RX200 シリーズ

R01AN0950JG0102
Rev.1.02
2013.02.20

VDE 認証 RX200 シリーズ MCU の IEC60730 セルフテストコード

はじめに

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気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 AnnexH 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

たとえば、洗濯機、食器洗い機、乾燥機、冷蔵庫、冷凍庫、および調理器などの用途があります。ストーブはクラス B に分類される傾向があります。

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 VDETestand CertificationInstituteGmbH 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。

注 1　本書は欧州規格 EN60335-1:2002/A1:2004AnnexR に基づいています。その中では規格 IEC60730-1（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 AnnexR には、定義、情報および該当するパラグラフについて IEC60730-1 にジャンプする 1 枚のシートが含まれています。

対象デバイス

RX200 シリーズ MCU

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。IEC60730:1999+A1:2003AnnexH-表 H.11.12.1CPU を参照してください。

次の CPU レジスタがテストされます。R0～R15、ISP、USP、INTB、PC、PSW、BPC、BPSW、FINTV 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。

注: **ファイル** CPU_Test.c **の次のステートメントは、他の** RXCPU **コアでサポートされ** RX210 **ではサポートされていないレジスタ（**FPSW**）のテストを行わないようにするために必要となります。**

```
#defineRX210
```

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。

```
externvoidCPU_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ているように 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。

1. カップリング GPR テスト（*1、下記参照）を使用する場合:
   CPU_Test_GPRsCouplingPartA
   CPU_Test_GPRsCouplingPartB

カップリング GPR テストを使用しない場合:

CPU_Test_GeneralA

CPU_Test_GeneralB

2. CPU_Test_Control（*2、下記参照）
3. CPU_Test_Accumulator
4. CPU_Test_PC

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。

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。

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。

詳細については、個別のテストを参照してください。

*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。

*2. RX210 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、RX210 を使用する場合は、プロジェクトに “RX210” 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。 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。 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。 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。 これは全体の GPR テストのパート B の部分で、テストを完了するには関数 CPU_Test_GPRsCouplingPartA を使用します。 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。 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。 各ペアのレジスタに対して、以下のように行います。 1. 両方のレジスタに h'55555555 を書き込みます。 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。 3. 両方のレジスタに h'AAAAAAAA を書き込みます。 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。 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。 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。 各ペアのレジスタに対して、以下のように行います。 1. 両方のレジスタに h'55555555 を書き込みます。 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。 3. 両方のレジスタに h'AAAAAAAA を書き込みます。 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。 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。 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。

注: 'RX210' が #define 定義されている場合、FPSW はテストされません。

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。

一般に各レジスタのテスト手順は以下のとおりです。

各レジスタに対して、以下のように行います。

1. h'55555555 を書き込みます。
2. 値を読み出して、h'55555555 であることを確認します。
3. h'AAAAAAAA を書き込みます。
4. 値を読み出して、h'AAAAAAAA であることを確認します。

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。

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。

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。

RX210 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、RX210 を使用する場合は、プロジェクトに "RX210" を定義してください。

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。 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。 レジスタ値はこのテストで保存されます。 テスト手順は以下のとおりです。 1. 上位 32 ビットに h'55555555 を書き込みます。 2. 下位 32 ビットに h'55555555 を書き込みます。 3. 上位値を読み出し、h'55555555 であることを確認します。 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。 5. 上位 32 ビットに h'AAAAAAAA を書き込みます。 6. 下位 32 ビットに h'AAAAAAAA を書き込みます。 7. 上位値を読み出し、h'AAAAAAAA であることを確認します。 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。 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。 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。 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。 関数を呼び出すことで PC をテストします。 これは関数呼び出し時、その関数の実行用に 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。 この関数が確実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。この戻り値が正しいかどうかがチェックされます。 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。IEC60730:1999+A1:2003AnnexH– H2.19.4.1CRC– 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。そのため、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

RX200 ファミリには、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 ANSIC で記述されています。

'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 値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 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示すポインタ
`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示すポインタ
`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

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ときに使用されます。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準備をします。関数 CRC_AddRange を使用する前に、これを 1 回呼び出します。	
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_Data`	テストするメモリ領域の先頭を示すポインタ
`uint32_t ui32_Length`	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。

以下の節では、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（vandeGoor1991）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できない。

