R01AN4823JJ0120
Rev.1.20
May.20.24

RL78シリーズ

RL78 MCUのためのIEC60730/60335セルフテスト・ライブラリ CARL78拡張版

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、および調理器/レンジなどの家電製品は、一般的にクラス B に分類されています。

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 IEC60730/60335 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、システムの電源投入後、また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。本書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】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

RL78/G14 マイクロコントローラ

目次

1. セルフテスト・ライブラリの概要

セルフテスト・ライブラリ（STL）は、CPU レジスタ、内部メモリ、およびシステム・クロックを対象とするセルフテスト関数で構成されます。以降で説明するように、テスト・ハーネスにはセルフテストを行う各モジュールのアプリケーション・プログラム・インタフェース（API）が用意されています。各関数は用途に応じて使用します。

セルフテスト・ライブラリ関数は、VDE 認定に準じてモジュール別に分かれています。CS+テスト・ハーネスでは、各テスト関数を順番に選択してスタンドアロンで実行することができます。

システムのハードウェア要件は、2 つ以上の独立したクロック・ソース（水晶/セラミック・オシレータと独立動作のオシレータまたは外部入力ソースなど）を利用できることです。これは、システムクロックを監視する別のクロック基準を設定するために必要となります。RL78 は、相互に独立して動作する高速と低速の内部オシレータを使用しており、この要件を満たします。

アプリケーション側でより高精度の外部基準クロックを用意したり、メイン・システム・クロック用として外付けの水晶/レゾネータを使用したりすることも可能です。

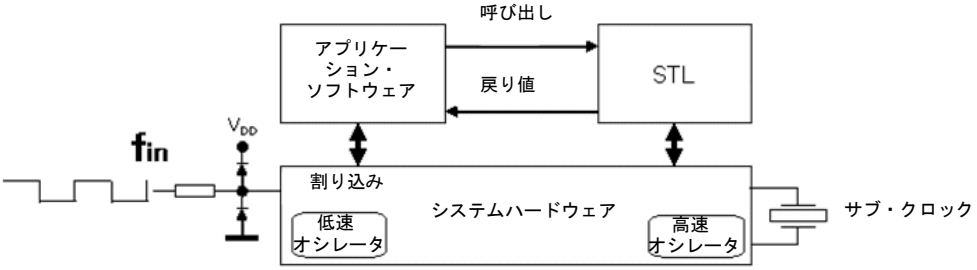


図 1　セルフテスト・ライブラリ（STL）の構成

RL78 のセルフテスト・ライブラリには以下の CPU セルフテスト関数があります。

- CPU レジスタ

  以下の CPU レジスタをテストします。

  4 つの全レジスタ・バンク内の全 CPU ワーク・レジスタ、スタック・ポインタ（SP）、プロセッサ・ステータス・ワード（PSW）、拡張レジスタ（ES および CS）、プログラムカウンタ(PC)。

  内部データ・パスは、以上のレジスタの正常動作テストの中で検証します。

  IEC 60730: 1999+A1:2003 Annex H - Table H.11.12.1 CPU.を参照してください。

- 不変メモリ

  MCU の内部 Flash メモリをテストします。

  IEC 60730: 1999+A1:2003 Annex H - H2.19.4.1 CRC - Single Word を参照してください。

- 可変メモリ

  内部 SRAM をテストします。

  IEC 60730: 1999+A1:2003 Annex H - H2.19.4.1 CRC - Single Word を参照してください。

- システム・クロック

  基準クロック・ソースを元にしてシステム・クロックの動作および周波数をテストします（このテストには内部または外部の独立した基準クロックが必要です）。

- IEC 60730: 1999+A1:2003 Annex H - H2.19.4.1 CRC - Single Word を参照してください。

2.　セルフテスト・ライブラリ関数

2.1　CPU レジスタ・テスト

本章では、CPU レジスタ・テストの各ルーチンについて説明します。テスト・ハーネスの制御ファイル‘main.c’には、各 CPU レジスタ・テストの C 言語で記述された API サンプルが用意されています。

これらのモジュールは CPU の基本的な動作をテストします。各 API 関数は、戻り値によりテスト結果を通知します。

各テスト・モジュールは、テストの開始時にレジスタの内容を保存し、完了時に復元します。

テストを行う CPU レジスタは以下の通りです。

- ワーク・レジスタおよびアキュムレータ：レジスタ・バンク 0～3 の AX、HL、DE、BC

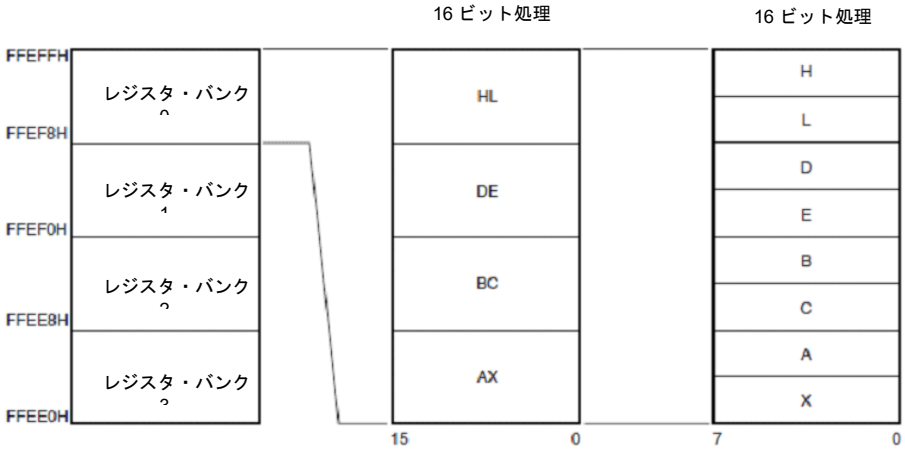


図 2　ワーク・レジスタの構成

- スタック・ポインタ（SP）

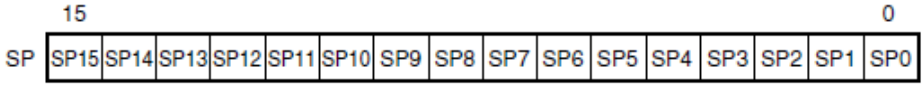


図 3　スタック・ポインタの構成

- プロセッサ・ステータス・ワード（PSW）

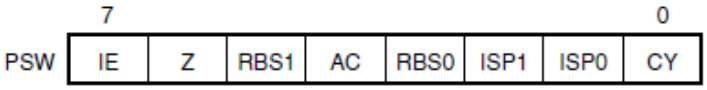


図 4　PSW レジスタの構成

- コード・アドレス拡張レジスタ（CS）

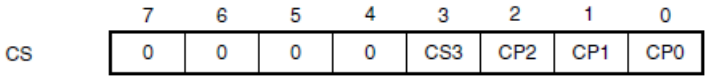


図 5　コード・アドレス拡張レジスタの構成

- データ・アドレス拡張レジスタ（ES）

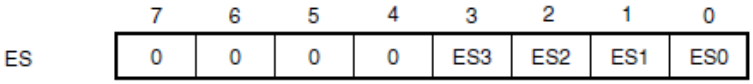


図 6　データ・アドレス拡張レジスタの構成

- プログラムカウンタ(PC)

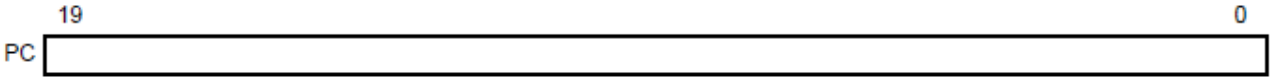


図 7　プログラムカウンタの構成

2.1.1　CPU レジスタ・テスト - ソフトウェア API

表 1　ソース・ファイル：CPU ワーク・レジスタ・テスト

STLファイル名	ヘッダ・ファイル
stl_RL78_registertest.asm	なし
テスト・ハーネス・ファイル名	ヘッダ・ファイル
main.c stl_global_data_example.c stl_main_example_support function.c stl_peripheralinit.c	stl.h main.h stl_gobal_data_example.h

構文	
char stl_RL78_registertest(void)	
説明	
RL78 のワーク・レジスタとアキュムレータをテストします。 4 つの全レジスタ・バンク（バンク 0、1、2、3）のレジスタ AX、HL、DE、BC 16 ビット・レジスタとしてレジスタをテストします。 各レジスタで以下のテストを実行します。 1. レジスタに h'5555 を書き込みます。 2. レジスタを読み出し、書き込み値と等しいことを確認します。 3. レジスタに h'AAAA を書き込みます。 4. レジスタを読み出し、書き込み値と等しいことを確認します。 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。また、このテストは必ずレジスタ・バンク 0 が選択された状態で開始して下さい。 テスト前のレジスタ内容はテスト完了後に復元されます。 テスト・ハーネスの制御ファイル（main.c）は、関数“indicate_test_result”を呼び出してテスト結果を処理します。 【注】関数“indicate_test_result“はモジュールstl_main_example_support function.c内にあります。	
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	
char	CPU レジスタ C のテスト結果 0 = テストはパスしました。 1 = 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。

表 2　ソース・ファイル：CPU レジスタ・テストーPSW

STLファイル名	ヘッダ・ファイル
stl_RL78_registertest_psw.asm	stl.h
テスト・ハーネス・ファイル名	ヘッダ・ファイル
main.c stl_global_data_example.c stl_main_example_support function.c stl_peripheralinit.c	main.h stl_gobal_data_example.h

構文	
char stl_RL78_registertest_psw(void)	
説明	
8 ビットのプロセッサ・ステータス・ワード（PSW）レジスタをテストします。 以下のテストを実行します。 1. レジスタに h'55 を書き込みます。 2. レジスタを読み出し、書き込み値と等しいことを確認します。 3. レジスタに h'AA を書き込みます。 4. レジスタを読み出し、書き込み値と等しいことを確認します。 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。 テスト前のレジスタ内容はテスト完了後に復元されます。 テスト・ハーネスの制御ファイル（main.c）は、関数“indicate_test_result”を呼び出してテスト結果を処理します。 【注】関数“indicate_test_result”はモジュールstl_main_example_support function.c内にあります。	
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	
char	CPU レジスタ C のテスト結果 0 = テストはパスしました。 1 = 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。

表 3　ソース・ファイル：CPU レジスタ・テストーSP

STLファイル名	ヘッダ・ファイル
stl_RL78_registertest_stack.asm	stl.h
テスト・ハーネス・ファイル名	ヘッダ・ファイル
main.c stl_global_data_example.c stl_main_example_support function.c stl_peripheralinit.c	main.h stl_gobal_data_example.h

構文	
char stl_RL78_registertest_stack(void)	
説明	
16 ビットのスタック・ポインタ（SP）レジスタをテストします。 以下のテストを実行します。 1. レジスタに h'5555 を書き込みます。 2. レジスタを読み出し、h'5554 と等しいことを確認します。 3. レジスタに h'AAAA を書き込みます。 4. レジスタを読み出し、書き込み値と等しいことを確認します。 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。 テスト前のレジスタ内容はテスト完了後に復元されます。 テスト・ハーネスの制御ファイル（main.c）は、関数“indicate_test_result”を呼び出してテスト結果を処理します。 【注】関数“indicate_test_result”はモジュールstl_main_example_support function.c内にあります。	
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	
char	CPU レジスタ C のテスト結果 0 = テストはパスしました。 1 = 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。

表 4　ソース・ファイル：CPU レジスタ・テストーCS

STLファイル名	ヘッダ・ファイル
stl_RL78_registertest_cs.asm	stl.h
テスト・ハーネス・ファイル名	ヘッダ・ファイル
main.c stl_global_data_example.c stl_main_example_support function.c stl_peripheralinit.c	main.h stl_gobal_data_example.h

構文	
char stl_RL78_registertest_cs(void)	
説明	
8 ビットのコード拡張（CS）レジスタをテストします。 以下のテストを実行します。 1. レジスタに h'05 を書き込みます。 2. レジスタを読み出し、書き込み値と等しいことを確認します。 3. レジスタに h'0A を書き込みます。 4. レジスタを読み出し、書き込み値と等しいことを確認します。 先頭 4 ビットは“0”の固定値です。 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。 テスト前のレジスタ内容はテスト完了後に復元されます。 テスト・ハーネスの制御ファイル（main.c）は、関数“indicate_test_result”を呼び出してテスト結果を処理します。 【注】関数“indicate_test_result”はモジュールstl_main_example_support function.c内にあります。	
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	
char	CPU レジスタ C のテスト結果 0 = テストはパスしました。 1 = 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。

表 5　ソース・ファイル：CPU レジスタ・テストーES

STLファイル名	ヘッダ・ファイル
stl_RL78_registertest_es.asm	stl.h
テスト・ハーネス・ファイル名	ヘッダ・ファイル
main.c stl_global_data_example.c stl_main_example_support function.c stl_peripheralinit.c	main.h stl_gobal_data_example.h

