# RX62T グループ

## RX62T グループ MCU の IEC60730 セルフテストコード

R01AN0654JG0100
Rev.1.00
2011.09.27

**【注意事項】**

**本書は英語版のアプリケーションノート**
**文書名：**IEC60730 Self Test Code for RX62T Group CPU
**文書番号：**R01AN0654EG0100
**を、参考用として日本語に翻訳したものです。よって、実際の製品へ適用する場合は、英語版「R01AN0654EG0100」を参照していただきますようお願いします。**

## 要旨

近年、自動電子制御システムはさまざまな用途に拡大しており、信頼性と安全性に対する要求はシステム設計における重要な要素となりつつあります。

たとえば、家庭電化製品向けの IEC60730 安全規格の制定により、メーカーは製品の安全で信頼性の高い動作を保証する自動電子制御を設計する必要があります。

IEC60730 規格は、製品設計のあらゆる面について規定していますが、その中でも Annex H は、マイクロコントローラをベースとする制御システムの設計に非常に重要で、以下のような自動電子制御の 3 つのソフトウェア分類があります。

1. クラス A：機器の安全性が意図されていない制御機能
   例：ルームサーモスタット、湿度コントローラ、照明コントローラ、タイマ、スイッチ
2. クラス B：被制御機器の安全でない動作を防止するように設計されている制御機能
   例：洗濯設備用のサーマルカットオフおよびドアロック
3. クラス C：特別な危険を防止するように設計されている制御機能
   例：密閉型機器用の自動バーナー制御およびサーマルカットオフ
   洗濯機、食器洗い機、乾燥機、冷蔵庫、冷凍庫、調理器、ストーブなどの用途

本アプリケーションノートでは、IEC60730 クラス B 安全規格への準拠を支援するために柔軟なサンプルソフトウェアルーチンの使い方に関するガイドラインを説明しています。これらのルーチンは VDE Test and Certification Institute GmbH によって認定され、テスト認定書のコピーが本アプリケーションノートのダウンロードパッケージに付属しています（以下の【注】1 を参照）。

これらのルーチンは IEC60730 への準拠を基本として開発されていますが、ルネサス MCU のセルフテストのためにシステムに実装することができます。

提供されるソフトウェアルーチンは、リセット後およびプログラム実行中に使用されます。エンドユーザはこれらのルーチンをシステム設計全体に柔軟に組み込むことができますが、本アプリケーションノートおよび付属するサンプルコードにその実例を示します。

RX62T は、使用可能なソフトウェアモジュールだけでなく、安全設計には理想的な追加のハードウェアオプションも提供します。1 つの重要な機能はポートアウトプットイネーブル（POE）モジュールです。このモジュールを使用することで、CPU の残りの部分の状態にかかわらず MTU（マルチファンクションタイマユニット）の PWM 出力端子および GPT（汎用 PWM タイマ）の大電流出力端子を強制的にハイインピーダンス状態にすることができます。これは外部負荷の安全性を確保するために利用可能な優れた機能です。

VDE はその動作をスタンドアロンで評価することはできませんが、IEC60730 ソフトウェア仕様の一部ではないので、RX62T 内部ウォッチドッグ（ウォッチドッグタイマ：WDT、および独立ウォッチドッグタイマ：IWDT）、クロック発生回路および I/O ポートのハードウェア機能を評価しています（詳細については VDE 証明書を参照してください）。

**【注】** 1. **本書は欧州規格** EN60335-1:2002/A1:2004 Annex R **に基づいています。その中では規格** IEC 60730-1 (EN60730-1:2000) **がいくつかの箇所で使用されています。上記の規格の** Annex R **には、定義、情報および該当するパラグラフについて** IEC 60730-1 **にジャンプする** 1 **枚のシートが含まれています。**

## 動作確認デバイス

RX62T グループ

## 目次

# 1. テスト

## 1.1 CPU

本章では、CPU テストルーチンについて説明します。IEC 60730: 1999+A1:2003 Annex H - 表 H.11.12.1 CPU を参照してください。

以下の CPU レジスタがテストされます。R0～R15、ISP、USP、INTB、PC、PSW、BPC、BPSW、FINTV、FPSW および ACC。

ソースファイル'CPU_Test.c'は、レジスタに実際にアクセスするためにインラインアセンブリとともに"C"言語を使用して CPU テストを実装します。カップリングテストバージョンの汎用レジスタを使用する場合は、ファイル CPU_Test_Coupling.c も必要です。ソースファイル'CPU_Test.h'は CPU テストのインタフェースを提供します。ファイル MisraTypes.h'には、MISRA に準拠する標準データ型の定義が含まれます。

これらは CPU 動作の基本をテストします。API 関数にはテストの結果を示すための戻り値はありません。その代わり、これらのテストのユーザが以下の宣言を使用してエラー処理関数を提供してください。

```
extern void CPU_Test_ErrorHandler(void);
```

エラーが検出された場合は、CPU テストによってこの関数にジャンプします。この関数は元のルーチンに戻りません。

CPU テストは複数の関数に分割できますが、時間が許す場合は、1 つの関数呼び出しを使用してすべてのテストを順に実行することもできます。詳細については、「1.1.1 ソフトウェア API」を参照してください。

テスト関数はすべて、ルネサスツールチェインマニュアルに指定されるように C 関数呼び出しの後にレジスタ保存の規則に従います。このため、ユーザはこれらの関数を通常の C 関数と同じように呼び出すことができ、レジスタ値をあらかじめ保存する必要はありません。

**【重要事項】　テストコードの変更を防止するために、**'CPU_Test.c'**ファイルの**"Optimisation"**オプションを**"OFF"**にしておいてください。**

### 1.1.1　ソフトウェア API

表 1　ソースファイル

| ファイル名 |
|---|
| CPU_Test.h |
| CPU_Test.c, CPU_Test_Coupling.c |

| 構文 |
|---|
| `void CPU_TestAll(void)` |
| **説明** |
| **これから説明するすべてのテストは、以下の順序で行います。**<br><br>1. **カップリング** GPR **テスト（***1**、下記参照）を使用する場合：**<br>CPU_Test_GPRsCouplingPartA<br>CPU_Test_GPRsCouplingPartB<br><br>**カップリング** GPR **テストを使用しない場合：**<br>CPU_Test_GeneralA<br>CPU_Test_GeneralB<br><br>2. CPU_Test_Control（*2、**下記参照**）<br>3. CPU_Test_Accumulator<br>4. CPU_Test_PC<br><br>**呼び出し元関数は、プロセッサがスーパーバイザモードであることを確認してください。この関数がユーザモードで呼び出された場合、一部のレジスタビットにアクセスすることができないので、テストはフェイルとなります。**<br><br>**呼び出し元関数は、このテスト中に割り込みが発生しないようにしてください。**<br><br>**エラーが検出された場合、外部関数**'CPU_Test_ErrorHandler'**が呼び出されます。**<br>**詳細については、個別のテストを参照してください。**<br><br>**【注】** *1 **コード内の**#define 'USE_TestGPRsCoupling'**は、汎用レジスタをテストするために使用する関数を選択するために使用します。**<br>*2 RX610 **はその他の** RX **デバイスとは少し異なる** PSW **レジスタを備えています。このため、**RX610 **を使用する場合は、プロジェクトに**"RX610"**を定義してください。** |

| 入力パラメータ | |
|---|---|
| なし | 該当せず |
| **出力パラメータ** | |
| なし | 該当せず |
| **戻り値** | |
| なし | 該当せず |

| 構文 | |
|---|---|
| `void CPU_Test_GPRsCouplingPartA(void)` | |
| **説明** | |
| 汎用レジスタ R0～R15 をテストします。レジスタ間のカップリングの不具合が検出されます。<br>これは全体の GPR テストのパート A の部分で、テストを完了するためには、関数 CPU_Test_GPRsCouplingPartB を使用します。<br><br>呼び出し元関数は、このテスト中に割り込みが発生しないようにしてください。<br><br>エラーが検出された場合、外部関数'CPU_Test_ErrorHandler'が呼び出されます。 | |
| **入力パラメータ** | |
| なし | 該当せず |
| **出力パラメータ** | |
| なし | 該当せず |
| **戻り値** | |
| なし | 該当せず |

| 構文 | |
|---|---|
| `void CPU_Test_GPRsCouplingPartB(void)` | |
| **説明** | |
| 汎用レジスタ R0～R15 をテストします。レジスタ間のカップリングの不具合が検出されます。<br>これは全体の GPR テストのパート B の部分で、テストを完了するためには、関数 CPU_Test_GPRsCouplingPartA を使用します。<br><br>呼び出し元関数は、このテスト中に割り込みが発生しないようにしてください。<br><br>エラーが検出された場合、外部関数'CPU_Test_ErrorHandler'が呼び出されます。 | |
| **入力パラメータ** | |
| なし | 該当せず |
| **出力パラメータ** | |
| なし | 該当せず |
| **戻り値** | |
| なし | 該当せず |

| 構文 |
|---|
| void CPU_Test_GeneralA(void) |
| **説明** |
| レジスタ R1、R2、R3、R4、R5、R14 および R15 をテストします。これらは関数が保存する必要がない汎用レジスタです。レジスタはペアでテストします。<br>各ペアのレジスタに対して、以下のように行います.<br>1. 両方のレジスタに h'55555555 を書き込みます。<br>2. 両方のレジスタを読み出し、一致していることを確認します。<br>3. 両方のレジスタに h'AAAAAAAA を書き込みます。<br>4. 両方のレジスタを読み出し、一致していることを確認します。<br><br>呼び出し元関数は、このテスト中に割り込みが発生しないようにしてください。<br><br>エラーが検出された場合、外部関数'CPU_Test_ErrorHandler'が呼び出されます。 |
| **入力パラメータ** |
| なし　　該当せず |
| **出力パラメータ** |
| なし　　該当せず |
| **戻り値** |
| なし　　該当せず |

| 構文 |
|---|
| void CPU_Test_GeneralB(void) |
| **説明** |
| レジスタ R0、R6、R7、R8、R9、R10、R11、R12 および R13 をテストします。これらは関数が保存する必要がある汎用レジスタです。レジスタはペアでテストします。<br>各ペアのレジスタに対して、以下のように行います.<br>1. 両方のレジスタに h'55555555 を書き込みます。<br>2. 両方のレジスタを読み出し、一致していることを確認します。<br>3. 両方のレジスタに h'AAAAAAAA を書き込みます。<br>4. 両方のレジスタを読み出し、一致していることを確認します。<br><br>呼び出し元関数は、このテスト中に割り込みが発生しないようにしてください。<br><br>エラーが検出された場合、外部関数'CPU_Test_ErrorHandler'が呼び出されます。 |
| **入力パラメータ** |
| なし　　該当せず |
| **出力パラメータ** |
| なし　　該当せず |
| **戻り値** |
| なし　　該当せず |

| 構文 |
| --- |
| `void CPU_Test_Control(void)` |
| **説明** |
| **制御レジスタ** ISP、USP、INTB、PSW、BPC、BPSW、FINTV **および** FPSW **をテストします。**<br>**このテストでは、レジスタ** R1～R5 **が動作していることを前提としています。**<br>**一般に各レジスタのテスト手順は以下のとおりです。**<br>**各レジスタに対して**<br>1. h'55555555 **を書き込みます。**<br>2. **値を読み出して、**h'55555555 **であることを確認します。**<br>3. h'AAAAAAAA **を書き込みます。**<br>4. **値を読み出して、**h'AAAAAAAA **であることを確認します。**<br><br>**しかし、レジスタ内の特定のビットに関する制約により、この値どおりのテストができず、その他のテスト値が選択される場合があることに注意が必要です。**<br><br>**呼び出し元関数は、プロセッサがスーパーバイザモードであることを確認してください。この関数がユーザモードで呼び出された場合、一部のレジスタビットにアクセスすることができないので、テストはフェイルとなります。**<br><br>**呼び出し元関数は、このテスト中に割り込みが発生しないようにしてください。**<br><br>RX610 **はその他の** RX **デバイスとは少し異なる** PSW **レジスタを備えています。このため、**RX610 **を使用する場合は、プロジェクトに**"RX610"**を定義してください。**<br><br>**エラーが検出された場合、外部関数** CPU_Test_ErrorHandler **が呼び出されます。** |