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. アレイから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** RX200 **ファミリには実装されていないので、下記のマーチ** XWOM **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ています。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 番目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始まり、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

RAM テストに関しては、2 つの実装が用意されています。

1) 標準の実装。
2) ハードウェア（HW）実装。このバージョンではデータ演算回路（DOC）およびテストの実行を補助するためのオプションとして 1 個の DMAC チャネルを使用します。

いずれの実装でも、同じコア API を共有しますが、HW 実装ではいくつかの関数が追加されます。詳細は「(3)マーチ C およびマーチ XWOM の HW 実装に固有の API」を参照してください。

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。このため、一度に h'0F バイトを超える範囲のテストを実行することをお勧めします。**

(1) マーチ CAPI

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することができます。

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:

標準	HW
ramtest_march_c.h	ramtest_march_c.h
ramtest_march_c.c	ramtest_march_c_HW.c
	ramtest_march_HW.h
	ramtest_march_HW.c

ソースファイルは ANSI.C で書かれており、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。

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 の最終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。 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 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。‘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 XWOMAPI

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することができます。

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

テストの実行速度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って実現します。

- RAMTEST_MARCH_X_WOM_MAX_WORDS_8BIT　（テスト領域の最大ワードは 0xFF です。）
- RAMTEST_MARCH_X_WOM_MAX_WORDS_16BIT　（テスト領域の最大ワードは 0xFFFF です。）
- RAMTEST_MARCH_X_WOM_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:

標準	HW
ramtest_march_x_wom.h	ramtest_march_x_wom.h
ramtest_march_x_wom.c	ramtest_march_x_wom_HW.c
	ramtest_march_HW.h
	ramtest_march_HW.c

ソースファイルは ANSI.C で書かれており、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。 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_X_WOM_**Extra**(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。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) マーチ C およびマーチ XWOM の HW 実装に固有の API

マーチ C およびマーチ XWOM テストの HW（ハードウェア）実装ではデータ演算回路（DOC）およびオプションとしてテストの実行を補助する DMAC チャネルを使用します。DMAC はテスト対象の RAM を初期化し、DOC は RAM から読み出した値と期待される値との比較に使われます。

RAM テスト中、DOC または選択された DMAC のチャネルにアクセスがないようにしてください。

オプションの DMAC の使用はファイル ramtest_march_HW.h 内の以下の#define によって指定されます。

#define	**定義時の意味**
RAMTEST_USE_DMAC	DMAC **が初期化されます。**
DMAC_CHANNEL	**使用する** DMAC **のチャネルを選択します。詳細はファイルを参照してください。**

'RAMTEST_USE_DMAC' が定義されたときには、特定の HW テストで DMAC を使うことができます。これは以下の定義で実現します。

#define	**定義されているファイル**
RAMTEST_MARCH_C_USE_DMAC	ramtest-march_c_HW.c
RAMTEST_MARCH_X_WOM_USE_DMAC	ramtest-march_x_wom_HW.c

宣言	
`void RamTest_March_HW_Init(void);`	
説明	
RAM テストの HW 実装で使用されるハードウェア(DOC およびオプションの DMAC)を初期化します。 DMAC は RAMTEST_USE_DMAC が定義されているときにのみ使用されます。 HW 実装を使用する他の RAM テスト関数を使用する前にこの関数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

宣言	
`bool_t RamTest_March_HW_PreTest(void);`	
説明	
これは、使用の前に、ハードウェア（DOC と DMAC）が正しく動作しているかをチェックするために使用されます。 DOC と、RAMTEST_USE_DMAC が定義されている場合には DMAC の、簡単な機能テストを行います。	
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。

宣言	
`bool_t RamTest_March_HW_Is_Init(void);`	
説明	
RamTest_March_HW_Init がすでに呼び出されたか否かをチェックします。 これは特定の RAM テストで、使用に先だってハードウェアが初期化されているかをチェックするために使用されます。 ユーザはこの関数を使用する必要はありません。	
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	
`void RamTest_March_HW_Wait_DMAC(void);`	
説明	
DAMC チャネルのデータ転送の完了を待ちます。 これは特定の RAM テストで使用されているもので、ユーザがこれを呼び出す必要はありません。 注: 理論的には、ユーザがこの待ちループの中にコードを追加することは可能です。しかし、これは RAM テスト中に呼び出されるため、実行中の RAM テストに影響する RAM を使用しないように十分な注意を払わねばなりません。 注: RAMTEST_USE_DMAC が定義されているときにのみ使用できます。	
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

(4) 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。

注: **スタックのテスト関数は、関数ポインタとして指定されたマーチ RAM テストのいずれかを使用します。使用前に初期化が必要なテストを使用する場合は、テスト使用前に以下のいずれかの関数を呼び出して初期化されていることを確認してください。**

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,`
`uint32_t ui32_EndAddr,`
`void* p_RAMSafe,`
`uint32_t ui32_NewUSP,`
`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 の関数のポインタ Typedef　bool_t (*TEST_FUNC)(uint32_t, uint32_t, void*); 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。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, uint32_t ui32_EndAddr, void* p_RAMSafe, uint32_t ui32_NewISP, 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_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	使用され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 Typedef bool_t (*TEST_FUNC)(uint32_t, uint32_t, void*); 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。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_Stacks(uint32_t ui32_StartAddr, uint32_t ui32_EndAddr, void* p_RAMSafe, uint32_t ui32_NewISP, uint32_t ui32_NewUSP, TEST_FUNC fpTest_Func);	
説明	
ユーザスタックと割り込みスタックの両方を含む領域の 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_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 Typedef　bool_t (*TEST_FUNC)(uint32_t, uint32_t, void*); 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。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

RX200 ファミリは実行時にメインクロックの周波数を監視するために使用できる、クロック周波数精度測定回路（CAC）を搭載しています。

IWDTCLK または CACREF 端子入力の外部クロックを基準として使用することができます。

外部基準クロックを使用する場合:

1. ファイル clock_monitor.c 内で
   CLOCK_MONITOR_USE_EXTERNAL_REFERENCE_CLOCK を定義。

IWDCLK を使用する場合:

1. CLOCK_MONITOR_USE_EXTERNAL_REFERENCE_CLOCK が定義されていないことを確認。
2. CLOCK_COUNT_EXPECTED が期待されるメインクロックの値を正しく定義していることを確認。

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逸脱したときには、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。