構文	
char stl_RL78_registertest_es(void)	
説明	
8 ビットのデータ拡張（ES）レジスタをテストします。 以下のテストを実行します。 1. レジスタに h'05 を書き込みます。 2. レジスタを読み出し、書き込み値と等しいことを確認します。 3. レジスタに h'0A を書き込みます。 4. レジスタを読み出し、書き込み値と等しいことを確認します。 先頭 4 ビットは“0”の固定値です。 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。 テスト前のレジスタ内容はテスト完了後に復元されます。 テスト・ハーネスの制御ファイル（main.c）は、関数“indicate_test_result”を呼び出してテスト結果を処理します。 【注】関数“indicate_test_result”はモジュールstl_main_example_support function.c内にあります。	
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	
char	CPU レジスタ C のテスト結果 0 = テストはパスしました。 1 = 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。

表 6　ソース・ファイル：CPU レジスタ・テストーPC

STLファイル名	ヘッダ・ファイル
stl_RL78_registertest_pc.asm	stl.h
テスト・ハーネス・ファイル名	ヘッダ・ファイル
main.c	main.h

構文	
char stl_RL78_registertest_pc(void)	
説明	
プログラムカウンタ（PC）レジスタをテストします。 以下のテストを実行します。 1. プログラムカウンタ（PC）テスト用関数を call 命令で呼び出します。 2. テスト用関数は、スタックに保存された戻り番地をL レジスタ・DE レジスタに設定し、復帰します。 3. テスト用関数を call 命令で呼び出し後、call 命令の次に配置された命令のアドレス(PC)と戻り値(L-DE)が等しいことを確認します。 L レジスタの先頭 4 ビットは“0”の固定値です。 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。 テスト前のレジスタ内容はテスト完了後に復元されます。 テスト・ハーネスの制御ファイル（main.c）は、関数“indicate_test_result”を呼び出してテスト結果を処理します。 【注】関数“indicate_test_result”はモジュールstl_main_example_support function.c内にあります。	
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	
char	CPU レジスタ C のテスト結果 0 = テストはパスしました。 1 = 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。

2.2　不変メモリ・テスト – Flash ROM

本章では、CRC ルーチンによる Flash メモリ・テストについて説明します。CRC は、メモリ内容を表す単一ワードまたはチェックサムを生成する不具合/エラー制御手法です。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 進除算の剰余です。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、よく使用されます。

Flash ROM をテストする場合は、ROM の内容に対する CRC 値を生成して保存しておきます。メモリのセルフテスト時に、同じ CRC アルゴリズムを使用して新たに CRC 値を生成します。この CRC 値と保存した CRC 値とを比較します。この方法は、すべての 1 ビット・エラーを認識し、複数ビット・エラーを高い確率で認識します。

CRC を使用する場合の複雑な点は、あらかじめ CRC 値を生成しておいて、別の CRC ジェネレータで生成された別の CRC 値と比較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 値にはさまざまな要素が影響するので、この処理は容易ではありません。実際には、アルゴリズムにデータを提示する順序、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、および実際の CRC 値で要求されるビット順などが相互に関連します。ハードウェアとソフトウェアのセルフテスト関数はいずれも繰り返して実行することができるため、目的のアプリケーションの動作に応じて全体の CRC 値を計算したり、部分的な CRC 値を計算したりすることが可能です。全体の CRC の計算（または複数回計算の 1 回目）では、初期値として h'0000 を使用します。複数回計算では、前回の結果を次の計算の初期値に使用します。

ハードウェア・モジュールは、RL78 に搭載された汎用 CRC 機能です。ハードウェア・モジュールの場合は、基本的に同じ CRC アルゴリズムを使用して異なるデータ形式で CRC 値を計算します。テスト・ハーネスには、参照用として互換性を備えた CRC 計算ルーチンが用意されています。

2.2.1　CRC 16-CCITT アルゴリズム

RL78 の CRC モジュールは CRC16-CCITT に対応します。このソフトウェアで CRC モジュールを駆動すると、以下の 16 ビットの CRC16-CCITT が生成されます。

ソフトウェア・アルゴリズム

- CCITT 16多項式 = 0x1021 ($x^{16} + x^{12} + x^5 + 1$)
- 入力データ幅 = 8ビット
- 入力データ = ビット反転なし
- 初期値 = 0x0000または前回の部分CRC計算の16ビット結果
- 計算結果 = 16ビット（ビット反転なし）

ハードウェア・アルゴリズム

- CCITT 16多項式 = 0x1021 ($x^{16} + x^{12} + x^5 + 1$)
- 入力データ幅 = 8ビット
- 入力データ = ビット反転あり
- 初期値 = 0x0000または前回の部分CRC計算の16ビット結果
- 計算結果 = 16ビット（ビット反転あり）

2.2.2　ソフトウェア CRCーソフトウェア API

以降に示す関数は、CRC 値を計算して Flash ROM に格納された値と照合します。

表 7　ソース・ファイル：ソフトウェア CRC

STLファイル名	ヘッダ・ファイル
stl_RL78_sw_crc.asm	stl.h
テスト・ハーネス・ファイル名	ヘッダ・ファイル
main.c stl_global_data_example.c stl_main_example_support function.c stl_peripheralinit.c	main.h stl_gobal_data_example.h

構文	
unsigned short stl_RL78_sw_crc_asm (unsigned short crc, CHECKSUM_CRC_TEST_AREA *p);	
説明	
ソフトウェアの CRC 計算モジュールを使用して、指定されるアドレス範囲の CRC 値を計算します。開始アドレスと計算範囲（長さ）は、下表に示す構造体により呼び出し元関数から渡されます。部分計算または全体計算の結果が返されるので、必要に応じて基準 CRC 値と照合することができます。 テスト・ハーネスの制御ファイル（main.c）は、関数“indicate_test_result”を呼び出してテスト結果を処理します。 【注】関数“indicate_test_result”はモジュール“stl_main_example_support function.c”内にあります。	
入力パラメータ	
unsigned short　crc	CRC 計算の初期値
CHECKSUM_CRC_TEST_AREA *p	開始アドレスと計算範囲を格納する構造体へのポインタ
出力パラメータ	
なし	該当せず
戻り値	
unsigned short	16 ビットの CRC 値（全体計算または部分計算の結果） CPU レジスタ BC

ソース・ファイル：ソフトウェア CRC パラメータ構造体

CRC 関数の計算パラメータを提供する以下の構造体は、stl.h と main.c のファイルに実装されています。

構文	
static CHECKSUM_CRC_TEST_AREA checksum_crc;	
説明	
main.c の呼び出し元関数からでソフトウェア CRC モジュール（stl_RL78_sw_crc.asm）に渡されるパラメータを提供する構造体宣言とインスタンス。	
入力パラメータ	
unsigned long length;	テストするメモリ範囲（長さ = バイト数）
unsigned long start_address	CRC 計算の開始アドレス
出力パラメータ	
なし	該当せず
戻り値	
なし	該当せず

2.2.3　ハードウェア CRC－ソフトウェア API

表 8　ソース・ファイル：ハードウェア CRC 計算

STLファイル名	ヘッダ・ファイル
stl_RL78_peripheral_crc.asm	stl.h
テスト・ハーネス・ファイル名	ヘッダ・ファイル
main.c stl_global_data_example.c stl_main_example_support function.c stl_peripheralinit.c	main.h stl_gobal_data_example.h

構文	
unsigned short stl_RL78_peripheral_crc(unsigned short gcrc, CHECKSUM_CRC_TEST_AREA *p)	
説明	
ハードウェアの CRC ペリフェラル（汎用 CRC）を使用して、指定されるアドレス範囲の CRC 値を計算します。開始アドレスと計算範囲（長さ）は、下表に示す構造体により呼び出し元関数から渡されます。返される結果は、指定パラメータに応じて部分計算または全体計算の値です。 テスト・ハーネスの制御ファイル（main.c）は、関数“indicate_test_result.c”を呼び出してテスト結果を処理します。 【注】関数“indicate_test_result”はモジュール“stl_main_example_support function.c”内にあります。	
入力パラメータ	
unsigned short gcrc	CRC 計算の初期値
CHECKSUM_CRC_TEST_AREA *p	開始アドレスと計算範囲を格納する構造体へのポインタ
出力パラメータ	
なし	該当せず
戻り値	
unsigned short	16 ビットの CRC 値（全体計算または部分計算の結果） CPU レジスタ BC

ソース・ファイル：ハードウェア CRC パラメータ構造体

構文	
static CHECKSUM_CRC_TEST_AREA checksum_crc;	
説明	
main.c の呼び出し元関数からハードウェア CRC モジュール（stl_RL78_peripheral_crc.asm）に渡されるパラメータを提供する構造体宣言とインスタンス。 【注】ソフトウェア CRC 関数の構造体と同じです。	
入力パラメータ	
unsigned int length;	テストするメモリ範囲（長さ = バイト数）
unsigned int start_address	CRC 計算の開始アドレス
出力パラメータ	
なし	該当せず
戻り値	
なし	該当せず

2.3　可変メモリ - SRAM

マーチ・テストは、RAM の効果的なテスト方法として広く認識されているテスト体系です。

マーチ・テストは限定された一連のマーチ・エレメントで構成され、マーチ・エレメントはメモリ・アレイのセル単位で実行される限定された一連の処理で構成されます。一般に、アルゴリズムのエレメントを増やせば不具合の検出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あり、現状の RAM 値は保存されません。そのためアプリケーション・システムの初期化後または動作中にテストを行う場合は、RAM 内容を保存する必要があります。マーチ C およびマーチ X のテスト・モジュールは RAM 領域を部分的にテストするため、テスト時にデータを保存するための大容量の一時領域を確保する必要はありません。追加されたテスト・モジュール（“stl_RL78_march_c_initial”および“stl_RL78_march_x_initial”）はシステムを初期化する前に実行するようになっており、メイン・アプリケーションを起動する前にメモリ領域全体をテストすることができます。

テストする RAM 領域は、テスト中に他の目的に使用することはできません。このため、スタックとして使用する RAM のテストは特に困難です。この領域は、アプリケーションの C スタックを初期化する前か、アプリケーション処理が終了した後でのみテストが可能です。

次の章では、各マーチ・テストについて説明します。

2.3.1　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、全体で 10 種類の処理を実行する 6 つのマーチ・エレメントで構成されます。以下の不具合を検出します。

縮退不具合（SAF）
- 単独セルまたは連続セルの論理値が常に 0 または 1 の場合。

遷移不具合（TF）
- 単独セルまたは連続セルが 0→1 または 1→0 に遷移しない場合。

カップリング不具合（CF）
- 1 つのセルに書き込むと次のセルの内容が変化する場合。

アドレス・デコーダ不具合（AF）
- アドレス・デコードに影響する不具合。
- 特定のアドレスの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。
- 特定のアドレスで複数のセルが同時にアクセスされる。
- 特定のセルが複数のアドレスからアクセスされる。

通常のマーチ C アルゴリズムは以下の 6 つのマーチ・エレメントを使用します。

1. アレイにすべて0を書き込みます（<>(w0)）。
2. 最下位アドレスから開始し、0を読み出し、1を書き込んで、アレイをビットごとにインクリメントします。（>(r0,w1)）。
3. 最下位アドレスから開始し、1を読み出し、0を書き込んで、アレイをビットごとにインクリメントします（>(r1,w0)）。
4. 最上位アドレスから開始し、0を読み出し、1を書き込んで、アレイをビッごとにデクリメントします。（<(r0,w1)）。
5. 最上位アドレスから開始し、1を読み出し、0を書き込んで、アレイをビットごとにデクリメントします（<(r1,w0)）。
6. アレイからすべて0を読み出します（<>(r0)）。

（2）マーチ X

マーチ X アルゴリズムは単純構造で高速ですが、マーチ・エレメントが 4 つで全体の処理が 4 種類であるために詳細なテストには適しません。

- 縮退不具合（SAF）
- 遷移不具合（TF）
- 反転カップリング不具合（Cfin）
- アドレス・デコーダ不具合（AF）

以下の 4 つのマーチ・エレメントを使用します。

1. アレイにすべて0を書き込みます（<>(w0)）。
2. 最下位アドレスから開始し、0を読み出し、1を書き込んで、アレイをビットごとにインクリメントします（>(r0,w1)）。
3. 最上位アドレスから開始し、1を読み出し、0を書き込んで、アレイをビッごとにデクリメントします。（<(r1,w0)）。
4. アレイからすべて0を読み出します（<>(r0)）。

2.3.2　可変メモリ・テスト – ソフトウェア API

2.3.2.1　システム・マーチ C

システム・マーチ C テストはアプリケーション・システムの初期化後に実行します。テスト・ハーネスからの通常の関数呼び出しで実行するために C スタック・リソースを使用します。RAM 領域の一部または全部のテストが可能ですが、破壊的であるためにテストする領域をバッファに退避してください。このため、1 回の実行で RAM の全領域をテストすることは推奨できません。また、このテスト自体によって、スタック領域として使用している RAM 領域が破壊されることのないように注意して下さい。