| 入力パラメータ | |
| --- | --- |
| なし | 該当せず |
| **出力パラメータ** | |
| なし | 該当せず |
| **戻り値** | |
| なし | 該当せず |

| 構文 | |
|---|---|
| `void CPU_Test_Accumulator(void)` | |
| **説明** | |
| ACC レジスタをテストします。<br><br>【注】 ビット 0～15 は読み出すことができないので、テストされません。<br>レジスタ値はこのテストで保持されます。<br><br>テスト手順は以下のとおりです。<br>1. 上位 32 ビットに h'55555555 を書き込みます。<br>2. 下位 32 ビットに h'55555555 を書き込みます。<br>3. 上位値を読み出し、h'55555555 であることを確認します。<br>4. 中位値（ビット 47～16）を読み出し、h'55555555 であることを確認します。<br>5. 上位 32 ビットに h'AAAAAAAA を書き込みます。<br>6. 下位 32 ビットに h'AAAAAAAA を書き込みます。<br>7. 上位値を読み出し、h'AAAAAAAA であることを確認します。<br>8. 中位値（ビット 47～16）を読み出し、h'AAAAAAAA であることを確認します。<br><br>このテストでは、レジスタ R1～R5 が動作していることを前提としています。<br><br>エラーが検出された場合、外部関数'CPU_Test_ErrorHandler'が呼び出されます。 | |
| **入力パラメータ** | |
| なし | 該当せず |
| **出力パラメータ** | |
| なし | 該当せず |
| **戻り値** | |
| なし | 該当せず |

| 構文 | |
|---|---|
| void CPU_Test_PC(void) | |
| **説明** | |
| この関数はプログラムカウンタ（PC）レジスタのテストを行います。<br><br>これにより、PC が確実に動作していることをチェックします。<br><br>関数を呼び出すことで PC をテストします。<br>これは関数呼び出し時、その関数を実行用に PC レジスタビットが必要となるように、呼び出される関数はその関数独自のセクションが割り付けられ、このテスト関数から離れた場所に配置することができます。<br><br>この関数が実際に実行されていることを確認できるように、提供されたパラメータの反転値を返します。この戻り値の正しいかどうかがチェックされます。<br><br>エラーが検出された場合、外部関数'CPU_Test_ErrorHandler'が呼び出されます。 | |
| **入力パラメータ** | |
| なし | 該当せず |
| **出力パラメータ** | |
| なし | 該当せず |
| **戻り値** | |
| なし | 該当せず |

## 1.2 ROM

本節では、CRC ルーチンを使用した ROM/フラッシュメモリテストについて説明します。IEC 60730: 1999+A1:2003 Annex H - H2.19.4.1 CRC - 単一ワードを参照してください。

CRC はメモリの内容を表すために単一ワードまたはチェックサムを生成する不具合/エラー制御方法です。CRC チェックサムは、メッセージビットストリームのビット繰り上がりをせずに（減算の代わりに XOR を使用）n 次の多項式の係数を表す長さ n+1 の定義済み（ショート）ビットストリームによる 2 進除算の余りです。除算の前に、n 個のゼロがメッセージストリームに付加されます。CRC は、2 進ハードウェアに実装するのが簡単で、数学的な分析も容易なので、よく使用されています。

ROM テストは、ROM の内容に対する CRC 値を生成し、保存することにより、実行することができます。

メモリセルフテスト時に同じ CRC アルゴリズムを使用して別の CRC 値を生成します。この CRC 値と保存された CRC 値とを比較します。この方法は、すべての 1 ビットエラーと高い確率で複数ビットエラーを認識します。

CRC を使用する複雑な点は、その他の CRC ジェネレータによって生成されるその他の CRC 値と比較する CRC 値を生成する必要があることです。基本的な CRC アルゴリズムが同じ場合でも、結果の CRC 値が変わることがある複数の要素があるので、これは難しいです。これには、データがアルゴリズムに提供される順序、使用するルックアップテーブル内の想定されるビット順序および実際の CRC 値のビットの必要な順序の組み合わせが含まれます。ビッグエンディアンおよびリトルエンディアン方式は、ビット順序が重要になるシリアルデータ転送を使用し、動作するように開発されたために、このような複雑さが生じています。この実装により、-CRC オプションを使用したルネサス RX 標準ツールチェインと同じ結果が得られます。このため、ルネサスツールチェインを使用して基準 CRC を自動的に ROM に挿入する場合は、値は計算された値と直接比較されます。

### 1.2.1 CRC16-CCITT アルゴリズム

RX62T には、CRC16-CCITT のサポートを含む CRC モジュールが含まれています。このソフトウェアを使用して CRC モジュールを駆動すると、この 16 ビット CRC16-CCITT が生成されます。

- 多項式 = 0x1021 ($x^{16} + x^{12} + x^5 + 1$)
- 幅 = 16 ビット
- 初期値 = 0xFFFF
- h'FFFF との XOR 演算結果が CRC に出力されます。

### 1.2.2　CRC ソフトウェア API

すべてのソフトウェアは ANSI C で作成されています。

'MisraTypes.h'には、MISRA に準拠する標準データ型の定義が含まれます。

本項以降に示す関数は、CRC 値を計算し、ROM に格納した値が正しいことを検証するために使用します。

**表 2　ソースファイル**

| ファイル名 |
|---|
| CRC_Verify.h, CRC_Verify.c |
| CRC.h, CRC.c |

| 構文 | |
|---|---|
| `bool_t CRC_Verify(const uint16_t ui16_NewCRCValue, const uint32_t ui32_AddrRefCRC)` | |
| **説明** | |
| **この関数は、基準 CRC が格納されるアドレスを提供して新しい CRC 値と基準 CRC を比較します。** | |
| **入力パラメータ** | |
| uint16_t ui16_NewCRCValue | **計算された新しい CRC 値** |
| uint32_t ui32_AddrRefCRC | 16 **ビット基準 CRC 値が格納されるアドレス** |
| **出力パラメータ** | |
| **なし** | **該当せず** |
| **戻り値** | |
| bool_t | **テスト結果：**TRUE = **成功、**FALSE = **失敗** |

以下の関数はファイル CRC.h および CRC.c に実装されます。

| 構文 | |
|---|---|
| `uint16_t CRC_Init(void)` | |
| **説明** | |
| CRC **モジュールを初期化します。この関数は、その他の** CRC **関数を呼び出す前に呼び出さなければなりません。** | |
| **入力パラメータ** | |
| uint8_t* pui8_DataBuf | **テストするメモリの先頭を指すポインタ** |
| uint32_t ui32_DataBufSize | **データの長さ（バイト）** |
| **出力パラメータ** | |
| **なし** | **該当せず** |
| **戻り値** | |
| uint16_t | 16 **ビット** CRC-CCITT **計算値** |

| 構文 | |
|---|---|
| uint16_t CRC_Calculate(uint8_t* pui8_Data, uint32_t ui32_Length) | |
| **説明** | |
| この関数は 1 つの指定されたメモリ領域の CRC を計算します。 | |
| **入力パラメータ** | |
| uint8_t* pui8_DataBuf | テストするメモリの先頭を指すポインタ |
| uint32_t ui32_DataBufSize | データの長さ（バイト） |
| **出力パラメータ** | |
| なし | 該当せず |
| **戻り値** | |
| bool_t | 16 ビット CRC-CCITT 計算値 |

以下の関数は、メモリ領域を単純に開始アドレスと長さで指定することができない場合に使用されます。これにより、メモリ領域の範囲やセクションを追加することができます。これは、関数 CRC_Calculate が 1 つの関数呼び出しで時間がかかりすぎる場合に使用することもできます。

| 構文 | |
|---|---|
| void CRC_Start(void) | |
| **説明** | |
| モジュールがデータの受信を開始するための準備をします。関数 CRC_AddRange を使用する前にこれを一度呼び出します。 | |
| **入力パラメータ** | |
| なし | 該当せず |
| **出力パラメータ** | |
| なし | 該当せず |
| **戻り値** | |
| なし | 該当せず |

| 構文 | |
|---|---|
| void CRC_AddRange(uint8_t* pui8_Data, uint32_t ui32_Length) | |
| **説明** | |
| 複数のアドレス範囲から構成されるデータの CRC を計算する場合は、CRC_Calculate ではなく、この関数を使用します。必要な各アドレス範囲に対して最初に CRC_Start を呼び出してから、CRC_AddRange を呼び出し、その後 CRC_Result を呼び出して CRC 値を取得します。 | |
| **入力パラメータ** | |
| uint8_t* pui8_DataBuf | テストするメモリの先頭を指すポインタ |
| uint32_t ui32_DataBufSize | データの長さ（バイト） |
| **出力パラメータ** | |
| なし | 該当せず |
| **戻り値** | |
| なし | 該当せず |

| 構文 | |
|---|---|
| `int16_t CRC_Result(void)` | |
| **説明** | |
| CRC_Start が呼び出された後に関数 CRC_AddRange を使用して追加されたすべてのメモリ領域に対する CRC 値を計算します。 | |
| **入力パラメータ** | |
| **なし** | **該当せず** |
| **出力パラメータ** | |
| **なし** | **該当せず** |
| **戻り値** | |
| uint16_t | CRC-CCITT 計算値 |

## 1.3 RAM

マーチテストは効果的な RAM テスト方法として広く認識されている一連のテストです。

マーチテストは限定された一連のマーチエレメントから構成され、マーチエレメントは、次のセルに進む前にメモリアレイ内のすべてのセルに適用される限定された一連の操作です。

一般に、アルゴリズムがより多くのエレメントから構成される場合、その不具合の検出対象範囲は広くなりますが、実行時間は遅くなります。

アルゴリズム自体は破壊的ですが（現在の RAM 値は保存されません）、提供されたテスト関数では、メモリ内容を保存できるように非破壊オプションが用意されています。これは、実際のアルゴリズムを実行する前に提供されるバッファにメモリをコピーし、テストの最後にバッファからメモリを復元することにより実現しています。API には、バッファと RAM テスト領域を自動的にテストするためのオプションがあります。

テストする RAM の領域は、テスト中に他のことに使用することはできません。このため、スタックのために使用する RAM のテストは特に困難になります。この問題を解決するために、API にはスタックのテストに使用できる関数が用意されています。

以下のセクションでは、特定のマーチテストについて説明します。その後にソフトウェア API の仕様を示します。

### 1.3.1 アルゴリズム

#### (1) マーチ C

マーチ C アルゴリズム（van de Goor 1991）は、合計 10 の操作のある 6 つのマーチエレメントから構成されます。以下の不具合を検出します。

1. 縮退故障（SAF）
   — セルまたはラインの論理値は常に 0 または 1 です。
2. 遷移不具合（TF）
   — 0 ⇒ 1 または 1 ⇒ 0 の遷移ができないセルまたはライン
3. カップリング不具合（CF）
   — 1 つのセルへの書き込み操作を行うと、2 番目のセルの内容が変更されます。
4. アドレスデコーダ不具合（AF）
   — アドレスデコーダに影響を与える不具合：
   — 特定のアドレスでセルにアクセスできない。
   — 特定のセルにアクセスできない。
   — 特定のアドレスから複数のセルが同時にアクセスされる。
   — 特定のセルに複数のアドレスからアクセスされる。

これらは 6 つのマーチエレメントです。

I. アレイにすべてゼロを書き込みます。
II. 最下位アドレスから始まり、ゼロを読み出し、1 を書き込み、アレイをビットごとにインクリメントします。
III. 最下位アドレスから始まり、1 を読み出し、ゼロを書き込み、アレイをビットごとにインクリメントします。
IV. 最上位アドレスから始まり、ゼロを読み出し、1 を書き込み、アレイをビットごとにデクリメントします。
V. 最上位アドレスから始まり、1 を読み出し、ゼロを書き込み、アレイをビットごとにデクリメントします。
VI. アレイがすべてゼロであることを読み出します。