```
/*Percentagetoleranceofmainclockallowedbeforeanerrorisreported.*/
#defineCLOCK_TOLERANCE_PERCENT10
```

CAC 関数だけでなく、RX200 ファミリは発振停止検出回路を備えています。メインクロックが停止した場合、その代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。

表 6　ソースファイル:

ファイル名
clock_monitor.h
clock_monitor.c

ClockMonitor_Init 関数には、2 つのバージョンが用意されています。

1. CLOCK_MONITOR_USE_EXTERNAL_REFERENCE_CLOCK が定義されていない場合の ClockMonitor_Init 関数。

構文	
`void ClockMonitor_Init (CLOCK_MONITOR_ERROR_CALL_BACK CallBack)`	
説明	
1. CAC モジュールおよび基準クロックとして IWDCLK を使用してメインクロックの監視を開始します。 2. 発振停止検出を有効にし、検出時に生成される NMI の構成を行います。	
入力パラメータ	
`CallBack`	メインクロックが許容範囲を超え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

2. CLOCK_MONITOR_USE_EXTERNAL_REFERENCE_CLOCK が定義されている場合の ClockMonitor_Init 関数。

構文	

`void ClockMonitor_Init (uint32_t MainClockFrequency,`
`uint32_t ExternalRefClockFrequency,`
`CLOCK_MONITOR_CACREF_PIN ePin,`
`CLOCK_MONITOR_ERROR_CALL_BACK CallBack)`	

説明	
1. CAC モジュールおよび基準クロックとして CACREF 端子からの外部クロックを使用してメインクロックの監視を開始します。 2. 発振停止検出を有効にし、検出時に生成される NMI の構成を行います。	
入力パラメータ	
`MainClockFrequency`	メインクロックの期待される周波数（Hz）
`ExternalRefClockFrequency`	外部基準クロックの周波数（Hz）
`ePin`	CACREF に使用される端子。詳細は CLOCK_MONITOR_CACREF_PIN を参照してください。
`CallBack`	メインクロックが許容範囲を超えた場合またはこの関数が失敗し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

1.5 独立ウォッチドッグ

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。プログラムが期待通りに動作していない場合は、必要に応じてウォッチドック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 RX200 ファミリ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。IWDT は、指定した時間の前にリフレッシュを行うのではなく指定した「ウィンドウ（時間間隔）」内にリフレッシュを行うようにするリフレッシュ許可期間の設定機能を持っています。エラーの検出時に内部のリセットまたは NMI 割り込みを生成するように構成することができます。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, IWDT_WINDOW_START WindowStart, IWDT_WINDOW_END WindowEnd, IWDT_ACTION Action)	
説明	
独立ウォッチドッグタイマを初期化し起動します。これを呼び出した後、ウォッチドッグのエラーを防止するために IWDT_kick 関数を適切な時間に呼び出さなければなければなりません。 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 NMISR.IWDTST フラグをチェックする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	
TimeOutperiod	タイムアウトとなるカウント数。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。
ClockSelection	IWDT クロックの選択。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。
WindowStart	ウィンドウの開始位置。詳細については、IWDT.h 内の列挙型 IWDT_WINDOW_START の宣言を参照してください。
WindowEnd	ウィンドウの終了位置。詳細については、IWDT.h 内の列挙型 IWDT_WINDOW_END の宣言を参照してください。
Action	エラー検出時にリセットを行うか、NMI を生成するかを選択します。
出力パラメータ	
なし	該当せず
戻り値	
なし	該当せず

構文	
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。	
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またはリフレッシュが正しく行われなかった場合に TRUE が戻されます。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。

構文	
`void IWDT_SleepMode_CountStop_Disable (void)`	
説明	
デフォルトでは、IWDT カウンタはスリープモードで停止します。スリープモードでもカウントを続けるようにデフォルト設定を変えるには、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

1.6　電圧

RX200 ファミリ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）が指定した電圧より低くなったことを検出することができます。提供されているコード例は電圧検出回路 1 を使用し、Vcc が指定したレベルより低くなったときに NMI 割り込みを発生する方法を示しています。この回路はリセットを発生させたり、外部端子からの電圧を監視する機能も備えていますが、これらはコード例ではサポートされていません。

表 8　ソースファイル:

ファイル名
Voltage.h
Voltage.c

構文	
`void VoltageMonitor_Init(VOLTAGE_MONITOR_LEVEL eVoltage)`	
説明	
電圧監視を初期化し起動します。Vcc が指定した電圧より低下すると NMI が発生します。 注: ノンマスカブル割り込み（NMI）は NMISR.LVDST フラグのチェックを行う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	
`VOLTAGE_MONITOR_LEVEL eVoltage`	電圧低下レベルの指定。詳細については、voltage.h 内の列挙型 VOLTAGE_MONITOR_LEVEL 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

1.7　ADC12

ADC12 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。診断の基準電圧および期待される結果がゼロ、フルスケールの 1/2、およびフルスケールの間で自動的に切り替わります。電源投入テストでは、ゼロとフルスケールの 2 つの変換が自動的に開始される診断モードが使用されます。

表 9　ソースファイル:

ファイル名
test_adc12.h
test_adc12.c

構文	
`void Test_ADC12_Init(void)`	
説明	
ADC12 モジュールを初期化します。他の ADC 関数の呼び出し前にこの関数を呼び出さなければなりません。	
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_ADC12_Wait(void)`	
説明	
この関数は 2 つの ADC 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_ADC12_Start(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。診断の基準電圧が自動的に切り替わります。（ゼロ、1/2 VREF および VREF) 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_ERROR_CALL_BACK Callback`	エラーが検出された場合に呼び出される関数 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(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。 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。 注: 実際の結果は期待される結果に対して一定の許容幅を持っています。詳細については、test.ad12.c 内の ADC12_TOLERANCE を参照してください。	
入力パラメータ	
`bool_t bCallErrorHandler`	関数 Test_ADC12_Start に提供される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。

1.8 温度

RX210 は MCU の温度を監視することができる温度センサモジュールを搭載しています。温度センサを使用するときには、ADC12 モジュールも必要となります。

表 10　**ソースファイル**:

ファイル名
Temperature.h