このテストは 8 ビットで RAM にアクセスするように設定されており、バイト単位のテストを行うことができます。ただし、すべての不具合を検出するには 2 バイト以上のデータ範囲をテストする必要があります。

表 9　ソース・ファイル：システム・マーチ C

STLファイル名	ヘッダ・ファイル
stl_RL78_march_c.asm	stl.h
テスト・ハーネス・ファイル名	ヘッダ・ファイル
main.c stl_global_data_example.c stl_main_example_support function.c stl_peripheralinit.c	main.h stl_gobal_data_example.h

宣言	
char stl_RL78_march_c(unsigned char __far *addr, unsigned short num)	
説明	
マーチ C アルゴリズムを使用して、呼び出し元関数から指定される RAM のアドレス範囲をテストし、その結果（パス/フェイル）を返します。このモジュールは、アプリケーション・システムの初期化後に実行します。 テスト・ハーネスの制御ファイル（main.c）は、関数“indicate_test_result”を呼び出してテスト結果を処理します。 【注】関数“indicate_test_result”はモジュール stl_main_example_support function.c 内にあります。	
入力パラメータ	
unsigned char __far *addr	テストする RAM の先頭アドレスのポインタ。
unsigned short num	テストする RAM 範囲（バイト数）
出力パラメータ	
なし	該当せず
戻り値	
char	CPU レジスタ C のテスト結果 0 = テストはパスしました。 1 = 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。

2.3.2.2　システム・マーチ X

システム・マーチ X セルフテスト関数は、アルゴリズムが簡略化されている点を除けば基本的にはマーチ C モジュールと同じです。ただし、アプリケーション・システムの初期化後に実行するように設計されているため、1 回の実行で全メモリ領域をテストすることは推奨できません。また、このテスト自体によって、スタック領域として使用している RAM 領域が破壊されることのないように注意して下さい。

このテストは 8 ビットで RAM にアクセスするように設定されており、バイト単位のテストを行うことができます。ただし、すべての不具合を検出するには 2 バイト以上のデータ範囲をテストする必要があります。

表 10　ソース・ファイル：システム・マーチ X

STLファイル名	ヘッダ・ファイル
stl_RL78_march_x.asm	stl.h
テスト・ハーネス・ファイル名	ヘッダ・ファイル
main.c stl_global_data_example.c stl_main_example_support function.c stl_peripheralinit.c	main.h stl_gobal_data_example.h

宣言	
char stl_RL78_march_x(unsigned char __far *addr, unsigned short num)	
説明	
マーチ X アルゴリズムを使用して、呼び出し元関数から指定される RAM のアドレス範囲をテストし、その結果（パス/フェイル）を返します。このモジュールは、アプリケーション・システムの初期化後に実行します。 テスト・ハーネスの制御ファイル（main.c）は、関数“indicate_test_result”を呼び出してテスト結果を処理します。 【注】関数“indicate_test_result”はモジュール stl_main_example_support function.c 内にあります。	
入力パラメータ	
unsigned char __far *addr	テストする RAM の先頭アドレスのポインタ
unsigned short num	テストする RAM 範囲（バイト数）
出力パラメータ	
なし	該当せず
戻り値	
char	CPU レジスタ C のテスト結果 0 = テストはパスしました。 1 = 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。

2.3.2.3 イニシャル・マーチ C

イニシャル・マーチ C テストはアプリケーション・システムの初期化前に実行します。実行にはテスト・ハーネスからの関数呼び出しを使用しません。テストの起動は修正した“startup.asm”モジュールからのジャンプで行い、“startup.asm”モジュールへの復帰もジャンプで行います。テスト結果は、8 ビット・アキュムレータ（A）に格納されます。このため、システムを起動し“C”環境を初期化する前に RAM の全領域をテストすることができます。

このテストは 8 ビットで RAM にアクセスするように設定されています。

表 11　ソース・ファイル：イニシャル・マーチ C

STLファイル名	ヘッダ・ファイル
stl_RL78_march_c_initial.asm	なし
テスト・ハーネス・ファイル名	ヘッダ・ファイル
startup.asm	なし

宣言	
stl_RL78_march_c_initial	
説明	
マーチ C アルゴリズムを使用して、呼び出し元関数から指定される RAM のアドレス範囲をテストし、その結果（パス/フェイル）を返します。このモジュールは、アプリケーション・システムの初期化前に実行します。関数呼び出しは使用しません。 テスト・ハーネスの制御ファイル（main.c）は、関数“indicate_test_result”を呼び出してテスト結果を処理します。 【注】関数“indicate_test_result”はモジュール stl_main_example_support function.c 内にあります。	
入力パラメータ	
CPUレジスタAX	テストする RAM の先頭アドレスを格納する 16 ビット・レジスタ
CPUレジスタBC	テストする RAM 範囲（バイト数）を格納する 16 ビット・レジスタ
出力パラメータ	
なし	該当せず
戻り値	
CPUレジスタA	テスト結果 0 = テストはパスしました。 1 = 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。

2.3.2.4　イニシャル・マーチ X

イニシャル・マーチ X テストはアプリケーション・システムの初期化前に実行します。実行にはテスト・ハーネスからの関数呼び出しを使用しません。テストの起動は修正した“startup.asm”モジュールからのジャンプで行い、“startup.asm”モジュールへの復帰もジャンプで行います。テスト結果は、8 ビット・アキュムレータ（A）に格納されます。このため、システムを起動し“C”環境を初期化する前に RAM の全領域をテストすることができます。

このテストは 8 ビットで RAM にアクセスするように設定されています。

表 12　ソース・ファイル：イニシャル・マーチ X

STLファイル名	ヘッダ・ファイル
stl_RL78_march_x_initial.asm	なし
テスト・ハーネス・ファイル名	ヘッダ・ファイル
startup.asm	なし

宣言	
stl_RL78_march_x_initial	
説明	
マーチ X アルゴリズムを使用して、呼び出し元関数から指定される RAM のアドレス範囲をテストし、その結果（パス/フェイル）を返します。このモジュールは、アプリケーション・システムの初期化前に実行します。関数呼び出しは使用しません。 テスト・ハーネスの制御ファイル（main.c）は、関数“indicate_test_result”を呼び出してテスト結果を処理します。 【注】関数“indicate_test_result”はモジュール stl_main_example_support function.c 内にあります。	
入力パラメータ	
CPUレジスタAX	テストする RAM の先頭アドレスを格納する 16 ビット・レジスタ
CPUレジスタBC	テストする RAM 範囲（バイト数）を格納する 16 ビット・レジスタ
出力パラメータ	
なし	該当せず
戻り値	
CPUレジスタA	CPU レジスタ A のテスト結果 0 = テストはパスしました。 1 = 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。

2.3.2.5　スタック領域テスト(マーチ C)

テスト・ハーネスからの通常の関数呼び出しで実行するために C スタック・リソースを使用します。STACK 領域の全部のテストが可能です。テストは破壊的であるため現在の状態をバッファに退避してからテストを行います。STACK_TEST_AREA パラメータの offset をテスト毎に切り替えることで、部分的にテストすることが可能です。

RAM テストは、システム・マーチ C を用いて行います。

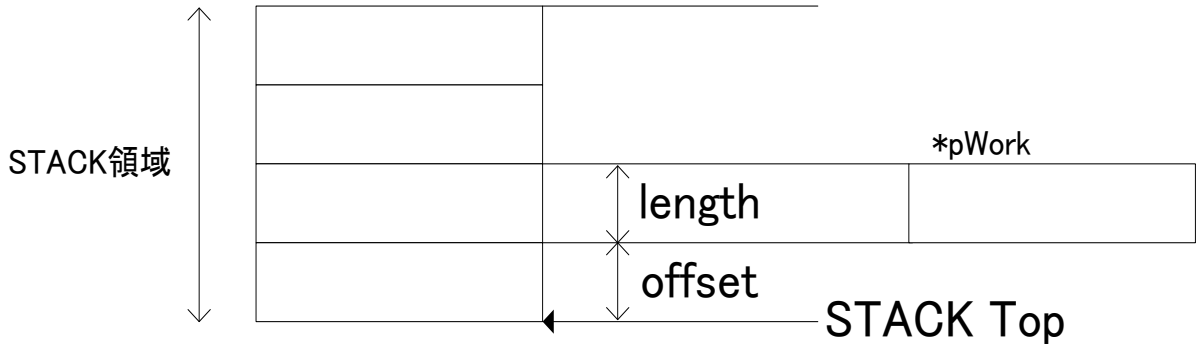
表 13　スタック領域テスト(マーチ C)

STLファイル名	ヘッダ・ファイル
stl_RL78_RamTest_Stacks_c.asm	なし
テスト・ハーネス・ファイル名	ヘッダ・ファイル
main.c	main.h

宣言	
char stl_RL78_RamTest_Stacks_c(STACK_TEST_AREA *p)	
説明	
スタックポインタ(SP)を指定された領域に切り替え、マーチ C アルゴリズムを使用して、指定されたバッファ RAM のアドレス範囲をテストし、その結果（パス/フェイル）が正常であれば、スタック領域の内容をバッファ RAM にコピーします。続いて、マーチ C アルゴリズムを使用して、スタック領域をテストし、バッファ RAM に退避した内容とスタックポインタ(SP)を復元します。そしてテスト結果（パス/フェイル）を返します。このモジュールは、アプリケーション・システムの初期化後に実行します。 テスト・ハーネスの制御ファイル（main.c）は、関数“indicate_test_result”を呼び出してテスト結果を処理します。 【注】関数“indicate_test_result”はモジュール stl_main_example_support function.c 内にあります。	
入力パラメータ	
STACK_TEST_AREA *p	バッファ RAM・サイズ・新スタック領域を格納する構造体へのポインタ
出力パラメータ	
なし	該当せず
戻り値	
char	CPU レジスタ C のテスト結果 0 = テストはパスしました。 1 = 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。

ソース・ファイル：スタック領域テストパラメータ構造体

構文	
static STACK_TEST_AREA stack_test;	
説明	
main.c の呼び出し元関数からスタック領域テストモジュール（stl_RL78_RamTest_Stacks_c.asm）に渡されるパラメータを提供する構造体宣言とインスタンス。 【注】stl_RL78_RamTest_Stacks_x 関数の構造体と同じです。	
入力パラメータ	
char *pWork;	スタックの内容を退避するエリアの先頭アドレス
unsigned short length	テスト対象サイズ
unsigned short offset	テスト対象スタック領域(スタック TOP からのオフセット)
char *pNewSp	テスト中に一時的に使用するスタックポインタ
出力パラメータ	
なし	該当せず
戻り値	
なし	該当せず



2.3.2.6　スタック領域テスト(マーチ X)

テスト・ハーネスからの通常の関数呼び出しで実行するために C スタック・リソースを使用します。STACK 領域の全部のテストが可能です。テストは破壊的であるため現在の状態をバッファに退避してからテストを行います。STACK_TEST_AREA パラメータの offset をテスト毎に切り替えることで、部分的にテストすることができます。

RAM テストは、システム・マーチ C を用いて行います。

表 14　スタック領域テスト(マーチ X)

STLファイル名	ヘッダ・ファイル
stl_RL78_RamTest_Stacks_x.asm	なし
テスト・ハーネス・ファイル名	ヘッダ・ファイル
main.c	main.h

宣言	
char stl_RL78_RamTest_Stacks_x(STACK_TEST_AREA *p)	
説明	
スタックポインタ(SP)を指定された領域に切り替え、マーチ X アルゴリズムを使用して、指定されたバッファ RAM のアドレス範囲をテストし、その結果（パス/フェイル）が正常であれば、スタック領域の内容をバッファ RAM にコピーします。続いて、マーチ X アルゴリズムを使用して、スタック領域をテストし、バッファ RAM に退避した内容とスタックポインタ(SP)を復元します。そしてテスト結果（パス/フェイル）を返します。このモジュールは、アプリケーション・システムの初期化後に実行します。 テスト・ハーネスの制御ファイル（main.c）は、関数“indicate_test_result”を呼び出してテスト結果を処理します。 【注】関数“indicate_test_result”はモジュール stl_main_example_support function.c 内にあります。	
入力パラメータ	
STACK_TEST_AREA *p	バッファ RAM・サイズ・新スタック領域を格納する構造体へのポインタ
出力パラメータ	
なし	該当せず
戻り値	
char	CPU レジスタ C のテスト結果 0 = テストはパスしました。 1 = 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。