(2) マーチ X

**【注】 このアルゴリズムは RX62T には実装されていないので、下記のマーチ X WOM バージョンに関連するため参考としてのみここに掲載します。**

マーチ X アルゴリズムは、合計 6 の操作で 4 つのマーチエレメントから構成されます。以下の不具合を検出します。

1. 縮退故障（SAF）
2. 遷移不具合（TF）
3. 反転カップリング不具合（Cfin）
4. アドレスデコーダ不具合（AF）

これらは 4 つのマーチエレメントです。

I. アレイにすべてゼロを書き込みます。
II. 最下位アドレスから始まり、ゼロを読み出し、1 を書き込み、アレイをビットごとにインクリメントします。
III. 最上位アドレスから始まり、1 を読み出し、ゼロを書き込み、アレイをビットごとにデクリメントします。
IV. アレイからすべてゼロを読み出します。

(3) マーチ X（ワード指向メモリバージョン）

マーチ X ワード指向メモリ（WOM）アルゴリズムは、標準マーチ X アルゴリズムから 2 段階で作成されました。最初に標準マーチ X は単一ビットデータパターンの使用からメモリアクセス幅に等しいデータパターンの使用に変換されます。この段階では、テストによってアドレスデコーダ不具合を含むワード間不具合が主に検出されます。2 番目の段階では、追加の 2 つのマーチエレメントを追加します。1 つのエレメントは交互の高/低ビットのデータパターンを使用し、2 番目のエレメントは反転値を使用します。これらのエレメントを追加して、ワード間カップリング不具合を検出します。

これらは 6 つのマーチエレメントです。

I. アレイにすべてゼロを書き込みます。
II. 最下位アドレスから開始し、ゼロを読み出し、1 を書き込み、アレイをワードごとにインクリメントします。
III. 最上位アドレスから開始し、1 を読み出し、ゼロを書き込み、アレイをワードごとにデクリメントします。
IV. 最下位アドレスから開始し、ゼロを読み出し、h'AA を書き込み、アレイをワードごとにインクリメントします。
V. 最上位アドレスから開始し、h'AA を読み出し、h'55 を書き込み、アレイをワードごとにデクリメントします。
VI. 全てのアレイから h'55 を読み出します。

### 1.3.2　ソフトウェア API

#### (1)　マーチ C API

このテストは 8、16 または 32 ビット RAM アクセスを使用するように構成されています。

これは、ヘッダファイルに RAMTEST_MARCH_C_ACCESS_SIZE を以下のいずれかになるように定義することにより実現します。

- RAMTEST_MARCH_C_ACCESS_SIZE_8BIT
- RAMTEST_MARCH_C_ACCESS_SIZE_16BIT
- RAMTEST_MARCH_C_ACCESS_SIZE_32BIT

1 つの関数呼び出しでテストすることができる RAM の最大サイズを指定すると、テストを高速化し、スタックとコードサイズを低減することができます。このためには、テスト領域に含まれる「ワード」数を保持するために使用する変数のサイズを指定します。「ワード」サイズは選択されたアクセス幅です。

これは、ヘッダファイルに RAMTEST_MARCH_C_MAX_WORDS を以下のいずれかになるように定義することにより実現します。

- RAMTEST_MARCH_C_MAX_WORDS_8BIT　（テスト領域の最大ワードは 0xFF です。）
- RAMTEST_MARCH_C_MAX_WORDS_16BIT（テスト領域の最大ワードは 0xFFFF です。）
- RAMTEST_MARCH_C_MAX_WORDS_32BIT（テスト領域の最大ワードは 0xFFFFFFFF です。）

**表 3　ソースファイル**

| ファイル名 |
|---|
| ramtest_march_c.h |
| ramtest_march_c.c |

ソースは ANSI C で作成され、ファイル MisraTypes.h に宣言されているように MISRA に準拠するデータ型を使用しています。

**【注】　API により、関数呼び出しを使用して 1 つだけのワードをテストすることができます。しかし、ワード間でカップリング不具合をテストするには、関数を使用して 1 ワードより大きいデータ範囲をテストすることが重要です。**

| **宣言** | |
|---|---|
| `bool_t RamTest_March_C(uint32_t ui32_StartAddr, uint32_t ui32_EndAddr, void* p_RAMSafe);` | |
| **説明** | |
| **マーチ C（Goor 1991）アルゴリズムを使用した RAM メモリテスト** | |
| **入力パラメータ** | |
| ui32_StartAddr | **テストする RAM の最初のワードのアドレス。これは選択されたメモリアクセス幅と合わせなければなりません。** |
| Ui32_EndAddr | **テストする RAM の最後のワードのアドレス。これは選択されたメモリアクセス幅に合わせ、ui32_StartAddr 以上の値でなければなりません。** |
| P_RAMSafe | **破壊メモリテストの場合、NULL に設定します。**<br>**非破壊メモリテストの場合は、テスト領域の内容をコピーするのに十分な大きさで、選択されたメモリアクセス幅に合うバッファの先頭に設定します。** |
| **出力パラメータ** | |
| **なし** | **該当せず** |
| **戻り値** | |
| bool_t | TRUE = **テストはパスしました。**FALSE = **テストまたはパラメータチェックはフェイルとなりました。** |

| 宣言 | |
|---|---|
| bool_t RamTest_March_C_**Extra**(uint32_t ui32_StartAddr, uint32_t ui32_EndAddr, void* p_RAMSafe); | |
| **説明** | |
| **マーチ** C（Goor 1991）**アルゴリズムを使用した非破壊** RAM **メモリテスト**<br>**この関数は、**RamTest_March_C **関数とは異なり、使用する前に**'RAMSafe'**バッファをテストします。**<br>'RAMSafe'**バッファのテストが失敗した場合は、テストは異常終了し、関数は** FALSE **を返します。** | |
| **入力パラメータ** | |
| ui32_StartAddr | **テストする** RAM **の最初のワードのアドレス。これは選択されたメモリアクセス幅と合わせなければなりません。** |
| Ui32_EndAddr | **テストする** RAM **の最後のワードのアドレス。これは選択されたメモリアクセス幅に合わせ、**ui32_StartAddr **以上の値でなければなりません。** |
| P_RAMSafe | **テスト領域の内容をコピーするのに十分な大きさで、選択されたメモリアクセス幅に合うバッファの先頭に設定します。** |
| **出力パラメータ** | |
| **なし** | **該当せず** |
| **戻り値** | |
| bool_t | TRUE = **テストはパスしました。**FALSE = **テストまたはパラメータチェックはフェイルとなりました。** |

(2)　マーチ X WOM API

このテストは 8、16 または 32 ビット RAM アクセスを使用するように構成されています。

これは、ヘッダファイルに RAMTEST_MARCH_X_WOM_ACCESS_SIZE を以下のいずれかになるように定義することにより実現します。

- RAMTEST_MARCH_ X_WOM_ACCESS_SIZE_8BIT
- RAMTEST_MARCH_ X_WOM_ACCESS_SIZE_16BIT
- RAMTEST_MARCH_ X_WOM_ACCESS_SIZE_32BIT

テストの実行時間を高速化するために、1 つの関数呼び出しでテストすることができる RAM の最大サイズを指定することができます。このためには、テスト領域に含まれる「ワード」数を保持するために使用する変数のサイズを指定します。「ワード」サイズは選択されたアクセス幅と同じです。

これは、ヘッダファイルに RAMTEST_MARCH_ X_WOM_MAX_WORDS を以下のいずれかになるように定義することにより実現します。

- RAMTEST_MARCH_X_MAX_WORDS_8BIT　（テスト領域の最大ワードは 0xFF です。）
- RAMTEST_MARCH_X_MAX_WORDS_16BIT（テスト領域の最大ワードは 0xFFFF です。）
- RAMTEST_MARCH_X_MAX_WORDS_32BIT（テスト領域の最大ワードは 0xFFFFFFFF です。）

**表 4　ソースファイル**

| ファイル名 |
|---|
| ramtest_march_x_wom.h |
| ramtest_march_x_wom.c |

ソースは ANSI C で作成され、ファイル MisraTypes.h に宣言されているように MISRA に準拠するデータ型を使用しています。

**【注】　API により、関数呼び出しを使用して 1 つだけのワードをテストすることができます。しかし、ワード間でカップリング不具合をテストするには、関数を使用して 1 ワードより大きいデータ範囲をテストすることが重要です。**

| **宣言** | |
|---|---|
| `bool_t RamTest_March_X_WOM(uint32_t ui32_StartAddr, uint32_t ui32_EndAddr, void* p_RAMSafe);` | |
| **説明** | |
| WOM のために変換されたマーチ X アルゴリズムに基づく RAM メモリテスト | |
| **入力パラメータ** | |
| ui32_StartAddr | テストする RAM の最初のワードのアドレス。これは選択されたメモリアクセス幅と合わせなければなりません。 |
| Ui32_EndAddr | テストする RAM の最後のワードのアドレス。これは選択されたメモリアクセス幅に合わせ、ui32_StartAddr 以上の値でなければなりません。 |
| P_RAMSafe | 破壊メモリテストの場合、NULL に設定します。<br>非破壊メモリテストの場合は、テスト領域の内容をコピーするのに十分な大きさで、選択されたメモリアクセス幅に合うバッファの先頭に設定します。 |
| **出力パラメータ** | |
| **なし** | **該当せず** |
| **戻り値** | |
| bool_t | TRUE = テストはパスしました。<br>FALSE = テストまたはパラメータチェックはフェイルとなりました。 |

| 宣言 | |
|---|---|
| bool_t RamTest_March_X_WOM_**Extra**(uint32_t ui32_StartAddr, uint32_t ui32_EndAddr,<br>void* p_RAMSafe); | |
| **説明** | |
| WOM のために変換されたマーチ X アルゴリズムに基づく非破壊 RAM メモリテスト。この関数は、RamTest_March_X_WOM_XXBit 関数とは異なり、使用する前に'RAMSafe'バッファをテストします。'RAMSafe'バッファのテストが失敗した場合は、テストは異常終了し、関数は FALSE を返します。 | |
| **入力パラメータ** | |
| ui32_StartAddr | テストする RAM の最初のワードのアドレス。これは選択されたメモリアクセス幅と合わせなければなりません。 |
| Ui32_EndAddr | テストする RAM の最後のワードのアドレス。これは選択されたメモリアクセス幅に合わせ、ui32_StartAddr 以上の値でなければなりません。 |
| P_RAMSafe | テスト領域の内容をコピーするのに十分な大きさで、選択されたメモリアクセス幅に合うバッファの先頭に設定します。 |
| **出力パラメータ** | |
| **なし** | **該当せず** |
| **戻り値** | |
| bool_t | TRUE = テストはパスしました。<br>FALSE = テストまたはパラメータチェックはフェイルとなりました。 |

(3)　RAM テストスタック API

この API により、スタックを含む RAM の領域に対して RAM テストを実行することができます。RAM テストを実行する関数はスタックを必要とするので、これらの関数は提供される新しい RAM 領域にスタックを再配置し、元のスタック領域をテストすることができます。どのスタック（ユーザまたは割り込み）がテスト領域にあるか、または両方がテスト領域にあるかどうかによって呼び出すことができる 3 つの関数が用意されています。