Temperature.c

構文	
void Temperature_Init(TEMPERATURE_GAIN_SELECT eGain, uint16_t Temperature_ADC_Value_Min, uint16_t Temperature_ADC_Value_Max, TEMPERATURE_ERROR_CALL_BACK Error_callback)	
説明	
温度センサを初期化し、ADC12 モジュールを有効にします。許容される温度範囲を ADC12 の出力値で指定します。この関数の呼び出し後、Temperature_Start 関数を定期的に呼び出し、温度センサ出力の AD 変換を実行します。次にその他の関数を使用してこの結果をチェックします。	
入力パラメータ	
eGain	温度センサのゲインを指定します。これにより必要な分解能の AD 変換値が得られます。詳細については、Temparature.h 内の列挙型 TEMPERATURE_GAIN_SELECT の宣言を参照してください。
Temperature_ADC_Value_Min	温度センサの読み取り時に ADC12 が出力する最小値を指定します。
Temperature_ADC_Value_Max	温度センサの読み取り時に ADC12 が出力する最大値を指定します。 注: ADC の出力は 12 ビットですので、h'FFF を超える値は指定しないでください。
Error_callback	温度（ADC12 値）が指定された許容範囲を逸脱したとき、この関数が関数 Temperature_CheckResult により呼び出されます。
出力パラメータ	
なし	該当せず
戻り値	
なし	該当せず

構文	
void Temperature_Start(void);	
説明	
温度を読み取る AD 変換を開始します。この際、ADC12 モジュールのそれまでの設定は保存されません。この動作で問題ないことを確認してください。 この関数の後に、関数 Temperature_Read_Wait または Temperature_CheckResult を使用してください。	
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 Temperature_Wait_Finish (void);`	
説明	
この関数は、Temperature_Start で開始された温度変換が完了するまで待ちます。	
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	
`uint16_t Temperature_Read_Wait (void);`	
説明	
この関数は、Temperature_Start で開始された温度変換が完了するまで待ち、ADC12 値を戻します。	
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`	ADC12 の出力値

構文	
`bool_t Temperature_CheckResult(bool_t bCallErrorHandler)`	
説明	
この関数は、Temperature_Start で開始された温度変換が完了するまで待ち、ADC12 の値が Temperature_Init 内で指定された範囲にあるかをチェックします。	
入力パラメータ	
`bCallErrorHandler`	温度が指定範囲を逸脱した場合に呼び出される Temperature_Init で設定された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`	結果が指定範囲内であれば TRUE を、指定範囲外の時には FALSE を戻します。

1.9 ポートアウトプットイネーブル（POE）

ポートアウトプットイネーブル 2（POE2）モジュールは MTU2A による相補 PWM 出力に使用される端子（MTIOC3B、MTIOC3D、MTIOC4A、MTIOC4B、MTIOC4C および MTIOC4D）の状態と MTU0 の端子（MTIOC0A、MTIOC0B、MTIOC0C および MTIOC0D）の状態を、いくつかの条件でハイインピーダンスとするために使用されます。条件となるのは、POE0#～POE3#および POE8#の端子の入力レベルの変化、MTU2A による相補 PWM 出力端子の出力レベルの変化、クロック生成回路による発振停止の検出、レジスタ設定（SPOER）の変更、およびイベントリンクコントローラ（ELC）からのイベント信号の変化です。

このソフトウェアは POE0 入力端子で立ち下がりエッジが検出されたとき、または発振停止が検出されたときに、特定の端子の設定をハイインピーダンス状態にします。

表 11　ソースファイル:

ファイル名
POE.h
POE.c

構文	
`void POE_Init(POE_CALL_BACK Callback);`	
説明	
このソフトウェアは POE を次のように構成します。 1. POE0（PD_7 端子）入力端子で立ち下がりエッジが検出されたとき、MTIOC3B、MTIOC3D、MTIOC4A、MTIOC4B、MTIOC4C および MTIOC4D 端子をハイインピーダンス状態にします。同時に、割り込みを発生させます。 2. 発振停止が検出されたとき、MTIOC0A、MTIOC0B、MTIOC0C、MTIOC0D、MTIOC3B、MTIOC3D、MTIOC4A、MTIOC4B、MTIOC4C および MTIOC4D 端子をハイインピーダンス状態にします。	
入力パラメータ	
`POE_CALL_BACK Callback`	POE0（PD_7 端子）入力端子で立ち下がりエッジが検出されたときに呼び出される関数
出力パラメータ	
なし	該当せず

構文	
`void POE_ClearFlags(void);`	
説明	
発振停止ハイインピーダンスステータスフラグをクリアします。 POE0～POE3 の検出ステータスフラグをクリアします。 この関数はハイインピーダンス状態とした端子の状態を元に戻します。	
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

2. 使用例

実際の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ています。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。

このサンプルでは、3.3V で動作する（Board_VCC=3.3V）ように設定された RSKRX210 上で実行されるようにテストを構成しています。E1 エミュレータを介して RSK に電源が供給されている場合、接続時に 3.3V オプションを選択します。外部の電源（5V）を使用する場合には、RSK のジャンパをデフォルトの J15=オープン、J17=端子 2 と 3 をショート、に設定してください。

ワークスペースのビルド時には、次のコンパイラ警告が表示されることがあります。

"C5170(W)Pointerpointsoutsideofunderlyingobject（ポインタが指定されたオブジェクト以外を指しています。）"

この警告メッセージはセクションアドレスオペレータ_sectop および_secend を使用したときに、コンパイラによって表示されることがあります。コードの動作は確認されており、この警告を無視しても問題はありません。

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るべきですが、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。注:電源を切る頻度の少ないアプリケーションを構築するときには、電源投入時以外でもこれらのテストを実行するようにスケジュールすることが必要な場合もあります。
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を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するかは、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 RX210 のタイマモジュールをセットアップして、関数（`Periodic TestCallBack`）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、RX210 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ができません。

以下の節では、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 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: resetprg.c **内の** Change_PSW_PM_to_UserMode **関数によってデバイスがユーザモードに入る前に、関数が呼び出され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

ROM は、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。

注:HEW の abs ファイルには基準 CRC は含まれません。必要に応じて、abs ファイルに加えて mot ファイルをダウンロードする必要があります。

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。

結果を一致させるためには、使用するすべての ROM セクションが HEW と CRC テストコードの両方で使用する CRC 計算に含まれていることを確認します。

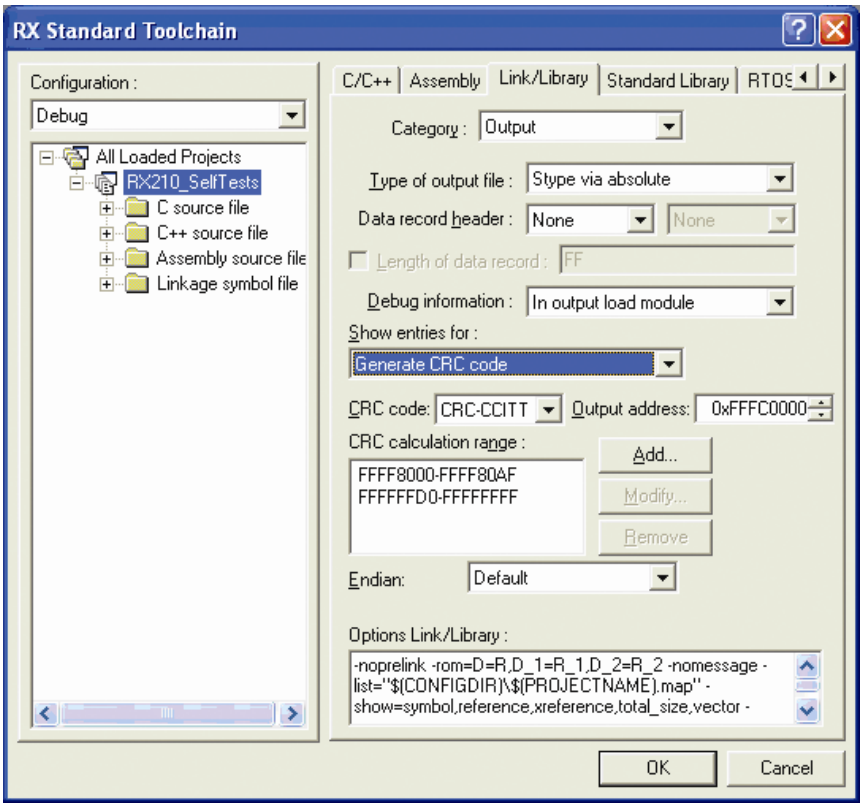

図 1　基準 CRC の追加

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 値を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 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。