ソース・ファイル：スタック領域テストパラメータ構造体

構文	
static STACK_TEST_AREA stack_test;	
説明	
main.c の呼び出し元関数からスタック領域テストモジュール（stl_RL78_RamTest_Stacks_x.asm）に渡されるパラメータを提供する構造体宣言とインスタンス。 【注】stl_RL78_RamTest_Stacks_c 関数の構造体と同じです。	
入力パラメータ	
char *pWork;	スタックの内容を退避するエリアの先頭アドレス
unsigned short length	テスト対象サイズ
unsigned short offset	テスト対象スタック領域(スタック TOP からのオフセット)
char *pNewSp	テスト中に一時的に使用するスタックポインタ
出力パラメータ	
なし	該当せず
戻り値	
なし	該当せず

2.4　システム・クロック・テスト

RL78 セルフテスト・ライブラリには、内部システム・クロック（CPU クロックとペリフェラル・クロック）をテストする目的で 2 つのセルフテスト・モジュール（ハードウェアとソフトウェア）が用意されています。ソフトウェア・モジュールは、従来の製品との下位互換性を確保するためのものです。このため、タイマアレイが特別なハードウェア性能を備えていない RL78 デバイス、またはアプリケーションでタイマを使用するために MCU セルフテストでは使用できない RL78 デバイスで使用することができます。これらのモジュールは、アプリケーションの動作中にメイン・システム・クロックの正常および異常な動作をアプリケーションで検出するために使用します。ただし、内部の低速オシレータで計測を行う場合は誤差が大きいためにシステム・クロックの計測精度が低下するので注意してください。したがって、システム・クロックの相対的な動作しか分かりませんが、システム・クロックが正常に動作していることおよび値が許容される限度内であることを確認する上では問題ありません。

これらの計測方法の基本的な処理は、メイン・クロックの動作時の周波数が所定の範囲を超えた場合に、それをシステムで検出することです。計測精度は基準クロックソースの精度で決まります。たとえば、外部の信号入力または 32 KHz の水晶を使用すれば、内部の低速オシレータよりもシステム・クロックの計測精度が上がります。ただし、この場合は別のコンポーネントが必要です。

テスト結果は“パス/フェイル”で返します。また、“基準クロックなし”の検出手段も組み込まれており、その異常があれば通常テストとは別のテスト結果を返します。戻り値形式はソフトウェアとハードウェアの計測関数で共通です。

モジュールは、ユーザがヘッダ・ファイル“stl_clocktest_h”で定義した基準値に基づいて、計測した（キャプチャした）タイム値が基準ウィンドウ内（上限値と下限値の間）にあるかどうかを照合します。このヘッダ・ファイルは、ソフトウェア計測とハードウェア計測の基準値およびソフトウェア計測の入力テスト・ポート・ピンを定義します。

2.4.1　ハードウェア計測

現行のすべての RL78 デバイスのタイマ・アレイ・ユニット（TAU）チャネル 5 には、システム・クロック動作をテストする入力キャプチャ・ソースを選択するためのオプションが用意されています。このキャプチャ入力は “セイフティ”レジスタ（TIS0）の中で選択します。オプションは以下の通りです。

マイコンによりクロックテスト用の入力ソース選択ができるチャネルが異なるため使用するマイコンに合わせて変更願います

- 内部低速オシレータ（fiL）
- 外部 32 KHz オシレータ（サブ・クロック）（fsub）
- 外部信号入力（TIO5）

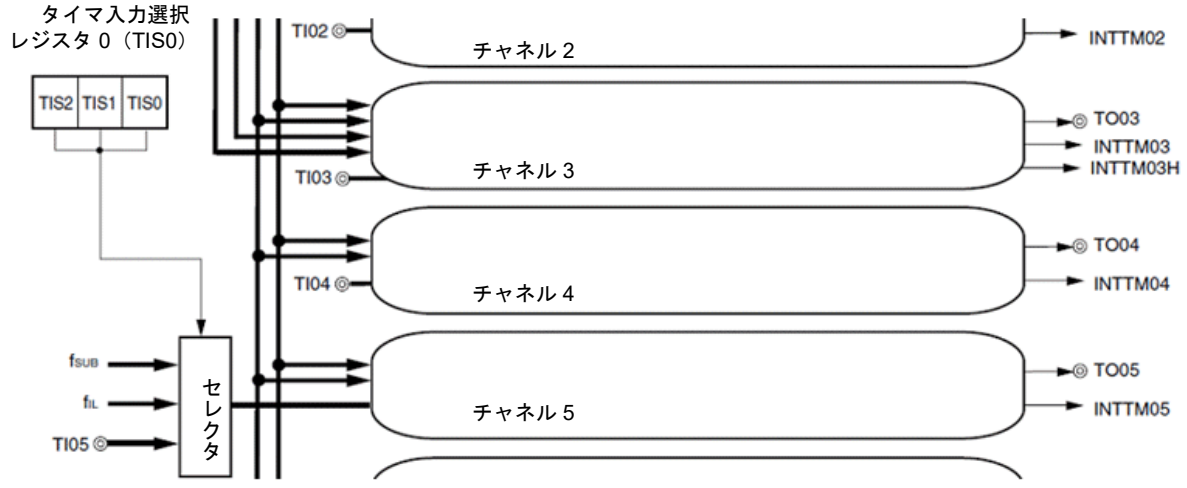


図 8　タイマ・アレイ・ユニット・チャネル 5 の構成

ハードウェア計測の基本的な処理は、TAU チャネル 5 の基準クロックの入力キャプチャ計測に準じます。ハードウェア・キャプチャ計測であるため、キャプチャするタイム値はシステム・クロックの基準クロックに基づく“周期”です。これはソフトウェア計測よりも高精度です。

計測手順は以下の通りです。

- 基準クロックに同期（最初のキャプチャ・イベントを待機）します
- 次のキャプチャ・イベントを待機します
- キャプチャ・レジスタの値と基準値の上限および下限とを比較します

テスト・ハーネスのサンプルは以下の設定を想定しています。

システム・クロック　= 32 MHz

基準クロック　= 32 kHz

計算式は 32000000/32768 = 976（h’3D0）

キャプチャ値には、基準値の上限および下限に対して許容される変動幅を設定してください。

2.4.2　ソフトウェア計測

ソフトウェア計測の基本的な処理は、ソフトウェア・カウンタによるテスト・ポート・ピンの変化の計測です。ただし、実際の比較値を求めるには計算と計測が必要で、同期処理および入力状態の監視には変化がともなうために計測値を正確に計算することは困難です。

計測手順は以下の通りです。

- 基準クロック（入力ピンの High から Low への変化）に同期します。
- 次の Low から High への変化を待機してソフトウェア・カウンタを起動します。
- 次の High から Low への変化までソフトウェア・カウンタをインクリメントします。
- ソフトウェア・カウンタ値と基準値の上限および下限とを比較します。

基本的な計算は次の数式で行います。

システム・クロック/（基準クロック/2）　x　カウント・ループで実行されたクロック・サイクル数

【注】ソフトウェア・カウンタの計測期間は基準クロックの 1/2 です。

テスト・ハーネス・プロジェクトのサンプルの設定を利用すると、以下のようになります。

システム・クロックが 32 MHz で基準クロックがサブ・クロックの 32 KHz の場合の計算式は、次の通りです。

32000000/(32768 / 2) x　ループ・カウント

以下の図 9 の引用コードに示すように、サイクル・カウントは次のように計算します。

基準クロックの½ = 15.26uS（32 KHz/2）

計測期間のループ・カウント（High 期間の計測）は 9 クロック・サイクルです。

32 MHz の場合で 281. 25nS（9 x 31.25nS）

以上から、テスト・ハーネスのサンプルの概算のソフトウェア・カウントは 15.26uS/281nS = 55（h’37）です。

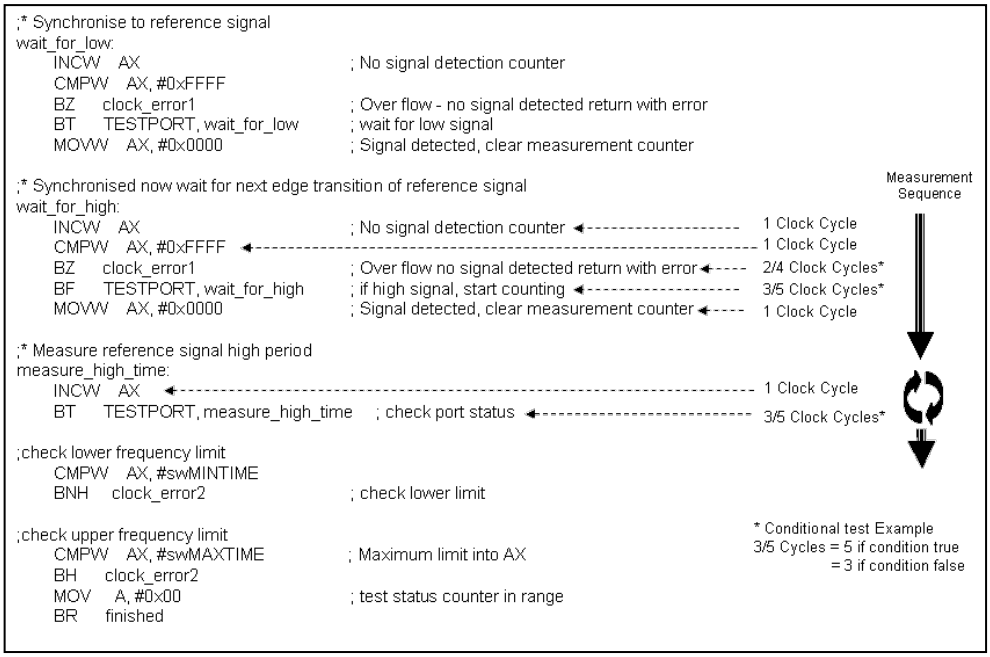


図 9　タイマ・アレイ・ユニット・チャネル 5 の構成

表 15　ソース・ファイル：ソフトウェア・クロック・テスト

STLファイル名	ヘッダ・ファイル
stl_RL78_sw_clocktest.asm	stl_clocktest.h stl.h
テスト・ハーネス・ファイル名	ヘッダ・ファイル
main.c stl_global_data_example.c stl_main_example_support function.c stl_peripheralinit.c	main.h stl_gobal_data_example.h

宣言	
char stl_RL78_sw_clocktest(void)	
説明	
ソフトウェア計測（ソフトウェア・カウンタ）を使用してシステム・クロックをテストします。計測結果（ソフトウェア・カウント）をクロック・テストのヘッダ・ファイル（stl_clocktest.h）で定義される上限値および下限値と比較して、結果（成功/失敗/基準クロックなし）を呼び出し元の関数に返します。 基準値の範囲は以下の数式で計算します。 システム・クロック/（基準クロック/2） x カウント・ループで実行されたクロック・サイクル数 テスト・ハーネスの制御ファイル（main.c）は、関数“indicate_test_result”を呼び出してテスト結果を処理します。 【注】関数“indicate_test_result”はモジュール stl_main_example_support function.c 内にあります。	
入力パラメータ	
swMAXTIME	比較上限値（stl_clocktest.h による定義）
swMINTIME	比較下限値（stl_clocktest.h による定義）
TESTPORT	外部基準信号入力（stl_clocktest.h による定義）のテスト・ポート入力ピン
出力パラメータ	
なし	該当せず
戻り値	
char	CPU レジスタ C のテスト結果 0 = テストはパスしました。 1 = 計測テストはフェイルとなりました（基準ウィンドウの範囲外です）。 2 = 計測テストはフェイルとなりました（基準クロックは検出されませんでした）。

表 16　ソース・ファイル：ハードウェア・クロック・テスト

STLファイル名	ヘッダ・ファイル
stl_RL78_hw_clocktest.asm	stl_clocktest.h stl.h
テスト・ハーネス・ファイル名	ヘッダ・ファイル
main.c stl_global_data_example.c stl_main_example_support function.c stl_peripheralinit.c	main.h stl_gobal_data_example.h

宣言	
char stl_RL78_hw_clocktest(void)	
説明	
ハードウェア計測（TAU チャネル 5）を使用してシステム・クロックをテストします。計測結果（キャプチャ値）をクロック・テストのヘッダ・ファイル（stl_clocktest.h）で定義される上限値および下限値と比較して、結果（パス/フェイル/基準クロックなし）を呼び出し元の関数に返します。 テスト・ハーネスの制御ファイル（main.c）は、関数“indicate_test_result”を呼び出してテスト結果を処理します。 【注】関数“indicate_test_result “はモジュール stl_main_example_support function.c 内にあります。	
入力パラメータ	
hwMAXTIME	比較上限値（stl_clocktest.h による定義）
hwMINTIME	比較下限値（stl_clocktest.h による定義）
CAPTURE_interrupt_FLAG	タイマ・チャネル・キャプチャ割り込みフラグ （stl_clocktest.h による定義）
出力パラメータ	
なし	該当せず
戻り値	
Char	CPU レジスタ C のテスト結果 0 = テストはパスしました。 1 = 計測テストはフェイルとなりました（基準ウィンドウの範囲外です）。 2 = 計測テストはフェイルとなりました（基準クロックは検出されませんでした）。