**表 5　ソースファイル**

| ファイル名 |
|---|
| ramtest_stack.h |
| ramtest_stack.c |

| 宣言 | |
|---|---|
| bool_t RamTest_Stack_User(uint32_t ui32_StartAddr,<br>uint32_t ui32_EndAddr,<br>void* p_RAMSafe,<br>uint32_t ui32_NewUSP,<br>TEST_FUNC fpTest_Func); | |
| **説明** | |
| **割り込みスタックではなく、ユーザスタックを含む領域の** RAM **テスト。** | |
| **入力パラメータ** | |
| ui32_StartAddr | **テストする** RAM **の最初のワードのアドレス。これは** fpTest_Func **の要件に適応しなければなりません。** |
| Ui32_EndAddr | **テストする** RAM **の最後のワードのアドレス。これは** fpTest_Func **の要件に適応しなければなりません。** |
| P_RAMSafe | **テスト** RAM **領域と同じサイズのバッファの先頭に設定します。これは** fpTest_Func **の要件に適応しなければなりません。** |
| Ui32_NewUSP | **再配置されるユーザスタックの新しいスタックポインタ値** |
| fpTest_Func | **使用する実際のメモリテストを指すタイプ** TEST_FUNC **の関数ポインタ**<br>Typedef bool_t(*TEST_FUNC)( uint32_t, uint32_t, void*);<br>**たとえば、**'RamTest_March_X_WOM'**です。** |
| **出力パラメータ** | |
| **なし** | **該当せず** |
| **戻り値** | |
| bool_t | TRUE = **テストはパスしました。**<br>FALSE = **テストまたはパラメータチェックはフェイルとなりました。** |

| 宣言 | |
|---|---|
| bool_t RamTest_Stack_Int(uint32_t ui32_StartAddr,<br>uint32_t ui32_EndAddr,<br>void* p_RAMSafe,<br>uint32_t ui32_NewISP,<br>TEST_FUNC fpTest_Func); | |
| **説明** | |
| **ユーザスタックではなく、割り込みスタックを含む領域の** RAM **テスト。** | |
| **入力パラメータ** | |
| ui32_StartAddr | **テストする** RAM **の最初のワードのアドレス。これは** fpTest_Func **の要件に適用しなければなりません。** |
| Ui32_EndAddr | **テストする** RAM **の最後のワードのアドレス。これは** fpTest_Func **の要件に適用しなければなりません。** |

| | |
|---|---|
| P_RAMSafe | テスト RAM 領域と同じサイズのバッファの先頭に設定します。これは fpTest_Func の要件に適用しなければなりません。 |
| Ui32_NewISP | 再配置される割り込みスタックの新しいスタックポインタ値 |
| fpTest_Func | 使用する実際のメモリテストを指すタイプ TEST_FUNC の関数ポインタ<br>Typedef bool_t(*TEST_FUNC)( uint32_t, uint32_t, void*);<br>たとえば、'RamTest_March_X_WOM'です。 |
| **出力パラメータ** | |
| **なし** | **該当せず** |
| **戻り値** | |
| bool_t | TRUE = テストはパスしました。<br>FALSE = テストまたはパラメータチェックはフェイルとなりました。 |

| **宣言** |
|---|
| `bool_t RamTest_Stack_Int(uint32_t ui32_StartAddr,`<br>`uint32_t ui32_EndAddr,`<br>`void* p_RAMSafe,`<br>`uint32_t ui32_NewISP,`<br>`TEST_FUNC fpTest_Func);` |
| **説明** |
| ユーザスタックではなく、割り込みスタックを含む領域の RAM テスト。 |
| **入力パラメータ** |
| ui32_StartAddr | テストする RAM の最初のワードのアドレス。これは fpTest_Func の要件に適用しなければなりません。 |
| Ui32_EndAddr | テストする RAM の最後のワードのアドレス。これは fpTest_Func の要件に適用しなければなりません。 |
| P_RAMSafe | テスト RAM 領域と同じサイズのバッファの先頭に設定します。これは fpTest_Func の要件に適用しなければなりません。 |
| Ui32_NewISP | 再配置される割り込みスタックの新しいスタックポインタ値 |
| Ui32_NewUSP | 再配置されるユーザスタックの新しいスタックポインタ値 |
| fpTest_Func | 使用する実際のメモリテストを指すタイプ TEST_FUNC の関数ポインタ<br>Typedef bool_t(*TEST_FUNC)(const uint32_t, const uint32_t, void* const);<br>たとえば、'RamTest_March_X_WOM'です。 |
| **出力パラメータ** | |
| **なし** | **該当せず** |
| **戻り値** | |
| bool_t | TRUE = テストはパスしました。<br>FALSE = テストまたはパラメータチェックはフェイルとなりました。 |

## 1.4　クロック

RX62T GPT モジュールは、低速 OCO クロックを使用して実行時にメインクロック（ICLK）周波数の偏差を検出する LOCO 関数を持っています。

**【注】** IWDT により低速 OCO が有効になるので、これを使用する前に IWDT モジュールを有効にしてください。

**【重要事項】** E1 デバッガは、コードを実行していないときには IWDT を無効にするので、この LOCO 関数とともに使用することはできません。これは LOCO クロックに影響を及ぼし、LOCO 関数は LCLK の偏差を誤って報告します。

メインクロックの周波数が実行時に構成された範囲から偏差がある場合は、エラーコールバック関数を呼び出さなければなりません。許容可能な周波数範囲は以下を使用して調整することができます。

```
/*Percentage tolerance of ICLK allowed before an
error is reported. (integer value)*/
#define CLOCK_TOLERANCE_PERCENT  5
```

LOCO 関数だけでなく、RX62T は発振停止検出回路を備えています。これはデフォルトではパワーオンリセットにより有効になります。メインクロックが停止した場合、その代わりに低速オンチップオシレータが自動的に使用され、NMI 割り込みが生成されます。このモジュールのユーザは、NMI 割り込みを処理し、NMISR.OSTST ビットをチェックしてください。

**【注】** 発振停止検出回路は、低速オンチップオシレータが停止したかどうかを検出せず、メインクロックが停止したかどうかのみを検出します

**表 6　ソースファイル**

| ファイル名 |
|---|
| ramtest_monitor.h |
| ramtest_monitor.c |

| 構文 | |
|---|---|
| `void ClockMonitor_Init (CLOCK_MONITOR_ERROR_CALL_BACK CallBack)` | |
| **説明** | |
| LOCO（低速 OCO）を使用してメインクロック（ICLK）を監視します。また、発振停止検出のために NMI 割り込みを有効にします。 | |
| **入力パラメータ** | |
| CallBack | メインクロックが許容範囲から偏差がある場合に呼び出されるコールバック関数 |
| **出力パラメータ** | |
| なし | 該当せず |
| **戻り値** | |
| なし | 該当せず |

### 1.4.1　設計に関する注意事項

LOCO の精度が保証されないので、LOCO はメインクロックの絶対値を測定することはできません。ソフトウェアを最初に実行するときにメインクロックがその値からどれくらい偏差があるかを検出することができます。

汎用タイマの LOCO 関数は ICLK に比例する現在の平均値を計算します。この平均値に対して ICLK の偏差を検出できるようにするためには、その前に平均値が計算できるようになるまでクロックを動作させてください。このモジュールは以下のフローを使用してこれを行います。

1. ユーザは ClockMonitor_Init 関数を呼び出します。
   LOCO 関数は、偏差を検出するためでなく平均を計算する場合に有効です。255 の周波数 LOCO 分割クロックの後に割り込みが生じるように構成されます。
2. LOCO 割り込みが発生する場合：
   計算された平均値に基づいて許容可能な ICLK 値範囲を構成します。偏差の検出を有効にします。
3. 別の LOCO 割り込みが発生した場合：
   つまり、最新の ICLK 値が構成された許容範囲を超えて偏差があります。
   ユーザが登録したコールバック関数を呼び出します。

## 1.5　ウォッチドッグ

ウォッチドッグは異常なプログラム実行を検出するために使用します。プログラムが適切に動作していない場合は、必要に応じてウォッチドッグタイマがソフトウェアによってリフレッシュされず、そのためエラーが検出されます。

このために RX62T の独立ウォッチドッグタイマ（IWDT）モジュールを使用します。ウォッチドッグがタイムアウトすると、内部リセットが行われます。IWDT によりリセットが行われたかどうかを決定するためにリセット後に使用する関数が用意されています。

**表 7　ソースファイル**

| ファイル名 |
|---|
| IWDT.h |
| IWDT.c |

| **構文** | |
|---|---|
| `void IWDT_Init (IWDT_TOP TimeOutperiod, IWDT_CKS_DIV ClockSelection)` | |
| **説明** | |
| **ウォッチドッグタイマを初期化し起動します。パラメータはウォッチドッグがタイムアウトするまでの時間を指定します。これを呼び出した後、ウォッチドッグのタイムアウトを防止するために** IWDT_kick **関数を定期的に呼び出さなければなりません。** | |
| **入力パラメータ** | |
| IWDT_TOP TimeOutperiod | **詳細については、**IWDT.h **内の列挙型** IWDT_TOP **の宣言を参照してください。** |
| IWDT_CKS_DIV ClockSelection | **詳細については、**IWDT.h **内の列挙型** IWDT_CKS_DIV **の宣言を参照してください。** |
| **出力パラメータ** | |
| **なし** | **該当せず** |
| **戻り値** | |
| **なし** | **該当せず** |

| 構文 | |
|---|---|
| `void IWDT_Kick(void)` | |
| 説明 | |
| ウォッチドッグカウントをリフレッシュします。ウォッチドッグカウントがタイムアウトする前にこれを呼び出してください。 | |
| 入力パラメータ | |
| なし | 該当せず |
| 出力パラメータ | |
| なし | 該当せず |
| 戻り値 | |
| なし | 該当せず |

| 構文 | |
|---|---|
| `bool_t IWDT_DidReset(void)` | |
| 説明 | |
| IWDT がタイムアウトした場合に返ります。これは、ウォッチドッグによりリセットが行われたかどうかを決定するためにリセット後に呼び出すことができます。 | |
| 入力パラメータ | |
| なし | 該当せず |
| 出力パラメータ | |
| なし | 該当せず |
| 戻り値 | |
| bool_t | ウォッチドッグがタイムアウトした場合、TRUE、それ以外の場合、FALSE です。 |

## 1.6　電圧

RX62T は電圧検出回路を搭載しています。これを使用して電源電圧（Vcc）が基準電圧 Vdet1 および Vdet2 より低くなったことを検出することができます。各基準電圧について、割り込みまたは内部リセットを発生させるように設定することができます。電圧監視によりリセットが行われたかどうかを決定するためにリセット後に使用する関数が用意されています。

**【注】 割り込みを発生するように設定された場合、これはノンマスカブル割り込み（NMI）となります。これはユーザコードによって処理し、ユーザコードは NMISR.LVDST フラグをチェックしてください。**

**表 8　ソースファイル**

| ファイル名 |
|---|
| Voltage.h |
| Voltage.c |

| 構文 | |
|---|---|
| `void VoltageMonitor_Init(bool_t bEnable_LVD1, bool_t bResetLVD1, bool_t bEnable_LVD2, bool_t bResetLVD2, VOLTAGE_MONITOR_ERROR_CALL_BACK Callback)` | |
| **説明** | |
| **電圧監視を初期化し起動します。**<br>**【注】 割り込みを発生するように構成された場合、これはノンマスカブル割り込み（NMI）となります。これはユーザコードによって処理してください。** | |
| **入力パラメータ** | |
| bool_t bEnable_LVD1 | LVD1 **モニタを有効にします。** |
| bool_t bResetLVD1 | LVD1 **低電圧検出後にリセットする場合は、**TRUE **をセットします。割り込みを発生する場合は、**FALSE **を設定します。** |
| bool_t bEnable_LVD2 | LVD2 **モニタを有効にします。** |
| bool_t bResetLVD2 | LVD2 **低電圧検出後にリセットする場合は、**TRUE **をセットします。割り込みを発生する場合は、**FALSE **を設定します。** |
| **出力パラメータ** | |
| **なし** | **該当せず** |
| **戻り値** | |
| **なし** | **該当せず** |

| 構文 | |
|---|---|
| `bool_t VoltageMonitor_DidReset(void)` | |
| **説明** | |
| **電圧監視が低電圧を検出した場合に返ります。電圧監視によってリセットが行われたかどうかを決定するためにリセット後に呼び出すことができます。** | |
| **入力パラメータ** | |
| **なし** | **該当せず** |
| **出力パラメータ** | |
| **なし** | **該当せず** |
| **戻り値** | |
| bool_t | **低電圧レベルが検出された場合、**TRUE**、それ以外の場合、**FALSE **です。** |

## 1.7　ADC10

ADC10 モジュールには、ADC10 モジュールをテストするために使用することができる診断モードがあります。

ADC10 をテストしているときに、ADC10 を通常の動作に使用することはできません。

ADC10 完了割り込みハンドラ（およびベクタエントリ）を提供するか、ユーザの ADC10 完了割り込みハンドラから呼び出さなければならない関数を提供するようにソフトウェアを構成することができます。