```
constCRC_RANGECRC_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 Test_Usage_RAM.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点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条件があります。ビルドのためにセットアップするときに Test_Usage_RAM.c のを参照して、コメントを慎重に読んでください。

HW バージョンの RAM テスト（DOC および場合により DMAC を使用）を使用する場合、テスト実行前に関数 RamTest_March_HW_Init を呼び出してください。これは Test_Usage_RAM.c ファイル内の以下の#define で指示されます。

```
#defineUSE_HW_VERSION_OF_RAM_TESTS
```

RAM テストをする際には以下の点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 バッファが必要です。
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 電源投入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。関数はデバイスが resetprg.c 内の関数 Change_PSW_PM_to_UserMode の呼び出しでユーザモードに入る前に呼び出され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. STACK_START_ADDRESS から STACK_END_ADDRESS までに定義されたスタック領域には非破壊テストが実施されます。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 XWOM 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にしています。

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。この関数には外部の基準クロックを使用するか、内部の基準クロックを使用するかによって 2 つのバージョンがあり、次の#define で選択されます。

```
#defineCLOCK_MONITOR_USE_EXTERNAL_REFERENCE_CLOCK
```

たとえば、

```
#ifdefCLOCK_MONITOR_USE_EXTERNAL_REFERENCE_CLOCK
   ClockMonitor_Init(CLOCK_FREQ_MAIN,125000,
               eCLOCK_MONITOR_CACREF_PIN_PA0,
               Clock_Test_Failure);
#else
/*NOTE:TheIWDTCLKclockmustbeenabledbeforestartingtheclockmonitoring.IWDT_Initd
oesthisanditiscalledattheendoftheTests_PowerOn*/
   ClockMonitor_Init(Clock_Test_Failure);
#endif
```

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 独立ウォッチドッグ」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();

   /*ClearOSTSTbitbywriting1toNMICLR.OSTCLRbit*/
   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 独立ウォッチドッグ