2.5　A/D コンバータ

2.5.1　A/D コンバータ・テスト

RL78/G14 には、＋側基準電圧、－側基準電圧、内部基準電圧(1.45V)を A/D 変換する機能があります。この機能を使用して A/D コンバータが正常に動作しているかを確認することができます。

表 17　ソース・ファイル：A/D コンバータ・テスト

STLファイル名	ヘッダ・ファイル
stl_adc.c	stl_adc.h stl.h
テスト・ハーネス・ファイル名	ヘッダ・ファイル
main.c stl_main_example_support function.c stl_peripheralinit.c	main.h

宣言	
void stl_ADC_Create (void)	
説明	
ADC を初期化し、静的変数(TEST_DATA)をクリアします。	
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

宣言	
char stl_ADC_Check_TestVoltage (void)	
説明	
静的変数(testVoltageIndex)に基づいて、＋側基準電圧、－側基準電圧、内部基準電圧(1.45V)を A/D 変換します。変換結果（キャプチャ値）をヘッダ・ファイル（stl_adc.h）で定義される上限値および下限値と比較して、結果（パス/フェイル）を呼び出し元の関数に返します。 テスト・ハーネスの制御ファイル（main.c）は、関数“indicate_test_result”を呼び出してテスト結果を処理します。 【注】関数“indicate_test_result “はモジュール stl_main_example_support function.c 内にあります。	
入力パラメータ	
VSS_RANGE_MAX	VSS 比較上限値（stl_adc.h による定義）
VDD_RANGE_MIN	VDD 比較下限値（stl_adc.h による定義）
AD_RESOLUTION_HEX	VDD 比較上限値（stl_adc.h による定義）
VBGR_RANGE_MIN	内部基準電圧比較下限値（stl_adc.h による定義）
VBGR_RANGE_MAX	内部基準電圧比較上限値（stl_adc.h による定義）
testVoltageIndex	静的変数 呼び出し毎に 0,1,2,0..と変化
出力パラメータ	
なし	該当せず
戻り値	
Char	CPU レジスタ C のテスト結果 0 = テストはパスしました。 1 = 計測テストはフェイルとなりました（基準ウィンドウの範囲外です）。

2.6　デジタル出力

RL78/G14 には、端子が出力モード時に端子のデジタル出力レベルをリードすることができる機能があります。この機能を使用して、デジタル出力が正常に動作しているかを確認することができます。

テスト対象ポートは、stl_RL78_GpioTest.h で定義します。

表 18　ソース・ファイル：デジタル出力・テスト

STLファイル名	ヘッダ・ファイル
stl_RL78_GpioTest.asm	stl_RL78_GpioTest.h stl.h
テスト・ハーネス・ファイル名	ヘッダ・ファイル
main.c stl_main_example_support function.c stl_peripheralinit.c	main.h

宣言	
char stl_RL78_GpioTest (void)	
説明	
静的変数(TEST_DATA)に基づいて、0、1を出力します。出力値と端子レベルを比較して呼び出し元の関数に返します。 テスト・ハーネスの制御ファイル（main.c）は、関数“indicate_test_result”を呼び出してテスト結果を処理します。 【注】関数“indicate_test_result “はモジュール stl_main_example_support function.c 内にあります。	
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	
Char	CPU レジスタ C のテスト結果 0 = テストはパスしました。 1 = 計測テストはフェイルとなりました。

2.7　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、必要なときにウォッチドッグがソフトウェアによってリフレッシュされないため、エラーが検出されます。

RL78/G14 のウォッチドッグタイマ（WDT）モジュールが、このために使用されます。WDT はウィンドウ機能を備えており、指定した期間の直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」内に必ずリフレッシュを行うようにしています。エラーが検出された場合、内部のリセットを生成するように設定することができます。WDT 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。

ウォッチドッグタイマの設定は、オプションバイト(000C0H/010C0H)で設定します。

アドレス：000C0/010C0H

<7>	<6>	<5>	<4>	<3>	<2>	<1>	<0>
WDTINT	WINDOW1	WINDOW0	WDTON	WDCS2	WDCS1	WDCS0	WDSTBYON

WDTINT	ウォッチドッグ・タイマのインターバル割り込みの使用／不使用
0	**インターバル割り込みを使用しない。**
1	オーバフロー時間の75% + 1/2 fIL到達時にインターバル割り込みを発生する。

WINDOW1	WINDOW0	ウォッチドッグ・タイマのウインドウ・オープン期間
0	0	設定禁止。
0	1	50%
1	0	75%
1	**1**	**100%**

WDTON	ウォッチドッグ・タイマのカウンタの動作制御
0	カウンタ動作禁止(リセット解除後，カウント停止)。
1	**カウンタ動作許可(リセット解除後，カウント開始)。**

WDCS2	WDCS1	WDCS0	ウォッチドッグ・タイマのオーバフロー時間 (fIL = 17.25 kHz (MAX)の場合)
0	0	0	2^6/fIL (3.71 ms)
0	0	1	2^7/fIL (7.42 ms)
0	1	0	2^8/fIL (14.84 ms)
0	1	1	2^9/fIL (29.68 ms)
1	0	0	2^{11}/fIL (118.72 ms)
1	0	1	2^{13}/fIL (474.90 ms)
1	1	0	2^{14}/fIL (949.80 ms)
1	**1**	**1**	**2^{16}/fIL (3799.19 ms)**

WDSTBYON	ウォッチドッグ・タイマのカウンタ動作制御(HALT/STOPモード時)
0	HALT/STOPモード時，カウンタ動作停止。
1	**HALT/STOPモード時，カウンタ動作許可。**

表 19　ソース・ファイル：ウォッチドッグタイマ・テスト

STLファイル名	ヘッダ・ファイル
stl_wdt.c	stl_wdt.h stl.h
テスト・ハーネス・ファイル名	ヘッダ・ファイル
main.c stl_main_example_support function.c	main.h stl_gobal_data_example.h

宣言	
void stl_wdt_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

2.8　電圧

RL78/G14 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。提供されているサンプルコードは Vcc が指定したレベルより低くなったときに割り込みを発生する方法を示しています。

電源電圧監視の設定は、オプションバイト(000C1H/010C1H)で設定します。

アドレス：000C1/010C1H

〈7〉	〈6〉	〈5〉	4	〈3〉	〈2〉	〈1〉	〈0〉
VPOC2	VPOC1	VPOC0	1	LVIS1	LVIS0	LVIMDS1	LVIMDS0

LVD の設定(割り込み・モード)

検出電圧		オプション・バイト設定値						
V_{LVD}		VPOC2	VPOC1	VPOC0	LVIS1	LVIS0	モード設定	
立ち上がり	立ち下がり						LVIMDS1	LVIMDS0
1.67 V	1.63 V	**0**	0	0	1	1	**0**	**1**
1.77 V	1.73 V		0	0	1	0		
1.88 V	1.84 V		0	1	1	1		
1.98 V	1.94 V		0	1	1	0		
2.09 V	2.04 V		0	1	0	1		
2.50 V	2.45 V		1	0	1	1		
2.61 V	2.55 V		1	0	1	0		
2.71 V	2.65 V		1	0	0	1		
2.81 V	2.75 V		1	1	1	1		
2.92 V	**2.86 V**		**1**	**1**	**1**	**0**		
3.02 V	2.96 V		1	1	0	1		
3.13 V	3.06 V		0	1	0	0		
3.75 V	3.67 V		1	0	0	0		
4.06 V	3.98 V		1	1	0	0		
–		上記以外は設定禁止						

表 20　ソース・ファイル：電圧低下・テスト

STLファイル名	ヘッダ・ファイル
stl_vdc.c	stl_vdc.h stl.h
テスト・ハーネス・ファイル名	ヘッダ・ファイル
main.c	main.h

宣言	
void Stl_VDC_Create (void)	
説明	
電圧低下割り込みを有効にします。Vcc が指定した電圧より低下すると割り込みが発生します。	
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

宣言	
void Stl_VDC_Interrupt (void)	
説明	
Vcc 電圧低下割り込みハンドラ。 Voltage_Test_Failure _interrupt (ユーザ定義処理)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

3. 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 CS+テスト・ハーネス・ワークスペースが提供されます。このコードは本書とともに検討して、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 2 つの部分に分けることができます。

（1）電源投入テスト

電源投入後またはリセット後に実行されるテストです。システムの正常動作を確認するためにできるだけ迅速に実行すべきです。これらのテストは次の通りです。

イニシャル・マーチ C（またはイニシャル・マーチ X）による全 RAM のテスト

全レジスタのテスト

Flash メモリの CRC テスト

クロック・テストは、最大のクロック速度を計測するようにクロック速度の初期値を設定しておけば後から実行することも可能です。

（2）定期テスト

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。本書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。

RAM テスト

本テストでは、一旦システムを初期化した後はメモリを部分的にテストするように設計されているため、“システム”の RAM テスト・モジュールを使用するべきです。アプリケーション・データを保存するバッファ領域のサイズは最小限で済みます。

レジスタ・テスト

本テストが実行されるかどうかは、アプリケーションのタイミングによります。

Flash メモリ・テスト

本モジュールは、複数回の CRC 計算の結果を蓄積できるため、システム動作に合わせて使用することができます。

クロック・テスト・モジュールは、アプリケーション・タイミングに合わせて自由に実行することができます。

以下の各章では、各種類のテストを使用する例を示します。

3.1　CPU

どんな CPU テストであれ不具合が検出された場合は非常に重大です。そこで、この関数で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を目的としています。

3.1.1　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。

3.1.2　定期テスト

CPU レジスタを定期的にテストする場合、関数が単独で実行するように設計されているためにアプリケーションに合わせて任意のタイミングで実行することができます。各関数は、アプリケーション・システムの動作に影響しないように、テストの完了時に元のレジスタ・データを復元します。テスト中は割り込みを禁止にする必要があります。

3.2　Flash ROM

ROM のテストでは、Flash メモリのある領域の内容の CRC 値を計算して、その領域の外の所定の位置にあらかじめ格納されている基準 CRC 値と比較します。

CS+ツール・チェーンでは、CRC 値を計算および累積してユーザが指定する位置に格納することができます。その場合、CS+では「汎用 CRC」「高速 CRC (CCR-16-CCITT)」「高速 CRC (SENT)」の 3 つの種類の CRC が選択可能ですが､本ライブラリで実施しているハードウェア CRC による演算（stl_RL78_peripheral 関数）および CRC 結果の比較値生成のための C 関数（reference_crc_calculation）は、「汎用 CRC」に相当します。「図 16　CS+のオブジェクト・コンバート・オプション」を参照して下さい。

本ライブラリで実施しているソフトウェア CRC（stl_RL78_sw_crc_asm 関数）の基準 CRC 値は、CS+で生成できません。reference_crc_CCIT16_Msb_calculation を参考にして下さい。

「図 10　基準 CRC 値の追加」を参照してください。

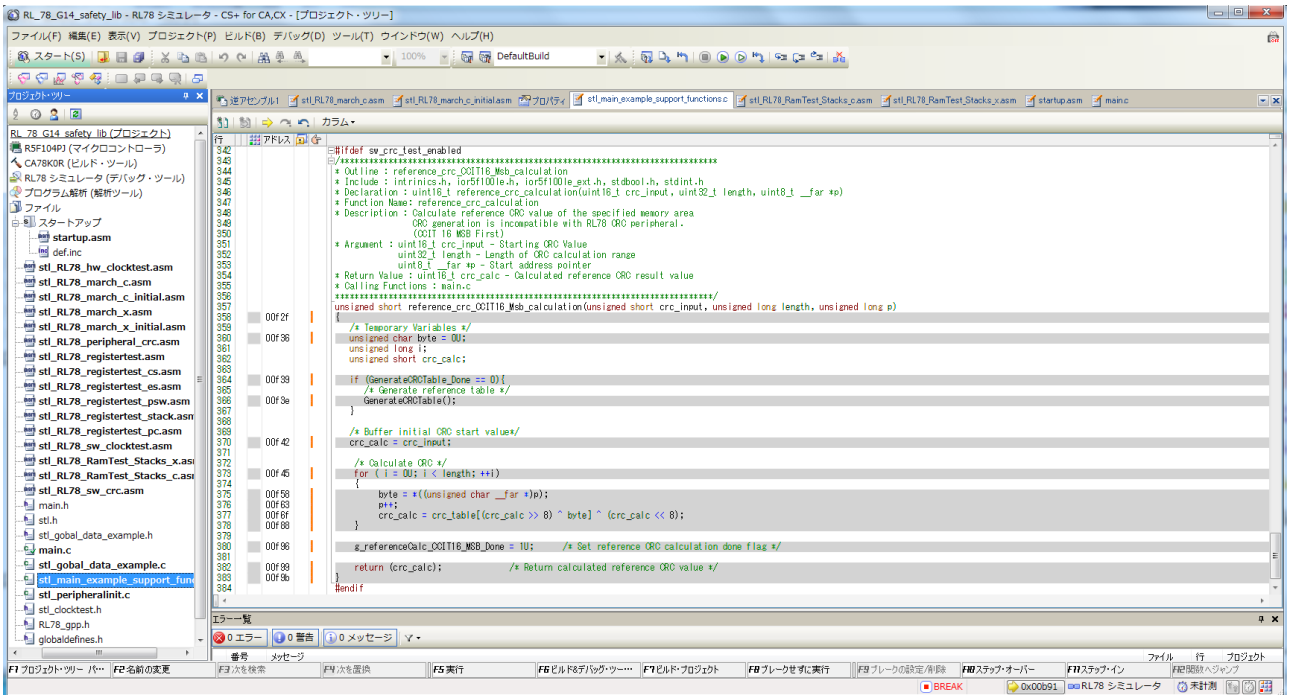


図 10　基準 CRC 値の追加

3.2.1　電源投入テスト

使用されたすべての ROM は電源投入時にテストしてください。ハードウェアおよびソフトウェアの CRC モジュールは、メモリの全範囲の CRC 値を計算することが可能です。