詳細については、#define ADC10_PROVIDE_INTERRUPT_HANDLER を参照してください。

**表 9　ソースファイル**

| ファイル名 |
|---|
| ADC10.h |
| ADC 10.c |

| 構文 | |
|---|---|
| `void Test_ADC10_Init(void)` | |
| **説明** | |
| ADC10 **モジュールを初期化します。** | |
| **入力パラメータ** | |
| **なし** | **該当せず** |
| **出力パラメータ** | |
| **なし** | **該当せず** |
| **戻り値** | |
| **なし** | **該当せず** |

| 構文 | |
|---|---|
| `bool_t Test_ADC10_Wait(void);` | |
| **説明** | |
| ADC10 **モジュールをテストし、結果を返します。**<br>**基準電圧** 0 **および** VREF **を使用します。関数は** 2 **つの変換を待ちます。これは電源投入テストに適しています。** | |
| **入力パラメータ** | |
| **なし** | **該当せず** |
| **出力パラメータ** | |
| **なし** | **該当せず** |
| **戻り値** | |
| bool_t | TRUE = **テストはパスしました。**FALSE = **テストはフェイルとなりました。** |

| 構文 | |
|---|---|
| `void Test_ADC10_Start(ADC10_ERROR_CALL_BACK Callback)` | |
| **説明** | |
| **診断テストのために AD 変換を開始します。変換が終了したら、結果をチェックしてください。**<br>ADC10_PROVIDE_INTERRUPT_HANDLER **が定義されている場合は、このモジュールがそれを自動的に実行します。定義されていない場合は、変換が終了したときにユーザが関数** ADC10_Interrupt **を呼び出さなければなりません。**<br>**この関数を呼び出すたびに、基準電圧が** 0V **と** VREF **の間で切り替わります。** | |
| **入力パラメータ** | |
| ADC10_ERROR_CALL_BACK Callback | **エラーが検出された場合に呼び出される関数** |
| **出力パラメータ** | |
| **なし** | **該当せず** |
| **戻り値** | |
| **なし** | **該当せず** |

## 1.8　ADC12

ADC12 モジュールには、ADC をテストするために使用することができる診断モードがあります。診断モードは、変換のために ADC を正しく使用するたびにテストが実行されるように構成することができます。RX62T には 2 つの AD12 モジュールがあります。これらの関数を使用して各モジュールを独立してテストすることができます。

**表 10　ソースファイル**

| ファイル名 |
|---|
| ADC12.h |
| ADC 12.c |

| 構文 | |
|---|---|
| `void Test_ADC12_Init(ADC12_MODULE module)` | |
| **説明** | |
| **指定された** ADC12 **モジュールを初期化します。** | |
| **入力パラメータ** | |
| ADC12_MODULE module | ADC **モジュール（**ADC12_MODULE_0 **または** ADC12_MODULE_1**）** |
| **出力パラメータ** | |
| **なし** | **該当せず** |
| **戻り値** | |
| **なし** | **該当せず** |

| 構文 | |
|---|---|
| `bool_t Test_ADC12_Wait(ADC12_MODULE module)` | |
| **説明** | |
| **指定された** ADC12 **モジュールをテストします。この関数は** 2 **つの** AD **変換が行われているときに待ち状態となります。このテストでは** ADC **構成は保存されないため、実行時の定期テストではなくパワーオンテストに適しています。** | |
| **入力パラメータ** | |
| ADC12_MODULE module | ADC **モジュール（**ADC12_MODULE_0 **または** ADC12_MODULE_1**）** |
| **出力パラメータ** | |
| **なし** | **該当せず** |
| **戻り値** | |
| bool_t | TRUE = **テストはパスしました。**FALSE = **テストはフェイルとなりました。** |

| 構文 | |
|---|---|
| `void Test_ADC12_Start(ADC12_MODULE module, ADC12_ERROR_CALL_BACK Callback)` | |
| **説明** | |
| ADC を使用するたびに診断テストを実行するように ADC モジュールをセットアップします。基準電圧が自動的に切り換えられます。（ゼロ、1/2 VREF および VREH）<br>ユーザコードは、Test_ADC12_CheckResult 関数を定期的に呼び出すか、または ADC が完了するたびに診断結果をチェックするために呼び出さなければなりません。 | |
| **入力パラメータ** | |
| ADC12_MODULE module | ADC モジュール（ADC12_MODULE_0 または ADC12_MODULE_1） |
| ADC12_ERROR_CALL_BACK Callback | エラーが検出された場合に呼び出される関数<br>【注】 この関数は、パラメータ bCallErrorHandler を TRUE に設定して Test_ADC12_CheckResult が呼び出された場合にのみ呼び出されます。 |
| **出力パラメータ** | |
| なし | 該当せず |
| **戻り値** | |
| なし | 該当せず |

| 構文 | |
|---|---|
| `bool_t Test_ADC12_CheckResult(ADC12_MODULE module, bool_t bCallErrorHandler)` | |
| **説明** | |
| ADC 診断結果が期待どおりであることを確認します。<br>これは Test_ADC12_Start の後に呼び出し、その後、定期的に呼び出すか、AD 変換が終了するたびに呼び出します。 | |
| **入力パラメータ** | |
| ADC12_MODULE module | ADC モジュール（ADC12_MODULE_0 または ADC12_MODULE_1） |
| bool_t bCallErrorHandler | 関数 Test_ADC12_Start に提供されるエラーコールバック関数を呼び出す場合は、TRUE、呼び出さない場合は、FALSE を設定します。 |
| **出力パラメータ** | |
| なし | 該当せず |
| **戻り値** | |
| bool_t | TRUE = テストはパスしました。FALSE = テストはフェイルとなりました。 |

# 2. 使用例

実際のテストソフトウェアソースファイルだけでなく、どのようにテストを実行することができるかを示すサンプルアプリケーションを含む HEW ワークスペースが提供されます。このコードは本ドキュメントとともに検討して、さまざまなテスト関数がどのように使用されているかを確認してください。

テストは 3 つの部分に分けることができます。

1. 電源投入テスト。これらはリセット後に一回実行されるテストです。これらのテストはできるだけ迅速に実行すべきですが、特に起動時間が重要な場合は、たとえばより高速なメインクロックを選択することができるようにすべてのテストを実行する前に初期化コードを実行することができます。
2. 定期テスト。これらのテストは通常のプログラム動作中に定期的に実行されるテストです。本ドキュメントは特定のテストがどれくらいの頻度で実行すべきかという判断は示しません。定期テストのスケジュールをどのように行うかは、アプリケーションがどのように構成されているかによってユーザの判断にまかされます。サンプルアプリケーションは RX62T のタイマモジュールをセットアップして、関数（PeriodicTestCallBack）を定期的に呼び出します。この関数を呼び出すたびに、特定のテストまたはその一部が実行されます。ユーザのアプリケーションの要件により、関数が呼び出されるたびにどれくらいの時間を使用することができるかが決定されます。
3. テストの監視。これは、RX62T を診断モードで使用して、あることを連続的に監視するために使用します。このため、このテストは電源投入テストまたは定期テストに分類することはできません。

以下のセクションでは、各種類のテストを使用する例を示します。

## 2.1 CPU

いずれかの CPU テストで不具合が検出された場合は、ユーザが用意する関数 CPU_Test_ErrorHandler が呼び出されます。CPU のエラーは非常に重大なので、この関数の目的はソフトウェア実行の信頼性が関連しない安全な位置にできるだけ迅速に移動することです。

### 2.1.1 電源投入テスト

すべての CPU テストはリセット後できるだけ速やかに実行する必要があります。

**【注】 関数は、デバイスがユーザモードになる前に resetprg.c 内の Change_PSW_PM_to_UserMode によって呼び出さなければなりません。**

関数 CPU_Test_All を使用すると、すべての CPU テストを自動的に実行することができます。

### 2.1.2 定期テスト

CPU を定期的にテストする場合、電源投入テストと同じように関数 CPU_Test_All を使用して、すべての CPU テストを自動的に実行することができます。あるいは、1 つの関数呼び出しで実行されるテストの量を減らすために、ユーザは CPU 定期テストをスケジュールするたびに個別の各 CPU テスト関数を順に呼び出すことができます。

## 2.2 ROM

その内容の CRC 値（CRC-CCITT）を計算して、CRC 計算に含まれない ROM の特定の位置に追加される基準 CRC 値と比較して ROM がテストされます。

ルネサス RX 標準ツールチェインを使用して、CRC 値を計算して、作成された mot ファイル内のユーザが指定した位置に追加することができます。これは HEW のダイアログを使用して行うことができます。図 1「基準 CRC の追加」を参照してください。

CRC_Init 関数を呼び出して、使用する前に CRC モジュールを初期化してください。

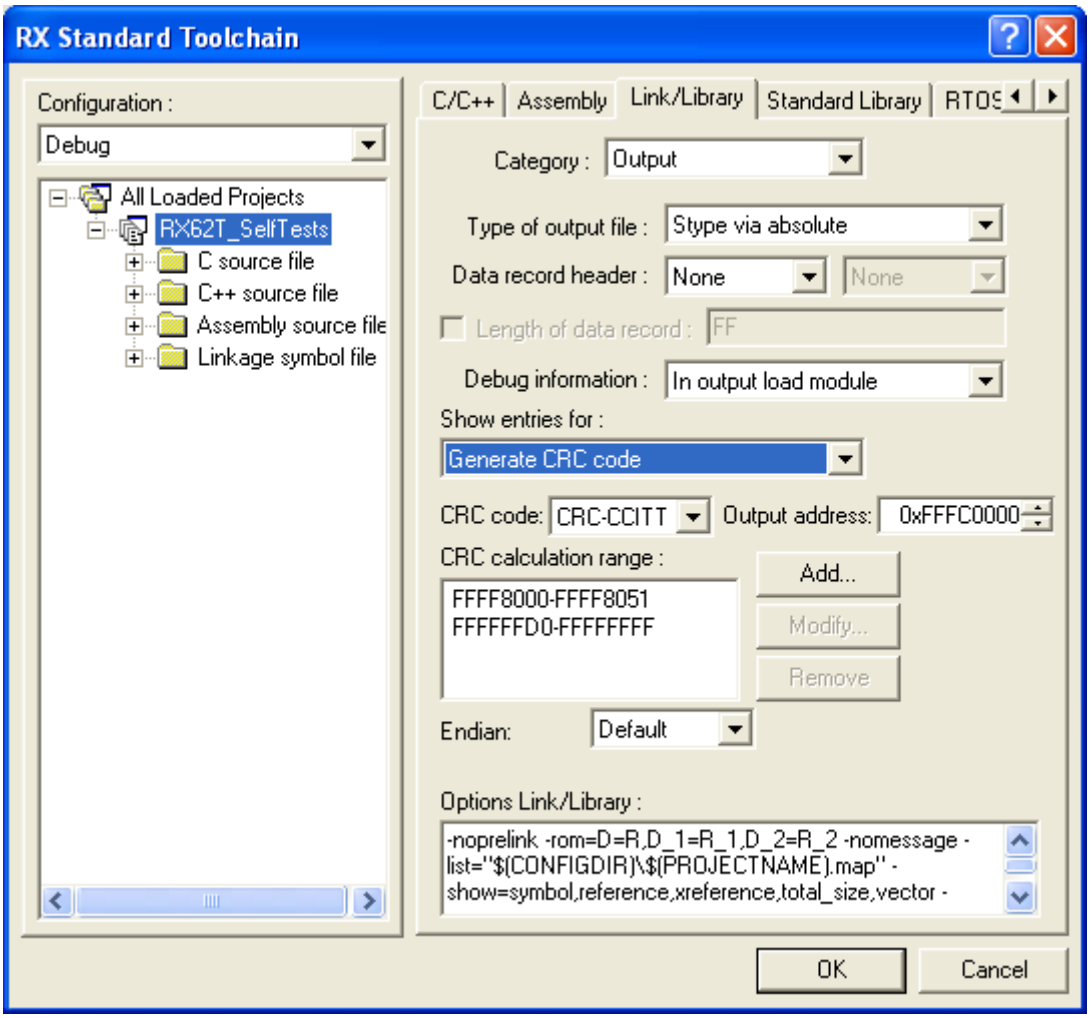
RX Standard Toolchain
Configuration :
Debug
All Loaded Projects
RX62T_SelfTests
C source file
C++ source file
Assembly source file
Linkage symbol file
C/C++
Assembly
Link/Library
Standard Library
Category :
Output
Type of output file :
Stype via absolute
Data record header :
None
None
Length of data record :
FF
Debug information :
In output load module
Show entries for :
Generate CRC code
CRC code:
CRC-CCITT
Output address:
0xFFFC0000
CRC calculation range :
FFFF8000-FFFF8051
FFFFFFD0-FFFFFFFF
Add...
Modify...
Remove
Endian:
Default
Options Link/Library :
-noprelink -rom=D=R,D_1=R_1,D_2=R_2 -nomessage -
list="$(CONFIGDIR)\$(PROJECTNAME).map" -
show=symbol,reference,xreference,total_size,vector -
OK
Cancel