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。

```
/*SetuptheIndependentWDT.
IWDT_Init(IWDT_TOP_16384,IWDT_CKS_DIV_16,
            IWDT_WINDOW_START_NO_START,IWDT_WINDOW_END_NO_END,
            IWDT_ACTION_NMI);
```

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。リフレッシュ可能期間のウィンドウが設定されているときには、リフレッシュは定期的に行うだけでなく指定したウィンドウにあわせた時間に行う必要であることに注意してください。ウォッチドッグリフレッシュは以下の関数呼び出しで行われます。

```
/*Regularlykickthewatchdogtopreventitperformingareset.*/
IWDT_Kick();
```

ウォッチドッグがエラー検出時に NMI を発生するように構成されているときには、ユーザはこれによる割り込みを処理しなければなりません。

ウォッチドッグがエラー検出時にリセットを行うように構成されているときには、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:Handleawatchdogreset.
   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。これは電源投入時のリセッ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、電圧が 4.15V より低くなった場合に NMI を発生するように電圧監視をセットアップします。

```
VoltageMonitor_Init(VOLTAGE_MONITOR_LEVEL_4_15);
```

電圧低下が検出されると NMI 割り込みが発生します。ユーザはこれを以下のように処理しなければなりません。

```
/*LowVoltageLVD1*/
if(1==ICU.NMISR.BIT.LVD1ST)
{
 Voltage_Test_Failure();

 /*ClearLVD1STbitbywriting1toNMICLR.LVD1CLRbit*/
 ICU.NMICLR.BIT.LVD1CLR=1;
}
```

2.7 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。

#defineADC12_TOLERANCE8

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。

ADC12 テストモジュールは Test_ADC12_Init への関数呼び出しで初期化されます。

2.7.1 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。基準電圧に 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。この関数の戻り値は結果を得るためにチェックしなければなりません。

2.7.2 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。

2.8 温度

MCU の温度をテストする場合、ADC12 モジュールが使用されることに注意してください。したがって、ユーザコードでアナログ端子の監視に ADC12 が使われている場合は、ADC12 モジュールのリソースの共有に関して慎重に検討されなければなりません。

温度センサは Temperature_Init 関数の呼び出しで使用前に初期化してください。この関数は温度の許容範囲を ADC12 の出力として指示します。具体的な値の算出に関しては RX210 ハードウェアマニュアルを参照してください。

```
/*TemperatureSensor*/
Temperature_Init(TEMPERATURE_GAIN_SELECT_AVCC0_2_7_VOLTS,
            TEMPERATURE_ADC_MIN,
            TEMPERATURE_ADC_MAX,
            Temperature_Test_Failure);
```

サンプルコードでは室温でエラーを検出せずにコードが実行されるように、センサ出力の変動を考慮して許容温度範囲は広く設定されています。しかし、エラーが検出される場合には、TEMPERATURE_ADC_MIN および TEMPERATURE_ADC_MAX の#define を調整することで、システムや環境に合うように許容範囲を調節してください。

2.8.1 電源投入テスト

サンプルソフトウェアでは、電源投入時に温度のテストは行いません。しかし、必要があれば、定期テストで説明されているものと同じ手順を使用することができます。

2.8.2 定期テスト

温度センサに対して ADC12 モジュールが定期的に使用されます。温度を読み取るためには、以下の関数を呼び出します。

```
/*StartADCreadingtemperaturesensoroutput.*/
Temperature_Start();
```

結果は、以下の関数に対する呼び出しにより、Temperature_Init 関数で提供された許容範囲の値に照らしてチェックされます。

```
/*TheregisteredErrorcallbackwillbecalledifthereisanerror.*/
Temperature_CheckResult(TRUE);
```

定期テストが長時間ブロックすることを防ぐには、定期テストをスケジュールするたびに、前回スケジュールしたテストで開始された温度テストの結果をチェックし、次に新たな変換を開始するようにできます。詳細については、関数 PeriodicTest_Temperature にある使用例を参照してください。

ユーザコードでは関数 Temperature_Is_Finished または Temperature_Wait_Finish により、アナログ端子を読み出すために ADC12 を使用していつ動作を開始できるかを判断できます。

2.9　ポートアウトプットイネーブル（POE）

POE は次の関数呼び出しで初期化され、動作を開始します。

```
POE_Init(POE_Event_Detected);
```

ご使用時には POE_Init 関数の説明と RX210 ハードウェアマニュアルの記述をあわせてお読みになり、POE のサンプル構成がユーザシステムの要件に合っているかを検討してください。ユーザシステムで使用される実際の端子にあわせて希望の動作を行うように POE.C ファイルの変更が必要な場合がありです。

POE のサンプル構成では、POE0（PD_7）端子の立ち下がりエッジで POE がトリガされるようになっています。サンプルテストをデフォルト状態の RSKRX210 上で実行する際に接続されていない端子によってトリガが発生しないように、POE_Init を呼び出す前にこの端子は PORTD.PCR.BIT.B7=1 の設定でハイ（H）レベルに内部でプルアップされています。