3.2.2　定期テスト

Flash メモリの定期テストは､アプリケーションの空き時間を利用しながら複数回に分けて実行することが推奨されます。また、ソフトウェア・モジュールを使用する場合は部分計算の結果をアプリケーションで保存する必要があります。この値は、次回の CRC 計算の初期値として使用します。

ハードウェアのペリフェラル・ユニットを使用する場合は、CRC の部分計算の結果をハードウェア CRC のペリフェラル・ユニットの結果レジスタに残しておくことも可能ですが、別に保存しておいて次の計算を実行する前に比較することが推奨されます。

この方法により、アプリケーションに都合のよいタイム・スロットを利用してすべての Flash メモリをテストすることができます。

3.3　RAM

RAM をテストする際には以下のことに注意してください。

- テスト対象の RAM は、そのときのスタックも領域含めて、他の領域に使用することはできません。
- 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。
- スタック領域は、バックアップ領域とテスト期間中で使用するスタック領域を指定することで、コピー/テスト/復元を行います。ただし、この操作の間は割り込み処理はできません。

3.3.1　電源投入テスト

イニシャル RAM テスト・モジュール（マーチ C またはマーチ X）を使用することが推奨されます。これらのモジュールは、電源投入時またはリセット時におけるすべての RAM 領域のテスト専用に設計されています。また、関数呼び出しが不要ですが RAM の内容が破壊されることから、システムおよび C スタックを初期化する前に実行するのに適しています。本ライブラリでは、アセンブラ・ファイル startup.asm に、イニシャル RAM テストが実装されています。

3.3.2　定期テスト

RAM の定期テストは、通常はアプリケーションの空き時間を利用しながら複数回に分けて実行することが推奨されます。また、テスト中は RAM の内容を一時的に保存するための領域が必要です。各テストでは、指定した範囲に対するパス/フェイルの結果が通知されます。これにより、アプリケーションに都合のよいタイム・スロットを利用してすべての RAM をテストすることができます。

3.4　システム・クロック

システム・クロックに不具合が検出された場合は、非情に重大です。そこでこの関数では、定義済みの別のクロックによる、システムが制御可能な安全な状態に移動することを目的としています。

3.4.1　電源投入テスト

システム・クロックは、電源投入時またはリセット時にテストする必要があります。また、システムを初期化した場合およびシステム・クロック周波数を全面的に設定して動作が安定した場合も、クロックのテストが必要です。

3.4.2　定期テスト

システム・クロックの定期テストはアプリケーションの空き時間を利用して行います。この理由は、クロック計測の精度を高める目的から、基準クロックが通常はシステム・クロックに比べて非常に低速であるためです。

（システム・クロック　= 32 MHz、基準クロック　= 15 KHz）

3.5　A/D コンバータ

3.5.1　電源投入テスト

電源投入時にも定期テストと同様の stl_ADC_Check_TestVoltage 関数を使用して ADC モジュールをテストすることができます。この関数では、－側基準電圧、＋側基準電圧、内部基準電圧(1.45V)のいずれかの AD 変換が実行されます。

3.5.2　定期テスト

定期テストは stl_ADC_Check_TestVoltage 関数を定期的に呼び出す必要があります。に基準変換を実行します。基準電圧は－側基準電圧、＋側基準電圧、内部基準電圧(1.45V)の間で切り替わります。

3.6　デジタル出力

3.6.1　電源投入テスト

電源投入時にも定期テストと同様の stl_RL78_GpioTest 関数を使用してデジタル出力をテストすることができます。この関数では、０出力、１出力のいずれかの出力値の確認が実行されます。

3.6.2　定期テスト

定期テストは stl_RL78_GpioTest 関数を定期的に呼び出す必要があります。出力値は０、１の間で切り替わります。

3.7　ウォッチドッグ

ウォッチドッグタイマ機能は、オプションバイト(000C0H/010C0H)で設定します。リセット解除後、ウォッチドッグ・タイマはカウント動作を開始します。この後、ウォッチドッグのタイムアウトとリセットの実行を阻止するようにウォッチドッグを定期的にリフレッシュする必要があります。ウィンドウ機能を使用している場合、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合わせた期間にリフレッシュする必要があることに注意してください。ウォッチドッグのリフレッシュは、以下を呼び出すことで行われます。

```
/*定期的にウォッチドッグをリフレッシュしてリセットの実行を阻止*/
stl_wdt_Kick ();
```

ウォッチドッグがエラー検出時にリセットを発生するように構成されている場合、ユーザはこれによって生じる割り込みを処理する必要があります。サンプルプログラムでは、ウォッチドッグがエラー検出時にWatchdog_Test_Failure 関数を呼び出すようにしています。

3.8　電圧

電圧検出回路は、オプションバイト(000C1H/010C1H)の設定で、主電源電圧を監視するように構成されています。サンプルでは、電圧が 2.86V より低くなった場合に割り込みを発生するように電圧監視をセットアップしています。

サンプルプログラムでは、電圧低下検出時に Voltage_Test_Failure_interrupt 関数を呼び出すようにしています。

3.9　コード・カバレッジ

コード・カバレッジは、シミュレータ・モードで関数リスト・セクションを表示して確認することができます。

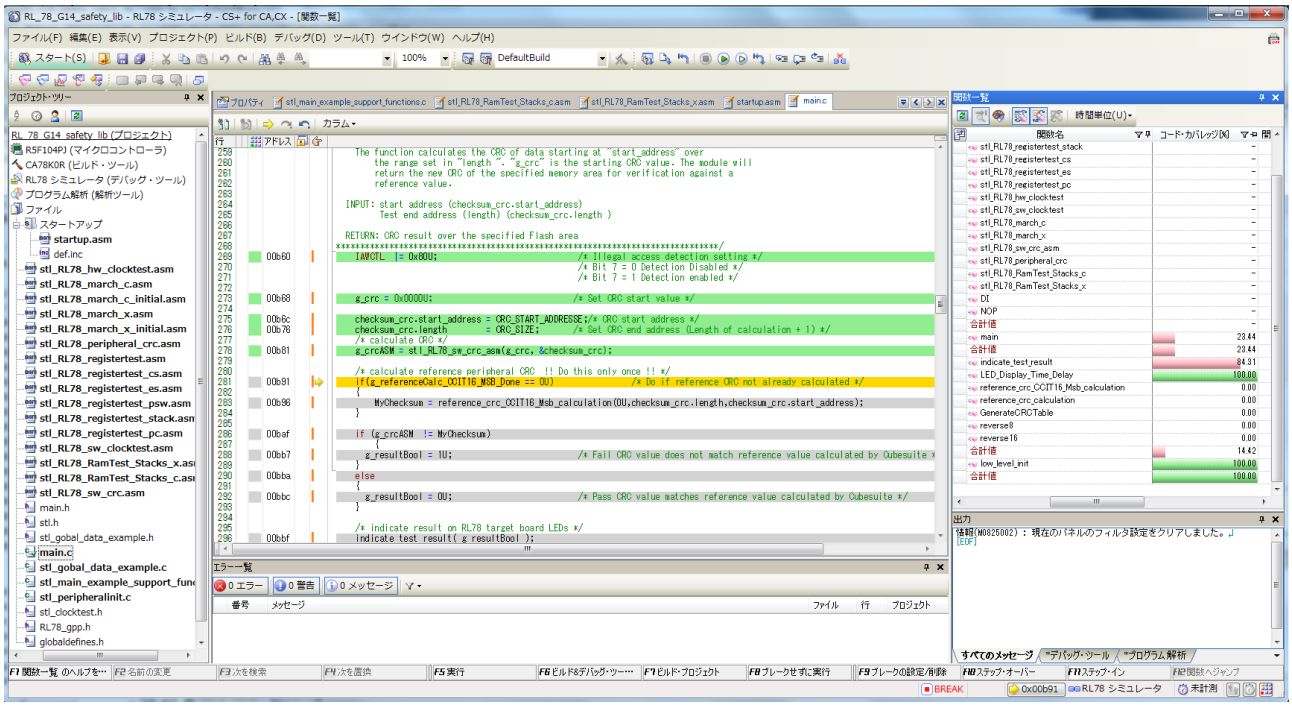


図 11　関数リスト・セクション

4.　ベンチマーク

4.1　開発環境

- E1(R0E000010KCE00)　オンチップデバッギングエミュレータ
- QB-R5F104PJ-TB　RL78/G14 ターゲット・ボード（100 ピン LQFP、14x 14mm）
- ツール・チェーン　CS+ for　CA/CX　V4.03.00　CA78K0R　V1.72

- MCU　R5F104PJAFB
- 内部クロック　32 MHz 高速オシレータ
システム・クロック　= 32 MHz
- 外部サブ・クロック　32 KHz

4.2　CS+の設定

以降では、テスト・プロジェクトの所定のオプションと設定を示します。図には、変更したオプションおよび設定のみを示します。それ以外はすべて CS+のデフォルトのプロジェクト設定です。