**図 1　基準 CRC の追加**

### 2.2.1　電源投入テスト

使用されたすべての ROM メモリは電源投入時にテストしてください。

この領域が 1 つの連続したブロックである場合は、関数 CRC_Calculate を使用して CRC 値を計算して、計算された CRC 値を返すことができます。

使用した ROM が 1 つの連続したブロックでない場合は、以下の手順を使用します。

1. CRC_Start を呼び出します。
2. CRC 計算に含めるメモリの各領域に対して CRC_AddRange を呼び出します。
3. CRC_Result を呼び出して、計算された CRC 値を取得します。

関数 CRC_Verify を使用して、計算された CRC 値と ROM に格納された基準 CRC 値と比較することができます。

ルネサス RX コンパイラは、使用した ROM のアドレスを取得するために使用できるセクションアドレスオペレータ__sectop、__secend および__secsize を提供します。サンプルアプリケーションはこれらを使用して CRC テスト時に使用した構造を初期化します。

```
const CRC_RANGE CRC_Ranges[CRC_RANGE_NUM] =
{
  __sectop("PResetPRG"), __secend("PResetPRG"),
  __sectop("C1"), __secend("PPCTEST_TESTFUNCTION"),
  __sectop("FIXEDVECT"), __secend("FIXEDVECT")
};
```

ユーザは、プロジェクトによって使用されるすべての ROM 領域が CRC 計算に含まれることを確認する必要があります。

### 2.2.2　定期テスト

1 つの関数呼び出しに時間が長くかかりすぎないように、これにより CRC 値をセクションごとに計算することができるので、ROM が連続している場合でも CRC_AddRange メソッドを使用して ROM の定期テストを実行することを推奨します。電源投入テストに指定される手順に従い、各アドレス範囲が十分に小さく、CRC_AddRange への呼び出しが長くかかりすぎないようにしてください。

## 2.3　RAM

サンプルアプリケーションには、RAM のテストの例としてファイル Test_Usage_RAM.h および.c が含まれています。

独自のプロジェクトにこの例を使用する場合、テストする必要がある RAM の領域がプロジェクトメモリマップによって大きく変わることがあることに注意することが非常に重要です。

サンプルコードでは、RAM 領域を定義する#define をセットアップするときにいくつかの前提事項があります。ビルドのためにセットアップするときに Test_Usage_RAM.c を参照して、コメントを慎重に読んでください。

RAM をテストする際には以下のことに注意してください。

1. テストする RAM は現在のスタックを含む他の領域に使用することはできません。
2. 非破壊テストでは、メモリの内容を安全にコピーし、復元することができる RAM バッファが必要となります。
3. スタックのテストには、スタックを再配置することができる RAM バッファが必要です。
4. 割り込みスタックとユーザスタックという 2 つのスタックがあります。これは使用する前に再配置する必要がある現在のスタックです。
5. スタックを再配置する場合、デバイスはスーパーバイザモードでなければなりません。割り込みを処理する場合は、デバイスは自動的にデフォルトモードになります。

### 2.3.1　電源投入テスト

グローバル変数初期化を行う前に（_INITSCT と同様に）RAM パワーオンテストを実行する場合は、スタック以外のすべての RAM の完全な破壊テストを実行することができます。スタックは非破壊テストでテストしなければなりません。しかし、起動時間が非常に重要な場合は、これを微調整して、電源投入 RAM テストを実行する前に使用したスタックの領域のみをより低速な非破壊テストでテストして、スタックの残りを破壊テストでテストすることができます。

サンプルアプリケーションでは、電源投入時に RAM をテストする例として関数 Tests_PowerOn_RAM を使用しています。デバイスがユーザモードになる前に resetprg.c 内の Change_PSW_PM_to_UserMode によって関数を呼び出さなければなりません。

マーチ C テストアルゴリズムを使用して以下の手順を実行します。

1. RAM_START_ADDRESS から RAM_END_ADDRESS までに定義された RAM 領域に破壊テストが実行されます。（この領域はスタックと RAM_Test_Buffer を除くすべての使用された RAM を定義します。）
2. 定期 RAM テストに使用した RAM_Test_Buffer に対して破壊テストを実行します。
3. RAM_START_ADDRESS から RAM_END_ADDRESS までに定義されたスタック領域に非破壊テストが実行されます。このプロセス時にスタックは再配置されます。

### 2.3.2　定期テスト

サンプルコードはすべての定期テストでマーチ X WOM アルゴリズムを使用します。すべての定期テストは非破壊テストでなければなりません。

定期テストは割り込みハンドラによって呼び出され、そのためデバイスはスーパーバイザモードであることを前提としています。

定期テストは、スタックのテスト、RAM バッファのテスト、残りの RAM 領域のテストの 3 つのテストに分かれています。このために関数 PeriodicTest_RAM_Buffer、PeriodicTest_Stack および PeriodicTest_RAM を使用します。PeriodicTest_Stack および PeriodicTest_RAM 関数は、終了したことを返すまで定期テストスケジューラによって繰り返し呼び出されるように設計されています。これにより、これらの関数は 1 つの関数呼び出しが決して長すぎる時間がかからないような小さな部分に分割することができます。

## 2.4　クロック

メインクロックの監視は ClockMonitor_Init への 1 つの関数呼び出しによってセットアップされます。例：

```
ClockMonitor_Init(Clock_Test_Failure);
```

これは、メインクロックが構成され、IWDT が有効になると、すぐに呼び出すことができます。IWDT の有効化の詳細については、「1.5　ウォッチドッグ」を参照してください。

次にクロック監視がハードウェアによって実行されるので、定期テスト時にソフトウェアは何も実行する必要はありません。

発振停止が検出されると、NMI 割り込みが生成されます。ユーザコードはこの NMI 割り込みを処理し、この例に示すように NMISR.OSTST フラグをチェックします。

```
if(1 == ICU.NMISR.BIT.OSTST)
{
   Clock_Stop_Detection();

   /*Clear OSTST bit by writing 1 to NMICLR.OSTCLR bit*/
   ICU.NMICLR.BIT.OSTCLR = 1;
}
```

次に OSTDCR.OSTDF ステータスビットを読み出して、メインクロックのステータスを決定することができます。

## 2.5　ウォッチドッグ

独立ウォッチドッグは、IWDT_Init への呼び出しによりリセット後にできるだけ迅速に初期化してください。

```
/*Setup the Independent WDT.
IWDT_Init(IWDT_TOP_16384, IWDT_CKS_DIV_256);
```

この後、ウォッチドッグのタイムアウトとリセットを防止するようにウォッチドッグを定期的にリフレッシュしてください。このためには、以下を呼び出します。

```
/*Regularly kick the watchdog to prevent it performing a reset. */
IWDT_Kick();
```

リセット後、コードは IWDT_DidReset を呼び出して IWDT によりウォッチドッグが発生したかどうかを確認してください。

```
if(TRUE == IWDT_DidReset())
{
   //todo: Handle a watchdog reset.
   while(1){;}
}
```

## 2.6　電圧

電圧検出回路は、VoltageMonitor_Init 関数への呼び出しを使用して主電源電圧を監視するように構成されます。これはパワーオンテスト後できるだけ迅速にセットアップする必要があります。以下の例では、電圧レベルが VDET1 または VDET2 より低くなった場合にリセットを行うように電圧監視をセットアップします。

```
VoltageMonitor_Init(TRUE, TRUE, TRUE, TRUE);
```

リセット後、コードは VoltageMonitor_DidReset を呼び出して、電圧監視によりリセットが発生したかどうかを確認してください。

```
if(TRUE == VoltageMonitor_DidReset())
{
  Volatge_Test_Failure();
}
```

VoltageMonitor_Init 関数を使用して、電圧検出ペリフェラルを構成して、リセットではなく NMI 割り込みを生成することができます。この場合、NMI ハンドラはこの例のように低電圧が検出されたかどうかをチェックしてください。

```
if(1 == ICU.NMISR.BIT.LVDST)
{
   Volatge_Test_Failure();
}
```

## 2.7　ADC10

ADC10 モジュールには診断モードが組み込まれ、さまざまな基準電圧をテストすることができます。

許容される誤差に対処するために、期待される結果を以下により定義された許容差内にすることができます。

```
#define ADC10_TOLERANCE
```

この値は ADC の定格の最大絶対精度として設定されます。校正されたシステムでは、この許容差を厳しくすることができます。

### 2.7.1　電源投入テスト

電源投入時に Test_ADC10_Wait 関数を使用して ADC10 モジュールをテストすることができます。基準電圧 VREF を使用する変換と 0V を使用する変換の 2 つの AD 変換が実行されるときに、これによりブロックされます。

### 2.7.2　定期テスト

AD 変換で待たないためには、定期テストでは Test_ADC10_Start 関数を使用してください。これはノンブロッキング関数です。関数が呼び出されるたびに、テストの基準電圧は VREF と 0V の間で切り替わります。

変換終了時に変換の結果が正しいかどうかをチェックしてください。このために、完了した ADC10 割り込みが処理されます。

テストモジュールはこの割り込みハンドラを提供し、'ADC10_PROVIDE_INTERRUPT_HANDLER'が定義されている場合、結果を自動的にテストします。

**【注】　診断モードでないときに割り込みが発生した場合、ユーザが用意した関数**'ADC10_Interrupt'**が呼び出されます。**

'ADC10_PROVIDE_INTERRUPT_HANDLER'が#define で定義されていない場合は、ユーザは独自のハンドラから関数'ADC10_Interrupt'を呼び出さなければなりません。

## 2.8　ADC12

ADC12 モジュールには診断モードが組み込まれ、さまざまな基準電圧をテストすることができます。

許容される不正確さに対処するために、期待される結果を以下のように定義された許容差内にすることができます。

#define ADC12_TOLERANCE

この値は ADC の定格の最大絶対精度として設定されます。校正されたシステムでは、この許容差を厳しくすることができます。

### 2.8.1　電源投入テスト

電源投入時に Test_ADC12_Wait 関数を使用して ADC12 モジュールをテストすることができます。このブロックは 2 つの AD 変換中、基準電圧として 1 つは VREF を残りは 0V を使用します。

### 2.8.2　定期テスト

定期テストは Test_ADC12_Start に対する 1 回の呼び出しで始まります。その後、ADC12 モジュールを使用するたびに基準電圧変換を実行します。基準電圧は 0V、VREF/2 および VREF の間で切り替わります。これらの基準電圧変換の結果は Test_ADC12_CheckResult への呼び出しを使用して定期的にチェックしてください。

# 3. ベンチマーク

## 3.1 環境

開発ボード：RSKRX62T

クロック：EXTAL = 12.5MHz、ICLK = 100MHz、PCLK = 50MHZ

MCU: R5F562TA

ツールチェイン：RX 標準ツールチェイン 1.0.0.0

インサーキットデバッガ：ルネサス E1

コンパイラ設定

| | |
|---|---|
| **最大レベル サイズの 最適化** | -cpu=rx600 -lang=c<br>-include="$(PROJDIR)¥Tests¥Common","$(PROJDIR)¥Tests¥CPU","$(PROJDIR)¥Tests¥RAM","$(PROJDIR)¥Tests¥ROM","$(PROJDIR)","$(PROJDIR)¥Tests¥Clock","$(PROJDIR)¥Tests¥voltage","$(PROJDIR)¥Tests¥adc10","$(PROJDIR)¥Tests¥adc12","$(PROJDIR)¥Tests¥iwdt"<br>-output=obj="$(CONFIGDIR)¥$(FILELEAF).obj" -optimize=max<br>-map="$(CONFIGDIR)¥$(PROJECTNAME).bls" -nologo |
| **最大レベル スピードの 最適化** | -cpu=rx600 -lang=c<br>-include="$(PROJDIR)¥Tests¥Common","$(PROJDIR)¥Tests¥CPU","$(PROJDIR)¥Tests¥RAM","$(PROJDIR)¥Tests¥ROM","$(PROJDIR)","$(PROJDIR)¥Tests¥Clock","$(PROJDIR)¥Tests¥voltage","$(PROJDIR)¥Tests¥adc10","$(PROJDIR)¥Tests¥adc12","$(PROJDIR)¥Tests¥iwdt"<br>-output=obj="$(CONFIGDIR)¥$(FILELEAF).obj" -optimize=max -speed<br>-map="$(CONFIGDIR)¥$(PROJECTNAME).bls" -nologo |