3. ベンチマーク

3.1 環境

開発ボード: RSKRX210

クロック: EXTAL=20MHz、ICLK=50MHz、PCLKB=25MHZ、PCLKD=50MHZ

MCU: R5F52108

ツールチェイン: RX 標準ツールチェイン 1.1.0.0

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=rx200 - include="$(PROJDIR)","$(PROJDIR)¥Tests¥Common","$(PROJDIR)¥Tests¥IWDT","$(PROJDIR)¥Tests¥RAM","$(PROJDIR)¥Tests¥ROM","$(PROJDIR)¥Tests¥CPU","$(PROJDIR)¥Tests¥ADC12","$(PROJDIR)¥Tests¥Clock","$(PROJDIR)¥Tests¥Voltage","$(PROJDIR)¥Tests¥Temperature","$(PROJDIR)¥Tests¥POE" -change_message=warning -output=obj="$(CONFIGDIR)¥$(FILELEAF).obj" -debug -optimize=max -map="$(CONFIGDIR)¥$(PROJECTNAME).bls" -nologo
最大レベル スピードの最適化	-cpu=rx200 - include="$(PROJDIR)","$(PROJDIR)¥Tests¥Common","$(PROJDIR)¥Tests¥IWDT","$(PROJDIR)¥Tests¥RAM","$(PROJDIR)¥Tests¥ROM","$(PROJDIR)¥Tests¥CPU","$(PROJDIR)¥Tests¥ADC12","$(PROJDIR)¥Tests¥Clock","$(PROJDIR)¥Tests¥Voltage","$(PROJDIR)¥Tests¥Temperature","$(PROJDIR)¥Tests¥POE" -change_message=warning -output=obj="$(CONFIGDIR)¥$(FILELEAF).obj" -debug -optimize=max -speed -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 -sdebug -rom=D=R,D_1=R_1,D_2=R_2 -nomessage -list="$(CONFIGDIR)¥$(PROJECTNAME).map" -optimize=speed -start=B_1,R_1,B_2,R_2,B,R,SU,SI,BRAM_TEST_STACK,BTEMPERATURE_TEST_2/01000,PResetPRG/0FFFF8000,C_1,C_2,C,C$*,D_1,D_2,D,P,PIntPRG,W*,L,PPCTEST_TESTFUNCTION,PADC12_TEST,PCLOCK_MONITOR_TEST,DADC12_TEST,DCLOCK_MONITOR_TEST,PCPU_TEST,PIWDT_TEST,PRAM_TEST_MarchC,PRAM_TEST_MarchXWOM,PRAM_TEST_MarchC_HW,PRAM_TEST_MarchXWOM_HW,PRAM_TEST_STACK,PVOLTAGE_TEST,PCRC,PTEMPERATURE_TEST,DTEMPERATURE_TEST,PRAM_TEST_HW,PPOE_TEST,DPOE_TEST/0FFFF8100,FIXEDVECT/0FFFFFFD0 -nologo -output="$(CONFIGDIR)¥$(PROJECTNAME).abs" -end -input="$(CONFIGDIR)¥$(PROJECTNAME).abs" -form=stype -output="$(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 12　CPU **テスト結果**

測定	**結果 非カップリングテスト**	**結果 カップリングテスト**
コードサイズ.	710 **バイト**	3706 **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**	205	1405
	4.10uS	28.10uS

3.2.2 ROM

表 13　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）		102	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	1k バイト	9000	4500
	4k バイト	37000	18000
	16k バイト	147500	69500
測定時間（ms）	1k バイト	0.18	0.09
	4k バイト	0.74	0.36
	16k バイト	2.95	1.39

3.2.3 RAM

テストは､8 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 14　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）

			通常版		HW 版（DOC+DMAC）	
			最適化			
測定			サイズ	速度	サイズ	速度
コードサイズ（バイト）			359	2588	655	3586
スタック使用量（バイト）			48	112	68	44
スタック使用量 Extra（バイト）			64	264	52	92
クロックサイクルカウント	破壊	1024 バイト	706500	296500	662500	304500
		500 バイト	345000	145000	323500	149000
		100 バイト	69000	29000	65000	29500
	非破壊	1024 バイト	723000	304000	675000	317000
		500 バイト	353000	148500	329500	155000
		100 バイト	70500	30000	66000	31000
	Extra	1024 バイト	1430000	606000	1337500	617500
		500 バイト	698000	296000	653500	301500
		100 バイト	140000	59500	131000	60500
測定時間（ms）	破壊	1024 バイト	14.13	5.93	13.25	6.09
		500 バイト	6.90	2.90	6.47	2.98
		100 バイト	1.38	0.58	1.30	0.59
	非破壊	1024 バイト	14.46	6.08	13.5	6.34
		500 バイト	7.06	2.97	6.59	3.10
		100 バイト	1.41	0.60	1.32	0.62
	Extra	1024 バイト	28.6	12.12	26.75	12.35
		500 バイト	13.96	5.92	13.07	6.03
		100 バイト	2.8	1.19	2.62	1.21

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）

			通常版		HW 版（DOC+DMAC）	
			最適化			
測定			サイズ	速度	測定	サイズ
コードサイズ（バイト）			409	2884	713	4975
スタック使用量（バイト）			48	64	52	64
スタック使用量 Extra（バイト）			64	124	68	136
クロックサイクルカウント	破壊	1024 バイト	669000	335500	587000	375500
		500 バイト	326500	164000	286500	183500
		100 バイト	65500	32500	57500	36500
	非破壊	1024 バイト	680000	339500	590000	379000
		500 バイト	332000	166000	288000	185000
		100 バイト	66500	33000	58000	37000
	Extra	1024 バイト	1349000	677000	1177500	766500
		500 バイト	659000	330500	575000	374000
		100 バイト	669000	66500	115500	75000
測定時間（ms）	破壊	1024 バイト	13.38	6.71	11.74	7.51
		500 バイト	6.53	3.28	5.73	3.67
		100 バイト	1.31	0.65	1.15	0.73
	非破壊	1024 バイト	13.60	6.79	11.80	7.58
		500 バイト	6.64	3.32	5.76	3.70
		100 バイト	1.33	0.66	1.16	0.74
	Extra	1024 バイト	26.98	13.54	23.55	15.33
		500 バイト	13.18	6.61	11.50	7.48
		100 バイト	2.64	1.33	2.31	1.50

