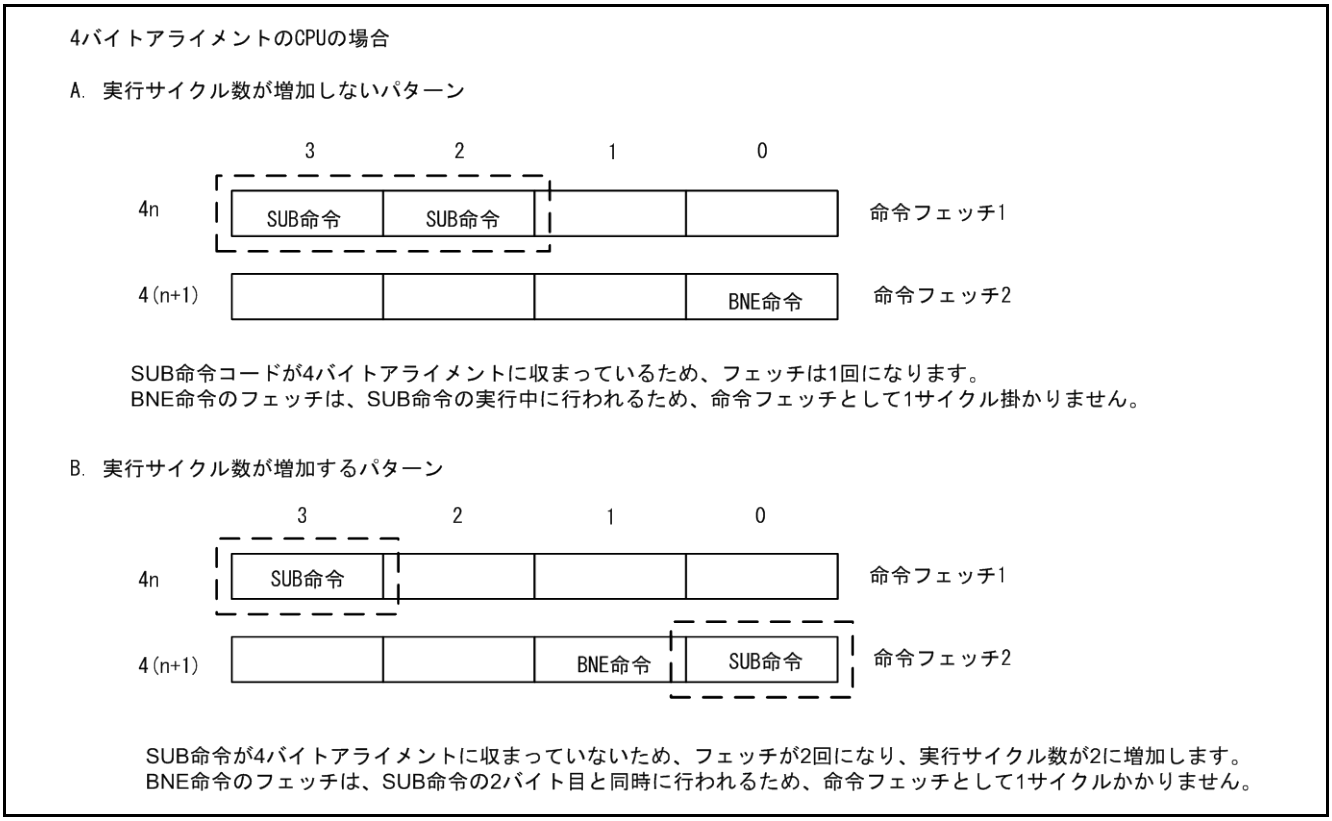

図4.2　命令の配置アドレスと命令実行サイクル数の関係

5.　サンプルコード

サンプルコードは、ルネサス エレクトロニクスホームページから入手してください。

6.　参考ドキュメント

ユーザーズマニュアル：ソフトウェア

RX ファミリ ユーザーズマニュアル ソフトウェア編 Rev.1.20（r01us0032jj0120）
（最新版をルネサス エレクトロニクスホームページから入手してください。）

テクニカルアップデート／テクニカルニュース
（最新の情報をルネサス エレクトロニクスホームページから入手してください。）

ユーザーズマニュアル：開発環境

RX ファミリ C/C++コンパイラパッケージ V.1.01 ユーザーズマニュアル Rev.1.00（r20ut0570jj0100）
（最新版をルネサス エレクトロニクスホームページから入手してください。）

RX ファミリ CC-RX V2.01.00 ユーザーズマニュアル RX コーディング編（r20ut2748jj0100）
（最新版をルネサス エレクトロニクスホームページから入手してください。）

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/contact/

改訂記録	RX ファミリ　アプリケーションノート ソフトウェアによるウェイト処理のコーディング例

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2014.02.03	—	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. リザーブアドレスのアクセス禁止

【注意】リザーブアドレス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、事前に問題ないことをご確認下さい。

同じグループのマイコンでも型名が違うと、内部メモリ、レイアウトパターンの相違などにより、特性が異なる場合があります。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。

ご注意書き

1. 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・システムの設計において、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て、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。
2. 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。
3. 本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害に関し、当社は、何らの責任を負うものではありません。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。
4. 当社製品を改造、改変、複製等しないでください。かかる改造、改変、複製等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、
   各品質水準は、以下に示す用途に製品が使用されることを意図しております。
   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、産業用ロボット等
   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、各種安全装置等
   当社製品は、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（原子力制御システム、軍事機器等）に使用されることを意図しておらず、使用することはできません。たとえ、意図し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損害が生じても、当社は一切その責任を負いません。なお、ご不明点がある場合は、当社営業にお問い合わせください。
6. 当社製品をご使用の際は、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他の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。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、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、社会的損害等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、お客様の機器・システムとしての出荷保証を行ってください。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、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の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。
9. 本資料に記載されている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・システムに使用することはできません。また、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に使用しないでください。当社製品または技術を輸出する場合は、「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。
10. お客様の転売等により、本ご注意書き記載の諸条件に抵触して当社製品が使用され、その使用から損害が生じた場合、当社は何らの責任も負わず、お客様にてご負担して頂きますのでご了承ください。
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禁じます。

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 エレクトロニクス株式会社　〒100-0004　千代田区大手町2-6-2（日本ビル）

■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/contact/