**【注】** CPU **テストファイルは最適化を行わずに作成されます。**

リンカ設定

| | |
|---|---|
| **最適化 = 速度** | -map="$(CONFIGDIR)¥$(PROJECTNAME).bls" -noprelink -nodebug -rom=D=R,D_1=R_1,D_2=R_2<br>-nomessage -list="$(CONFIGDIR)¥$(PROJECTNAME).map" -optimize=speed<br>-start=B_1,R_1,B_2,R_2,B,R,SU,SI,BADC10_TEST_1,BRAM_TEST_STACK/01000,PResetPRG/0FFFF8000,C_1,C_2,C,C$*,D*,P,PIntPRG,W*,PADC10_TEST,PADC12_TEST,PCPU_TEST,PCRC,PPCTEST_TESTFUNCTION,PCLOCK_MONITOR_TEST,PVOLTAGE_TEST,PIWDT_TEST,PRAM_TEST_MarchC,PRAM_TEST_MarchXWOM,PRAM_TEST_STACK/0FFFF8100,FIXEDVECT/0FFFFFFD0 -nologo -stack<br>-output="$(CONFIGDIR)¥$(PROJECTNAME).abs" -end -input="$(CONFIGDIR)¥$(PROJECTNAME).abs"<br>-form=stype -output="$(CONFIGDIR)¥$(PROJECTNAME).mot" -exit |

## 3.2　結果

### 3.2.1　CPU

**【注】　これらのテストでは最適化を使用することはできません。**

**表 11　CPU テスト結果**

| 測定 | 結果<br>非カップリングテスト | 結果<br>カップリングテスト |
|---|---|---|
| **コードサイズ** | 768 **バイト** | 3764 **バイト** |
| CPU_TestAll **のスタック使用量** | 24 **バイト** | 24 **バイト** |
| **関数** CPU_TestAll **の実行時間** | 290 **クロック** | 1281 **クロック** |
| | 3.02 uS | 13.35 uS |

### 3.2.2　ROM

**表 12　CRC16-CCITT のテスト結果**

| | | 最適化 | |
|---|---|---|---|
| 測定 | | サイズ | 速度 |
| **コードサイズ（バイト）** | | 90 | 518 |
| **スタック使用量（バイト）** | | 16 | 4 |
| **クロックサイクルカウント**（／1000） | 1KB | 7968 | 7872 |
| | 4KB | 31584 | 31488 |
| | 16KB | 125760 | 125760 |
| **測定時間**（ms） | 1KB | 0.08 | 0.08 |
| | 16KB | 0.33 | 0.33 |
| | 64KB | 1.31 | 1.31 |

### 3.2.3　RAM

テストは、8、16 および 32 ビットアクセス幅構成で実行されました。より小さなリミットを使用してもパフォーマンスが向上しないので、32 ビットワードリミットを常に使用しました。

'Extra'という名前は自動安全バッファテストを含む関数を指します。

### 3.2.4　マーチ C

表 13　マーチ C テスト結果（8 ビットアクセス、32 ビットワードリミット）

| | | | 最適化 | |
|---|---|---|---|---|
| 測定 | | | サイズ | 速度 |
| コードサイズ（バイト） | | | 365 | 1272 |
| スタック使用量（バイト） | | | 48 | 44 |
| スタック使用量 Extra（バイト） | | | 64 | 120 |
| クロックサイクルカウント | 破壊 | 1024 バイト | 677760 | 289920 |
| | | 500 バイト | 331200 | 141120 |
| | | 100 バイト | 66432 | 28032 |
| | 非破壊 | 1024 バイト | 693120 | 295680 |
| | | 500 バイト | 337920 | 144960 |
| | | 100 バイト | 67968 | 29184 |
| | Extra | 1024 バイト | 1370880 | 584640 |
| | | 500 バイト | 670080 | 286080 |
| | | 100 バイト | 134400 | 57216 |
| 測定時間（ms） | 破壊 | 1024 バイト | 7.06 | 3.02 |
| | | 500 バイト | 3.45 | 1.47 |
| | | 100 バイト | 692 | 292 |
| | 非破壊 | 1024 バイト | 7.22 | 3.08 |
| | | 500 バイト | 3.52 | 1.51 |
| | | 100 バイト | 708 | 304 |
| | Extra | 1024 バイト | 14.28 | 6.09 |
| | | 500 バイト | 6.98 | 2.98 |
| | | 100 バイト | 1.4 | 596 |

表 14　マーチ C テスト結果（16 ビットアクセス、32 ビットワードリミット）

| | | | 最適化 | |
|---|---|---|---|---|
| 測定 | | | サイズ | 速度 |
| コードサイズ（バイト） | | | 409 | 3921 |
| スタック使用量（バイト） | | | 48 | 72 |
| スタック使用量 Extra（バイト） | | | 64 | 144 |
| クロックサイクルカウント | 破壊 | 1024 バイト | 641280 | 278400 |
| | | 500 バイト | 312960 | 136320 |
| | | 100 バイト | 62976 | 27648 |
| | 非破壊 | 1024 バイト | 650880 | 281280 |
| | | 500 バイト | 317760 | 138240 |
| | | 100 バイト | 63360 | 28032 |
| | Extra | 1024 バイト | 1291200 | 560640 |
| | | 500 バイト | 630720 | 274560 |
| | | 100 バイト | 126720 | 55296 |
| 測定時間（ms） | 破壊 | 1024 バイト | 6.68 | 2.90 |
| | | 500 バイト | 3.26 | 1.42 |
| | | 100 バイト | 0.66 | 0.29 |
| | 非破壊 | 1024 バイト | 6.78 | 2.93 |
| | | 500 バイト | 3.32 | 1.44 |
| | | 100 バイト | 0.66 | 0.29 |
| | Extra | 1024 バイト | 13.45 | 5.84 |
| | | 500 バイト | 6.57 | 2.86 |
| | | 100 バイト | 1.32 | 0.58 |

表 15　マーチ C テスト結果（32 ビットアクセス、32 ビットワードリミット）

| | | | 最適化 | |
|---|---|---|---|---|
| 測定 | | | サイズ | 速度 |
| コードサイズ（バイト） | | | 408 | 4843 |
| スタック使用量（バイト） | | | 44 | 36 |
| スタック使用量 Extra（バイト） | | | 60 | 76 |
| クロックサイクルカウント | 破壊 | 1024 バイト | 611520 | 233280 |
| | | 500 バイト | 299520 | 114240 |
| | | 100 バイト | 59904 | 23040 |
| | 非破壊 | 1024 バイト | 616320 | 235200 |
| | | 500 バイト | 301440 | 115200 |
| | | 100 バイト | 60672 | 23424 |
| | Extra | 1024 バイト | 1227840 | 468480 |
| | | 500 バイト | 600000 | 229440 |
| | | 100 バイト | 120960 | 46464 |
| 測定時間（ms） | 破壊 | 1024 バイト | 6.37 | 2.43 |
| | | 500 バイト | 3.12 | 1.19 |
| | | 100 バイト | 0.624 | 0.240 |
| | 非破壊 | 1024 バイト | 6.42 | 2.45 |
| | | 500 バイト | 3.14 | 1.2 |
| | | 100 バイト | 0.632 | 0.244 |
| | Extra | 1024 バイト | 12.79 | 4.88 |
| | | 500 バイト | 6.25 | 2.39 |
| | | 100 バイト | 1.26 | 0.484 |

### 3.2.5　マーチ X WOM

表 16　マーチ X WOM テスト結果（8 ビットアクセス、32 ビットワードリミット）

| | | | 最適化 | |
|---|---|---|---|---|
| 測定 | | | サイズ | 速度 |
| コードサイズ（バイト） | | | 295 | 2085 |
| スタック使用量（バイト） | | | 32 | 20 |
| スタック使用量 Extra（バイト） | | | 48 | 44 |
| クロックサイクルカウント | 破壊 | 1024 バイト | 74880 | 59520 |
| | | 500 バイト | 36864 | 28800 |
| | | 100 バイト | 7680 | 5760 |
| | 非破壊 | 1024 バイト | 92544 | 65664 |
| | | 500 バイト | 45312 | 32640 |
| | | 100 バイト | 9216 | 6528 |
| | Extra | 1024 バイト | 167040 | 126720 |
| | | 500 バイト | 81792 | 62208 |
| | | 100 バイト | 16512 | 12672 |
| 測定時間（ms） | 破壊 | 1024 バイト | 0.78 | 0.62 |
| | | 500 バイト | 0.38 | 0.30 |
| | | 100 バイト | 0.08 | 0.06 |
| | 非破壊 | 1024 バイト | 0.96 | 0.68 |
| | | 500 バイト | 0.47 | 0.34 |
| | | 100 バイト | 0.10 | 0.07 |
| | Extra | 1024 バイト | 1.74 | 1.32 |
| | | 500 バイト | 0.85 | 0.65 |
| | | 100 バイト | 0.17 | 0.13 |

表 17　マーチ X WOM テスト結果（16 ビットアクセス、32 ビットワードリミット）

| | | | 最適化 | |
|---|---|---|---|---|
| 測定 | | | サイズ | 速度 |
| コードサイズ（バイト） | | | 347 | 2411 |
| スタック使用量（バイト） | | | 32 | 20 |
| スタック使用量 Extra（バイト） | | | 48 | 52 |
| クロックサイクルカウント | 破壊 | 1024 バイト | 38784 | 33024 |
| | | 500 バイト | 18816 | 16512 |
| | | 100 バイト | 3840 | 3456 |
| | 非破壊 | 1024 バイト | 48768 | 36480 |
| | | 500 バイト | 23808 | 18432 |
| | | 100 バイト | 4608 | 3840 |
| | Extra | 1024 バイト | 87936 | 69120 |
| | | 500 バイト | 43008 | 34560 |
| | | 100 バイト | 8832 | 7680 |
| 測定時間（ms） | 破壊 | 1024 バイト | 0.40 | 0.34 |
| | | 500 バイト | 0.20 | 0.17 |
| | | 100 バイト | 0.04 | 0.04 |
| | 非破壊 | 1024 バイト | 0.51 | 0.38 |
| | | 500 バイト | 0.25 | 0.19 |
| | | 100 バイト | 0.05 | 0.04 |
| | Extra | 1024 バイト | 0.92 | 0.72 |
| | | 500 バイト | 0.45 | 0.36 |
| | | 100 バイト | 0.09 | 0.08 |

表 18　マーチ X WOM テスト結果（32 ビットアクセス、32 ビットワードリミット）

| | | | 最適化 | |
|---|---|---|---|---|
| 測定 | | | サイズ | 速度 |
| コードサイズ（バイト） | | | 353 | 3627 |
| スタック使用量（バイト） | | | 32 | 20 |
| スタック使用量 Extra（バイト） | | | 48 | 36 |
| クロックサイクルカウント | 破壊 | 1024 バイト | 20064 | 16416 |
| | | 500 バイト | 9888 | 8160 |
| | | 100 バイト | 2112 | 1824 |
| | 非破壊 | 1024 バイト | 25056 | 18048 |
| | | 500 バイト | 12288 | 9312 |
| | | 100 バイト | 2592 | 2304 |
| | Extra | 1024 バイト | 45120 | 34560 |
| | | 500 バイト | 22176 | 17760 |
| | | 100 バイト | 4704 | 4320 |
| 測定時間（ms） | 破壊 | 1024 バイト | 0.22 | 0.17 |
| | | 500 バイト | 0.10 | 0.09 |
| | | 100 バイト | 0.02 | 0.02 |
| | 非破壊 | 1024 バイト | 0.26 | 0.19 |
| | | 500 バイト | 0.13 | 0.10 |
| | | 100 バイト | 0.03 | 0.02 |
| | Extra | 1024 バイト | 0.47 | 0.36 |
| | | 500 バイト | 0.23 | 0.19 |
| | | 100 バイト | 0.05 | 0.05 |