4.2.1　一般オプション

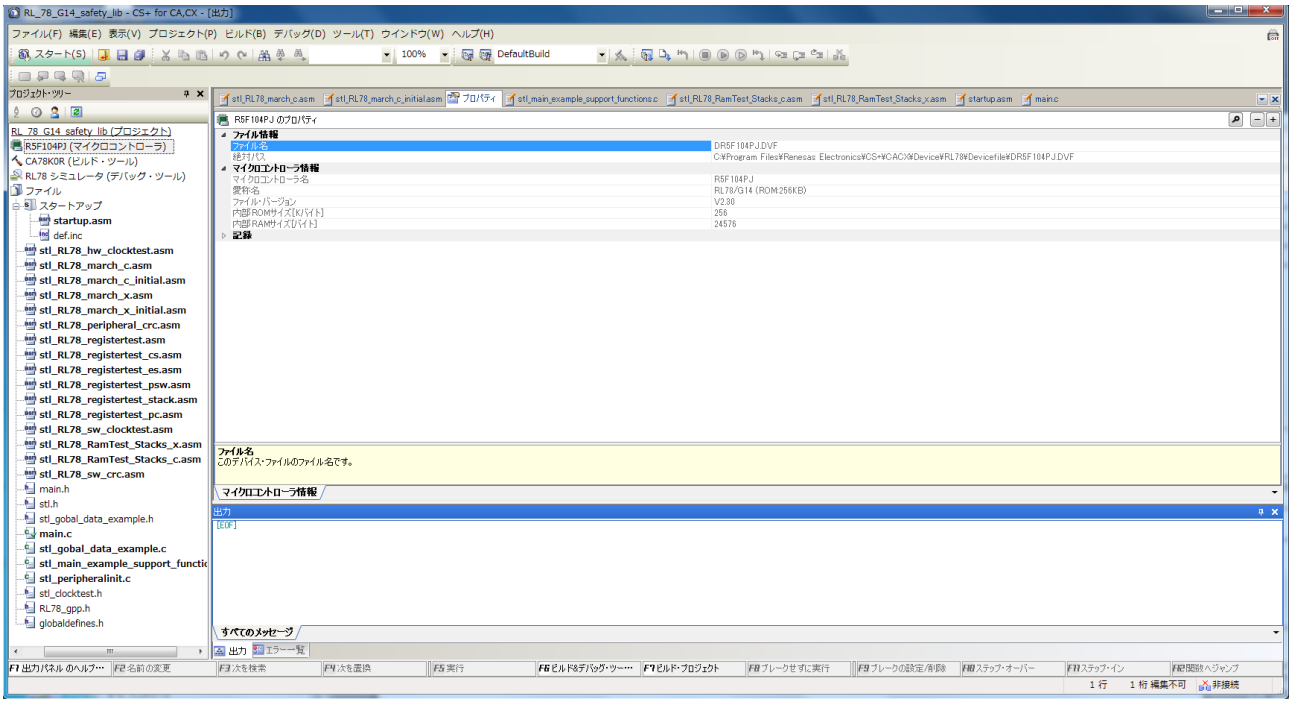


図 12　CS+の共通オプション－動作確認デバイス

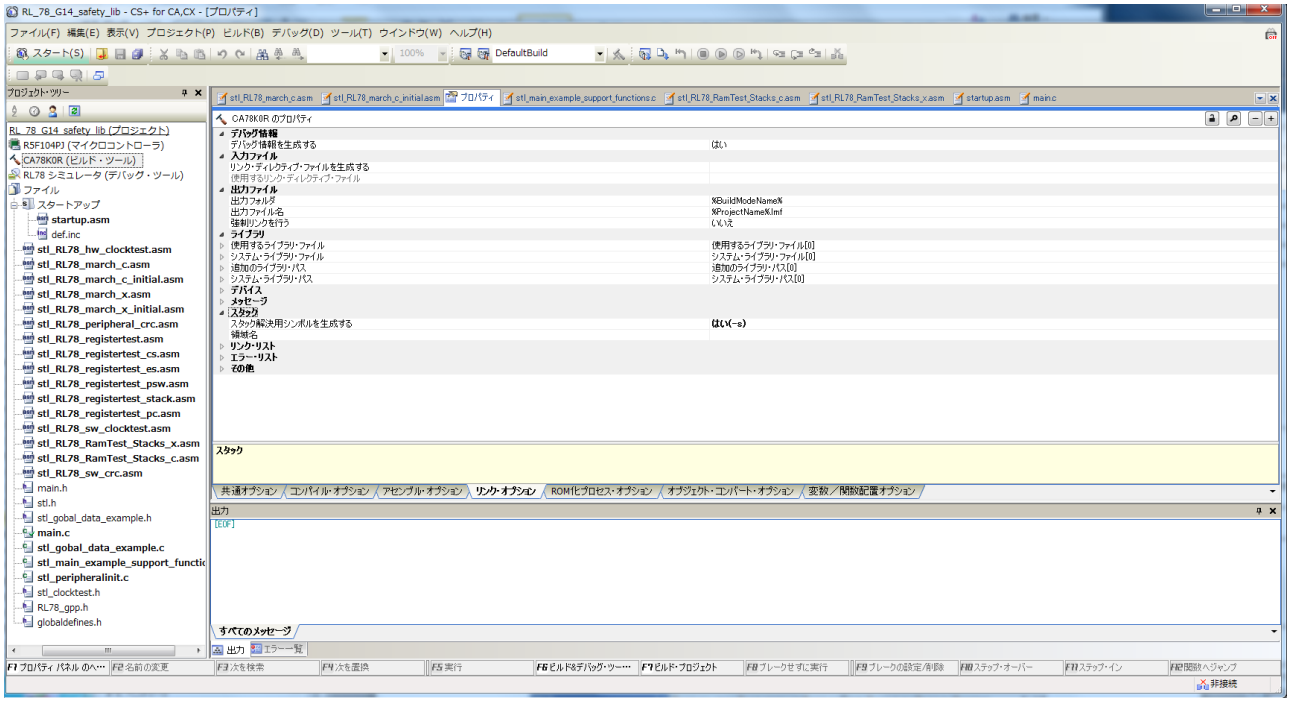


図 13　CS+のリンク・オプション

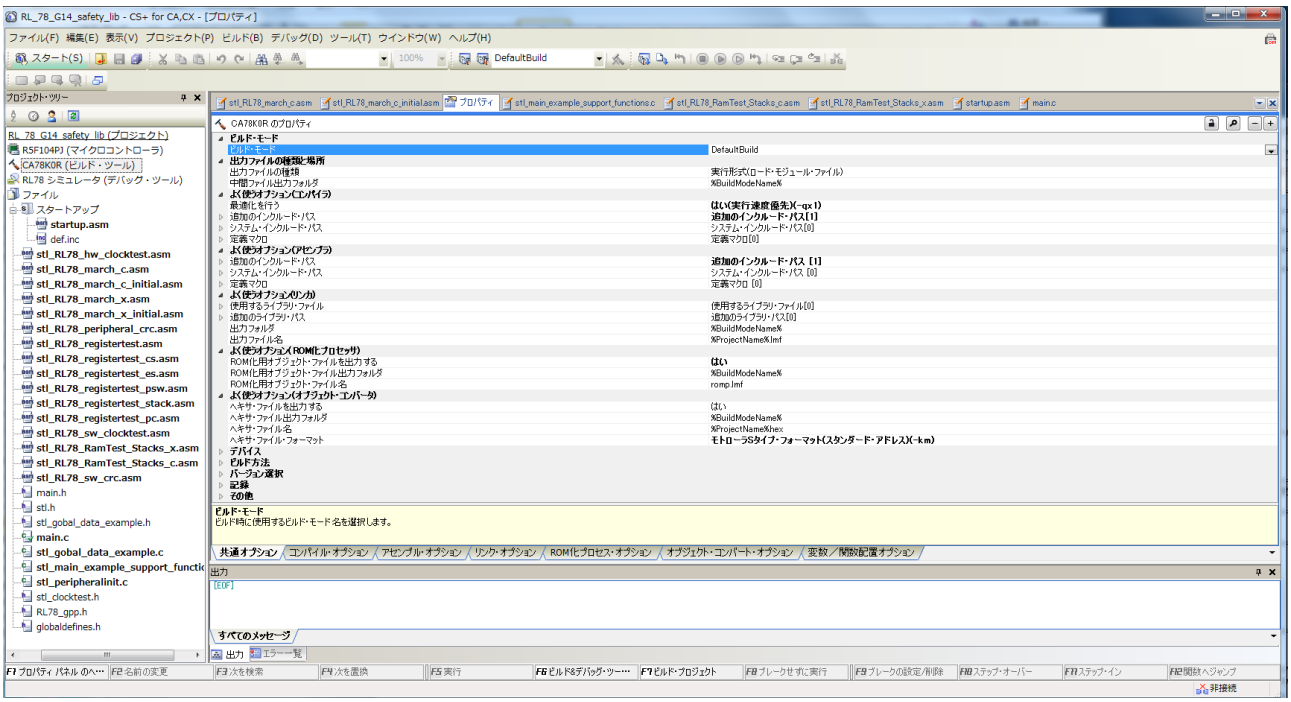


図 14　CS+の共通オプション

4.2.2　コンパイラ設定

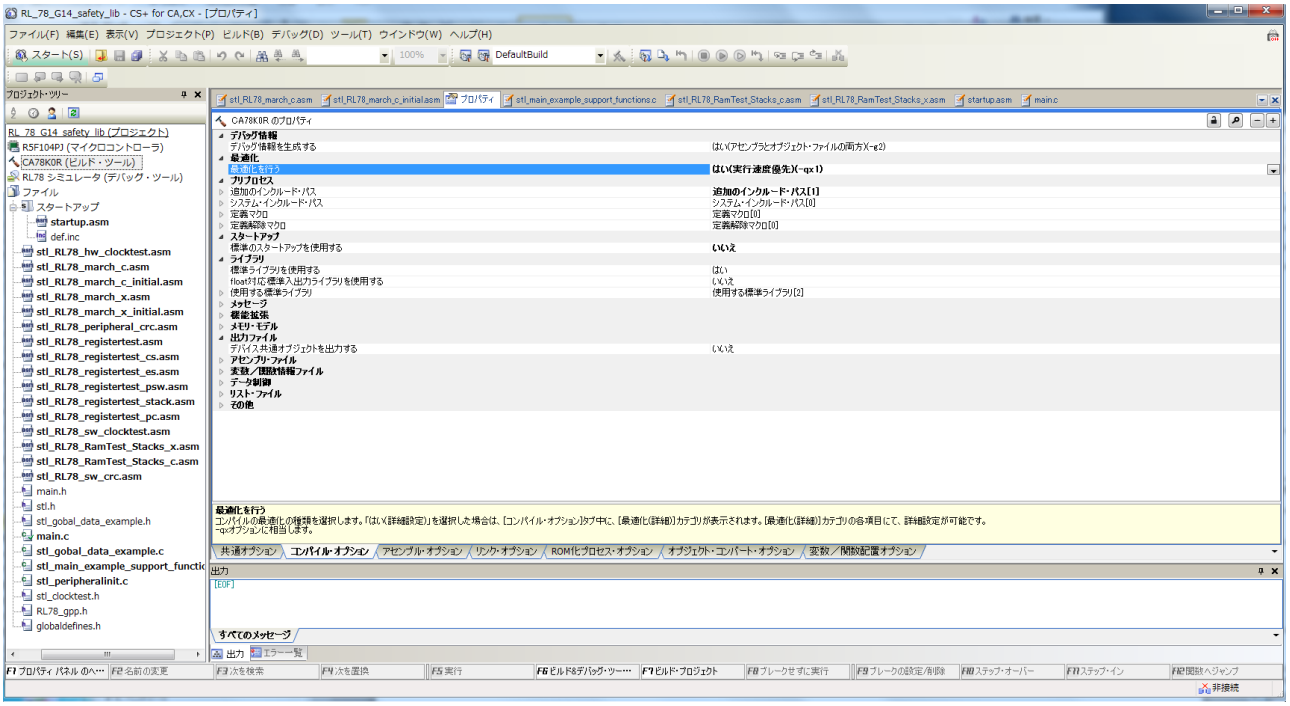


図 15　CS+のコンパイラ・オプション

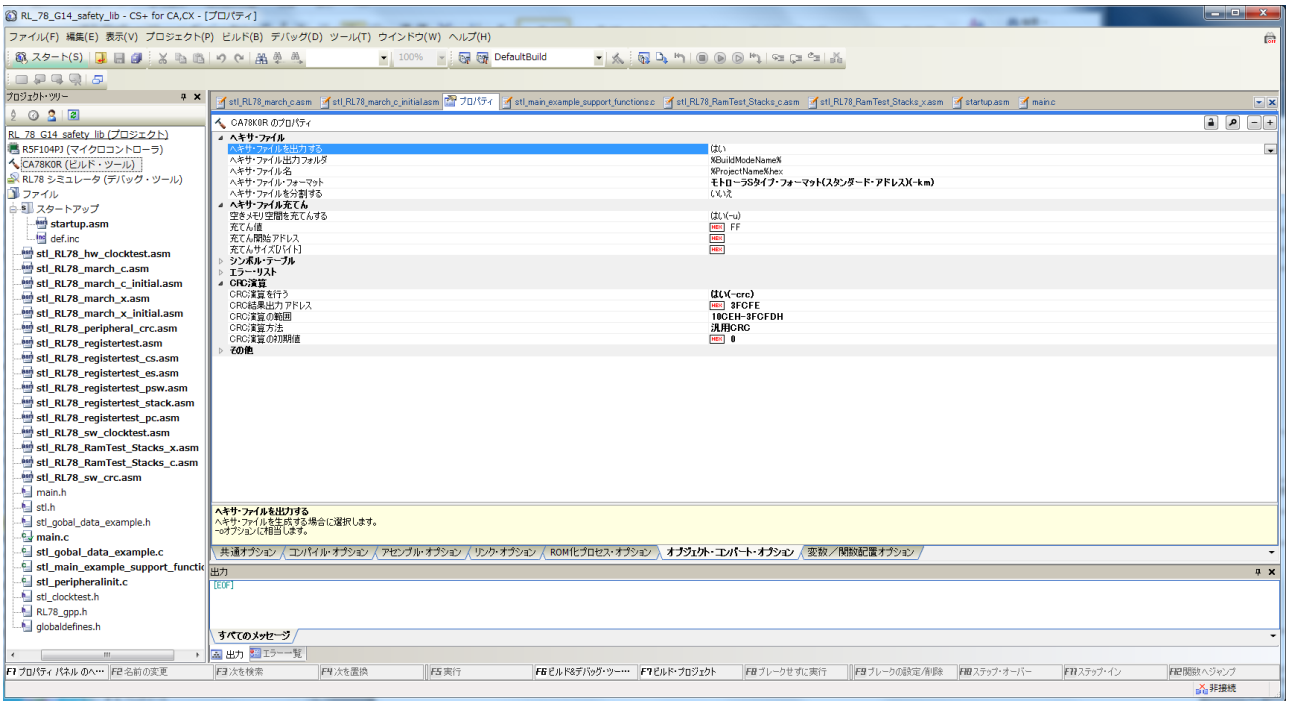


図 16　CS+のオブジェクト・コンバート・オプション

4.3　ベンチマークテスト結果

ライブラリ関数	テスト対象バイト数	処理時間
CPUワーク・レジスタ・テスト stl_RL78_registertest	-	10.312µs
CPUレジスタ・テストーPSW stl_RL78_registertest_psw	-	1.375µs
CPUレジスタ・テストーSP stl_RL78_registertest_stack	-	1.156µs
CPUレジスタ・テストーCS stl_RL78_registertest_cs	-	1.062µs
CPUレジスタ・テストーES stl_RL78_registertest_es	-	1.062µs
CPUレジスタ・テストーPC stl_RL78_registertest_pc	-	2.218µs
ソフトウェアCRC stl_RL78_sw_crc_asm	257072バイト	394200µs(394.200ms)
ハードウェアCRC stl_RL78_peripheral_crc	257072バイト	185000µs (185.000ms)
システムマーチC stl_RL78_march_c	128バイト	1434µs(1.434ms)
システムマーチX stl_RL78_march_x	128バイト	797.906µs
イニシャルマーチC stl_RL78_march_c_initia	24540バイト	274542µs (274.542ms)
イニシャルマーチX stl_RL78_march_x_initial	24540バイト	152609µs(152.609ms)
ハードウェア・クロック・テスト stl_RL78_hw_clocktest	-	56.40µs
ソフトウェア・クロック・テスト stl_RL78_sw_clocktest	-	5700µs (5.700ms)
スタック領域テスト(マーチC) stl_RL78_RamTest_Stacks_c	512バイト+512バイト	11782µs (11.782ms)
スタック領域テスト(マーチX) stl_RL78_RamTest_Stacks_x	512バイト+512バイト	6694µs(6.694ms)