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）

通常版					HW 版（DOC+DMAC）	
最適化						
測定			サイズ	速度	測定	サイズ
コードサイズ（バイト）			288	2754	579	2163
スタック使用量（バイト）			28	20	56	40
スタック使用量 Extra（バイト）			44	44	40	84
クロックサイクルカウント	破壊	1024 バイト	79000	47000	55500	34000
		500 バイト	38500	23000	27000	16500
		100 バイト	7500	48000	5500	36000
	非破壊	1024 バイト	95500	54500	67500	46000
		500 バイト	46500	27000	33000	22500
		100 バイト	9500	5500	6500	48000
	Extra	1024 バイト	174000	102000	123000	81000
		500 バイト	85000	50000	60500	39500
		100 バイト	17000	10500	12000	8000
測定時間（ms）	破壊	1024 バイト	1.58	0.94	1.11	0.68
		500 バイト	0.77	0.46	0.54	0.33
		100 バイト	0.15	0.96	0.11	0.72
	非破壊	1024 バイト	1.91	1.09	1.35	0.92
		500 バイト	0.93	0.54	0.66	0.45
		100 バイト	0.19	0.11	0.13	0.96
	Extra	1024 バイト	3.48	2.04	2.46	1.62
		500 バイト	1.70	1.00	1.21	0.79
		100 バイト	0.34	0.21	0.24	0.16

表 17　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）

通常版					HW 版（DOC+DMAC）	
最適化						
測定			サイズ	速度	測定	サイズ
コードサイズ（バイト）			346	2754	639	3045
スタック使用量（バイト）			28	28	36	40
スタック使用量 Extra（バイト）			44	64	52	80
クロックサイクルカウント	破壊	1024 バイト	42000	25000	17000	13500
		500 バイト	20500	12000	8000	6500
		100 バイト	4000	2500	2000	1500
	非破壊	1024 バイト	52000	29000	20000	16500
		500 バイト	25500	14500	10000	8000
		100 バイト	5000	3000	2000	1500
	Extra	1024 バイト	94500	54500	37000	30000
		500 バイト	46000	27000	18000	15000
		100 バイト	9000	5500	4000	3000
測定時間（ms）	破壊	1024 バイト	0.84	0.50	0.34	0.27
		500 バイト	0.41	0.24	0.16	0.13
		100 バイト	0.08	0.05	0.04	0.03
	非破壊	1024 バイト	1.04	0.58	0.40	0.33
		500 バイト	0.51	0.29	0.20	0.16
		100 バイト	0.10	0.06	0.04	0.03
	Extra	1024 バイト	1.89	1.09	0.74	0.60
		500 バイト	0.92	0.54	0.36	0.30
		100 バイト	0.18	0.11	0.08	0.06

3.2.6　スタックテスト

注:　使用するアルゴリズムにより異なるので、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	389	389
コードサイズ（バイト）RAM	36	36
スタック使用量（バイト）	12	12

4. 補足情報

4.1 IO 端子状態の読み出し

ハードウェアマニュアルからの以下の抜粋に記載されているように、対応する端子のポート入力データレジスタを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。

19.3.3 ポート入力データレジスタ（PIDR）

アドレス：PORT0.PIDR 0008 C040h,PORT1.PIDR 0008 C041h,PORT2.PIDR 0008 C042h,PORT3.PIDR 0008 C043h,
PORT4.PIDR 0008 C044h,PORT5.PIDR 0008 C045h,PORTA.PIDR 0008 C04Ah,PORTB.PIDR 0008 C04Bh,
PORTC.PIDR 0008 C04Ch,PORTD.PIDR 0008 C04Dh,PORTE.PIDR 0008 C04Eh,PORT1.PIDR 0008 C051h,
PORTJ.PIDR 0008 C052h

	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

x: **不定**

ビット	シンボル	ビット名	機能	R/W
b0	B0	Pm0ビット	**ポートの端子状態を反映**	R
b1	B1	PM1ビット		R
b2	B2	Pm2ビット		R
b3	B3	Pm3ビット		R
b4	B4	Pm4ビット		R
b5	B5	Pm5ビット		R
b6	B6	Pm6ビット		R
b7	B7	Pm7ビット		R

m = 0～5, A ～E, H, J

PIDR **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。**
PORTm.PIDR **レジスタを読むと、**PORTm.PDR **レジスタ、**PORTm.PMR**の値に関係なく端子の状態が読めます。**
P35 **は**NMI **端子の状態が読み出されます。**
存在しない端子のビットは予約ビットです。予約ビットは、読んだ場合、その値は不定です。書き込みは無効になります。

図 2　PIDR **レジスタ**

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	2012.01.26	—	暫定版
1.01	2012.03.14	1.1.1	ソフトウェア API–関数名 TestPCReg を CPU_Test_PC に変更
1.02	2012.03.29	2	RSKRX210 上でデフォルト 3.3V で動作するようサンプルを構成
1.02	2012.03.29	2.8	テストの際の許容温度範囲の調整方法
1.02	2013.02.20	2.9	RSKRX210 上でのテストの使用に関する POE 関連の説明

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、本文を参照してください。なお、本マニュアルの本文と異なる記載がある場合は、本文の記載が優先するものとします。

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・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/contact/