### 3.2.6　スタックテスト

**【注】 使用するアルゴリズムによって異なるので、これにはタイミング情報が含まれません。実際のメモリテストと比べてスタックを移動する時間は無視できるので、通常の RAM テスト結果を参照してください。**

**【注】 インラインアセンブリが使用されるので最適化に関係なく結果は同じです。**

| | 最適化 | |
|---|---|---|
| **測定** | **サイズ** | **速度** |
| **コードサイズ（バイト）プログラム** | 281 | 281 |
| **コードサイズ（バイト）**RAM | 36 | 36 |
| **スタック使用量（バイト）** | 12 | 12 |

### 3.2.7　ADC10

関数 Test_ADC10_Wait のスタック使用量と実行時間

| | 最適化 | |
|---|---|---|
| **測定** | **サイズ** | **速度** |
| ROM **サイズプログラム + データ** | 363 **バイト** | 393 **バイト** |
| RAM **サイズ（スタックを除く）** | 4 **バイト** | 4 **バイト** |
| **スタック使用量** | 16 **バイト** | 4 **バイト** |
| **実行時間：**uS | 16 uS | 16 uS |
| **実行時間：クロックサイクル** | 1536 | 1536 |

### 3.2.8　ADC12

関数 Test_ADC12_Wait のスタック使用量と実行時間

| | 最適化 | |
|---|---|---|
| **測定** | **サイズ** | **速度** |
| ROM **サイズプログラム + データ** | 318 **バイト** | 422 **バイト** |
| RAM **サイズ（スタックを除く）** | 0 **バイト** | 0 **バイト** |
| **スタック使用量** | 12 **バイト** | 4 **バイト** |
| **実行時間：**uS | 45 uS | 45 uS |
| **実行時間：クロックサイクル** | 4320 | 4320 |

### 3.2.9　電圧監視

このソフトウェアはペリフェラルを連続的に監視しているため、実行時間のベンチマークは、適用できません。

| | 最適化 | |
|---|---|---|
| **測定** | **サイズ** | **速度** |
| ROM **サイズプログラム + データ** | 229 | 96 |
| RAM **サイズ（スタックを除く）** | 0 | 0 |
| **スタック使用量** | 4 | 4 |

### 3.2.10　クロック監視

このソフトウェアはペリフェラルを連続的に監視しているため、実行時間のベンチマークは、適用できません。

| | 最適化 | |
|---|---|---|
| **測定** | **サイズ** | **速度** |
| ROM **サイズプログラム** + **データ** | 275 | 133 |
| RAM **サイズ（スタックを除く）** | 0 | 0 |
| **スタック使用量** | 4 | 4 |

### 3.2.11　ウォッチドッグ

このソフトウェアはペリフェラルを連続的に監視しているため、実行時間のベンチマークは、適用できません。

| | 最適化 | |
|---|---|---|
| **測定** | **サイズ** | **速度** |
| ROM **サイズプログラム** + **データ** | 147 | 78 |
| RAM **サイズ（スタックを除く）** | 0 | 0 |
| **スタック使用量** | 21 | 4 |

# 4. ユーティリティ

## 4.1 低消費電力制御

RX62T は、ハードウェアマニュアルから抜粋した図 2 に記載されている低消費電力モードをサポートしています。

表9.2　各モードにおける移行および解除方法と動作状態

| 移行および解除方法と動作状態 | スリープモード | 全モジュールクロックストップモード | ソフトウェアスタンバイモード | ディープソフトウェアスタンバイモード |
|---|---|---|---|---|
| 移行方法 | 制御レジスタ＋命令 | 制御レジスタ＋命令 | 制御レジスタ＋命令 | 制御レジスタ＋命令 |
| リセット以外の解除方法 | 割り込み | 割り込み（注1） | 割り込み（注2） | 割り込み（注3） |
| 解除後の状態（注4） | プログラム実行状態（割り込み処理） | プログラム実行状態（割り込み処理） | プログラム実行状態（割り込み処理） | プログラム実行状態（リセット処理） |
| 発振器 | 動作 | 動作 | 停止 | 停止 |
| CPU | 停止（保持） | 停止（保持） | 停止（保持） | 停止（不定） |
| 内蔵RAM（0000 0000h～0000 3FFFh） | 動作（保持） | 停止（保持） | 停止（保持） | 停止（不定） |
| ウォッチドッグタイマ（WDT） | 動作 | 動作 | 停止（保持） | 停止（不定） |
| 独立ウォッチドッグタイマ（IWDT） | 動作 | 動作 | 停止（保持） | 停止（不定） |
| 電圧検出回路 | 動作 | 動作 | 動作 | 動作 |
| パワーオンリセット回路 | 動作 | 動作 | 動作 | 動作 |
| 周辺モジュール | 動作 | 停止（注5） | 停止（注5） | 停止（不定） |
| I/O端子状態 | 動作 | 保持 | 保持 | 保持 |

**図 2　低消費電力モード**

電源ソフトウェアモジュールはこれらのすべてのモードの切り換えをサポートしていますが、IWDT を使用している場合は、ソフトウェアスタンバイとディープソフトウェアスタンバイモードにアクセスすることはできません。

表 19　ソースファイル

| ファイル名 |
|---|
| Power.h |
| Power.c |

| 構文 | |
|---|---|
| void Power_Init(void) | |
| 説明 | |
| 電源モジュールを初期化します。このモジュールのその他の関数を使用する前にこの関数を呼び出します。 | |
| 入力パラメータ | |
| なし | 該当せず |
| 出力パラメータ | |
| なし | 該当せず |
| 戻り値 | |
| なし | 該当せず |

| 構文 | |
|---|---|
| void Power_Set(etPOWER_MODE eMode) | |
| 説明 | |
| デバイスを指定された低消費電力モードに設定します。<br>【注】 WDT または IWDT を使用している場合は、ソフトウェアスタンドバイモードとディープソフトウェアスタンドバイモードにすることはできません。<br>【注】 IWDT を使用している場合は、スリープまたは全モジュールクロック停止モードになった後も定期的に起動されなければなりません。<br>"IWDT_KICK_IN_SLEEP_MODE"が定義された場合、このモジュールは WDT モジュールを使用してこれを自動的に実行し、IWDT を起動して、スリープに戻ることができるように定期的に復帰します。<br>この関数は BRK 割り込みを生成します。"IWDT_KICK_IN_SLEEP_MODE"関数を呼び出すハンドラは、低消費電力モードに実際に切り換えます。 | |
| 入力パラメータ | |
| eMode | 必要な低消費電力モード |
| 出力パラメータ | |
| なし | 該当せず |
| 戻り値 | |
| なし | 該当せず |

| 構文 | |
|---|---|
| bool_t Power_Did_DeepStandbyReset (void) | |
| 説明 | |
| リセット後、この関数を使用して、ディープソフトウェアスタンバイモードの終了によってリセットが発生したかどうかを知ることができます。<br>【注】 この関数はディープソフトウェアスタンバイモードを使用する場合にのみ適用可能です。 | |
| 入力パラメータ | |
| なし | 該当せず |
| 出力パラメータ | |
| なし | 該当せず |
| 戻り値 | |
| bool_t | ディープソフトウェアスタンバイモードが終了した場合、TRUE、それ以外の場合は、FALSE を返します。 |

| 構文 | |
|---|---|
| void Power_DeepStandby_IOResume (void) | |
| 説明 | |
| ディープソフトウェアスタンバイモードを終了するためにリセット後、LSI 内部状態にかかわらずこの関数が呼び出されるまで IO 端子の状態を保持してください。<br>【注】 この関数はディープソフトウェアスタンバイモードを使用する場合にのみ適用可能です。 | |
| 入力パラメータ | |
| なし | 該当せず |
| 出力パラメータ | |
| なし | 該当せず |
| 戻り値 | |
| なし | 該当せず |

| 構文 | |
|---|---|
| void Power_BRK_InterruptHandler (void) | |
| 説明 | |
| BRK 割り込みハンドラはこの関数を呼び出さなければなりません。このため、この関数はスーパーバイザモードで呼び出され、特権 WAIT 命令を実際に実行して、関数 Power_Set で要求される低消費電力モードに入ることができます。 | |
| 入力パラメータ | |
| なし | 該当せず |
| 出力パラメータ | |
| なし | 該当せず |
| 戻り値 | |
| なし | 該当せず |

# 5. 追加情報

## 5.1 IO 端子状態の読み出し

ハードウェアマニュアルから抜粋した図 3 に記載されているように、対応する端子の PORT レジスタから読み出すことにより IO 端子の実際値を常に読むことができます。

14.2.2.3　ポートレジスタ（PORT）

アドレス　PORT1.PORT 0008 C041h、PORT2.PORT 0008 C042h、PORT3.PORT 0008 C043h、PORT4.PORT 0008 C044h、PORT5.PORT 0008 C045h、PORT6.PORT 0008 C046h、PORT7.PORT 0008 C047h、PORT8.PORT 0008 C048h、PORT9.PORT 0008 C049h、PORTA.PORT 0008 C04Ah、PORTB.PORT 0008 C04Bh、PORTD.PORT 0008 C04Dh、PORTE.PORT 0008 C04Eh

| | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
|---|---|---|---|---|---|---|---|---|
| | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
| リセット後の値 | X | X | X | X | X | X | X | X |

注1.　PORT1.PORTは、下位2ビットが有効で、上位6ビットは予約ビットです。
PORT2.PORTは、下位5ビットが有効で、上位3ビットは予約ビットです。
PORT3.PORTは、下位4ビットが有効で、上位4ビットは予約ビットです。
PORT5.PORTは、下位6ビットが有効で、上位2ビットは予約ビットです。
PORT6.PORTは、下位6ビットが有効で、上位2ビットは予約ビットです。
PORT7.PORTは、下位7ビットが有効で、上位1ビットは予約ビットです。
PORT8.PORTは、下位3ビットが有効で、上位5ビットは予約ビットです。
PORT9.PORTは、下位7ビットが有効で、上位1ビットは予約ビットです。
PORTA.PORTは、下位6ビットが有効で、上位2ビットは予約ビットです。
PORTE.PORTは、下位6ビットが有効で、上位2ビットは予約ビットです。
注2.　予約ビットは、読むと"1"が読めます。書き込みは無効になります。

| ビット | シンボル | ビット名 | 機能 | R/W |
|---|---|---|---|---|
| b0 | B0 | Pn0 ビット | ポートの端子状態を反映 | R |
| b1 | B1 | Pn1 ビット | | R |
| b2 | B2 | Pn2 ビット | | R |
| b3 | B3 | Pn3 ビット | | R |
| b4 | B4 | Pn4 ビット | | R |
| b5 | B5 | Pn5 ビット | | R |
| b6 | B6 | Pn6 ビット | | R |
| b7 | B7 | Pn7 ビット | | R |

n = 1～9、A、B、D、E

PORT レジスタは、ポートの端子の状態を反映するレジスタです。
PORTn.PORT レジスタ（n = 1～9、A、B、D、E）を読むと、端子の状態が読めます。

**図 3　PORT レジスタ**

## 5.2 ポートアウトプットイネーブルモジュール

RX62T には POE モジュールがあります。このモジュールを使用することで、CPU の状態にかかわらず MTU（マルチファンクションタイマユニット）の PWM 出力端子および GPT（汎用 PWM タイマ）の大電流出力端子を強制的にハイインピーダンス状態にすることができます。これは、以下の条件で可能となります。

- POE 入力端子が、立ち下がりエッジまたは低レベルを検出する。
- クロックパルスジェネレータ内の発振停止検出回路が、発振の停止を検出する。

詳細については、RX62T ハードウェアマニュアルを参照してください。

## 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/inquiry

## 改訂記録

| Rev. | 発行日 | 改訂内容 | |
|---|---|---|---|
| | | ページ | ポイント |
| 1.00 | 2011.09.27 | — | 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