5. 追加ハードウェア・リソース

RL78 シリーズには、ユーザ・サポートとして以下の安全性およびセルフテスト機能が用意されています。これらの追加機能は VDE では認定されていませんが、有用性が高いリソースとして参考のために紹介します。

5.1 追加安全機能

RL78 シリーズの MCU には以下の安全機能が追加されています。

5.1.1 RAM・パリティ・ジェネレータ・チェッカ

この機能をイネーブルすると、RAM の任意の領域に書き込まれる各バイトのパリティ・チェックが実行されます。パリティは、RAM にデータが書き込まれるときに生成され、その RAM からデータが読み出されるときにチェックされます。

この機能はデータ・アクセスに対してのみ使用可能で、RAM から実行するコードには使用できません。RAM でパリティ・エラーが検出されると内部リセットが生成されます。リセット・ソースは“RESF”レジスタを調べて判定できます。リセット・ソースが無効なメモリ・アクセスの場合は、“IAWRF”ビットがセットされます。

RAM パリティ・エラー制御レジスタ（RPECTL）のフォーマット

アドレス：F00F5H　リセット時：00H　R/W

記号	<7>	6	5	4	3	2	1	<0>
RPECTL	RPERDIS	0	0	0	0	0	0	RPEF

RPERDIS	パリティ・エラー・リセット・マスク・フラグ
0	パリティ・エラー・リセットをイネーブルする。
1	パリティ・エラー・リセットをディスエーブルする。

RPEF	パリティ・エラー・ステータス・フラグ
0	パリティ・エラーは発生していない。
1	パリティ・エラーが発生した。

図 17　RAM パリティ・エラーのチェック

5.1.2　RAM ガード保護

この書き込み保護機能をイネーブルすると、RAM の指定した領域からのデータの読み出しはできますが、その領域への書き込みはできません。この領域に書き込みを行ってもエラーは発生しません。

この機能の設定が可能な RAM 領域は限定されており、図 22 に示すように“GRAM0、GRAM1”ビットで選択します。

無効メモリ・アクセス検出制御レジスタ（IAWCTL）のフォーマット

アドレス：F0078H　リセット時：00H　R/W

記号	7	6	5	4	3	2	1	0
IAWCTL	IAWEN	0	GRAM1	GRAM0	0	GPORT	GINT	GCSC

IAWEN	無効メモリ・アクセス検出の制御
0	無効メモリ・アクセス検出をディスエーブルする。
1	無効メモリ・アクセス検出をイネーブルする。

GRAM1	GRAM0	RAM保護空間
0	0	ディスエーブル。RAMは書き込み可能。
0	1	下位RAMアドレスの128バイト
1	0	下位RAMアドレスの256バイト
1	1	下位RAMアドレスの512バイト

図 18　RAM ガード保護

5.1.3　無効メモリ・アクセス保護

この機能は、無効メモリ・アクセスを検出するためにさらに保護を設定します。

“IAWCTL”レジスタの“IAWEN”ビットがセットされている場合は、リセット以外でディスエーブルすることはできません。また、Flash メモリのオプション・バイト・レジスタでウォッチドッグがイネーブルされている場合は、無効メモリ保護は自動的にイネーブルされます。

無効メモリ・アクセスが検出されると内部リセットが生成されます。リセット・ソースは “RESF”レジスタを調べて判定できます。リセット・ソースが無効なメモリ・アクセスの場合は、“IAWRF”ビットがセットされます。

無効メモリ・アクセス検出制御レジスタ（IAWCTL）のフォーマット

アドレス：F0078H　リセット時：00H　R/W

記号	7	6	5	4	3	2	1	0
IAWCTL	IAWEN	0	GRAM1	GRAM0	0	GPORT	GINT	GCSC

IAWEN	無効メモリ・アクセス検出の制御
0	無効メモリ・アクセス検出をディスエーブルする。
1	無効メモリ・アクセス検出をイネーブルする。

図 19　無効メモリ・アクセス保護

5.1.4　I/O ポート SFR 保護

この書き込み保護機能は SFR レジスタへの書き込みを禁止します。書き込みを行ってもエラーは生成されませんが、該当レジスタの内容は変化しません。

データ・ポート・レジスタ（Pxx）には保護を設定できません。

アプリケーションで SFR レジスタを変更する場合または安全上の理由から SFR 設定をリフレッシュする場合は、保護の解除が可能です。

保護される I/O ポート SRF レジスタは以下の通りです。

PMxx、PUxx、PIMxx、POMxx、 PMCxx、ADPC、およびPIOR
Pxxは保護できません。

図 20 に示すように、I/O ポート SFR レジスタは“GPORT”ビットで保護を設定します。

無効メモリ・アクセス検出制御レジスタ（IAWCTL）のフォーマット

アドレス：F0078H　　リセット時：00H　　R/W

記号	7	6	5	4	3	2	1	0
IAWCTL	IAWEN	0	GRAM1	GRAM0	0	GPORT	GINT	GCSC

IAWEN	無効メモリ・アクセス検出の制御
0	無効メモリ・アクセス検出をディスエーブルする。
1	無効メモリ・アクセス検出をイネーブルする。

GRAM1	GRAM0	RAM保護空間
0	0	ディスエーブル。RAMは書き込み可能。
0	1	下位RAMアドレスの128バイト
1	0	下位RAMアドレスの256バイト
1	1	下位RAMアドレスの512バイト

GPORT	ポート・レジスタ保護
0	ディスエーブル。ポート・レジスタは読み出し/書き込み可能。
1	イネーブル。ポート・レジスタは書き込み禁止。読み出し可能。

図 20　I/O ポート SFR のガード保護

5.1.5　割り込み SFR 保護

この書き込み保護機能は割り込み SFR レジスタへの書き込みを禁止します。書き込みを行ってもエラーは発生しませんが、該当レジスタの内容は変化しません。アプリケーションで SFR レジスタを変更する場合または安全上の理由から SFR 設定をリフレッシュする場合は、保護の解除が可能です。

保護される割り込みレジスタは以下の通りです。

IFxx、MKxx、PRxx、EGPx、および EGNx

図 21 に示すように、割り込み SFR レジスタは“GINT”ビットで保護を設定します。

無効メモリ・アクセス検出制御レジスタ（IAWCTL）のフォーマット

アドレス：F0078H　　リセット時：00H　　R/W

記号	7	6	5	4	3	2	1	0
IAWCTL	IAWEN	0	GRAM1	GRAM0	0	GPORT	GINT	GCSC

IAWEN	無効メモリ・アクセス検出の制御
0	無効メモリ・アクセス検出をディスエーブルする。
1	無効メモリ・アクセス検出をイネーブルする。

GRAM1	GRAM0	RAM保護空間
0	0	ディスエーブル。RAMは書き込み可能。
0	1	下位RAMアドレスの128バイト
1	0	下位RAMアドレスの256バイト
1	1	下位RAMアドレスの512バイト

GPORT	ポート・レジスタ保護
0	ディスエーブル。ポート・レジスタは読み出し/書き込み可能。
1	イネーブル。ポート・レジスタは書き込み禁止。読み出し可能。

GINT	割り込みレジスタ保護
0	ディスエーブル。割り込みレジスタは読み出し/書き込み可能。
1	イネーブル。割り込みレジスタは書き込み禁止。読み出し可能。

図 21　割り込み SFR のガード保護

5.1.6　制御レジスタ保護

この書き込み保護機能は制御レジスタへの書き込みを禁止します。書き込みを行ってもエラーは発生しませんが、該当レジスタの内容は変化しません。アプリケーションで制御レジスタを変更する場合または安全上の理由から制御レジスタの設定をリフレッシュする場合は、保護の解除が可能です。

保護される制御レジスタは以下の通りです。

CMC、CSC、OSTS、CKC、PERx、OSMC、LVIM、LVIS、および RPECTL

図 22 に示すように、制御レジスタは“GCSC”ビットで保護を設定します。

無効メモリ・アクセス検出制御レジスタ（IAWCTL）のフォーマット

アドレス：F0078H　　リセット時：00H　　R/W

記号	7	6	5	4	3	2	1	0
IAWCTL	IAWEN	0	GRAM1	GRAM0	0	GPORT	GINT	GCSC

IAWEN	無効メモリ・アクセス検出の制御
0	無効メモリ・アクセス検出をディスエーブルする。
1	無効メモリ・アクセス検出をイネーブルする。

GRAM1	GRAM0	RAM保護空間
0	0	ディスエーブル。RAMは書き込み可能。
0	1	下位RAMアドレスの128バイト
1	0	下位RAMアドレスの256バイト
1	1	下位RAMアドレスの512バイト

GPORT	ポート・レジスタ保護
0	ディスエーブル。ポート・レジスタは読み出し/書き込み可能。
1	イネーブル。ポート・レジスタは書き込み禁止。読み出し可能。

GINT	割り込みレジスタ保護
0	ディスエーブル。割り込みレジスタは読み出し/書き込み可能。
1	イネーブル。割り込みレジスタは書き込み禁止。読み出し可能。

GCSC	チップ・ステート制御レジスタ保護
0	ディスエーブル。チップ・ステート制御レジスタは読み出し/書き込み可能。
1	イネーブル。チップ・ステート制御レジスタは書き込み禁止。読み出し可能。

図 22　無効メモリ・アクセス保護

6. 関連アプリケーションノート

本アプリケーションノートに関連するアプリケーションノートを以下に示します。併せてご参照ください。

- RL78 Family VDE Certified IEC60730/60335 Self Test Library　APPLICATION NOTE（R01AN0749E）

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://www.renesas.com/index.jsp

お問合せ先
http://www.renesas.com/contact/

7. VDE 認定ステータス

ライブラリを構成する各モジュール（アセンブラ・ファイル）の、VDE 認証ステータスを表 21 に示します。

表 21　各モジュールの VDE 認証ステータス

モジュール	Ver.	VDE 認証ステータス
stl_RL78_registertest.asm	3.00	有効（VDE 認証取得モジュールとコード部分が同一）
stl_RL78_registertest_stack.asm	3.00	
stl_RL78_registertest_psw.asm	3.00	
stl_RL78_registertest_cs.asm	3.00	
stl_RL78_registertest_es.asm	3.00	
stl_RL78_registertest_pc.asm	3.00	
stl_RL78_hw_clocktest.asm	3.00	
stl_RL78_sw_clocktest.asm	3.00	
stl_RL78_peripheral_crc.asm	3.00	
stl_RL78_sw_crc.asm	3.00	
stl_RL78_march_c.asm	3.00	
stl_RL78_march_x.asm	3.00	
stl_RL78_march_c_initial.asm	3.00	
stl_RL78_march_x_initial.asm	3.00	
stl_RL78_RamTest_Stacks_c.asm	3.01	
stl_RL78_RamTest_Stacks_x.asm	3.01	
stl_adc.c	3.00	
stl_RL78_GpioTest.asm	3.00	

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2019.03.11	-	初版発行
1.01	2019.03.11	11,23-26,28 他	ROM テスト CPU レジスタ・テストーPC スタック領域テスト、他
1.10	2019.10.09	33, -36, 40	A/D コンバータ ウォッチドッグ 電圧
1.20	2024.05.20	56	表 19. 各モジュールの VED 認証ステータス

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. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   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. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「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。
5. クロックについて
   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。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   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、将来の拡張機能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。
8. 製品間の相違について
   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、フラッシュメモリ、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、ノイズ幅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。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、お客様の責任において、お客様の機器・システムを設計ください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品または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またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 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. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング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。
6. 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）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等

   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、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（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合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、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他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。
10. 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。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. 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。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
12. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
13. 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。
14. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

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直接的、間接的に支配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。

(Rev.5.0-1　2020.10)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。