RL78 ファミリ
RL78 MCU のための IEC60730/60335 セルフテスト・ライブラリ

R01AN7508JJ0101
Rev.1.01
Oct.20.25

要旨

今日、自動電子制御システムが多くの多様なアプリケーションに拡大し続けているため、信頼性と安全性の要件は、システム設計においてますます増大する要素になりつつあります。

たとえば、家電製品向けの IEC60730 安全規格を導入するには、製造業者が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 規格は製品設計のすべての側面をカバーしていますが、Annex H はマイクロコントローラベースの制御システムの設計にとって非常に重要です。これにより、自動電子制御用の 3 つのソフトウェア分類が提供されます。

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

このアプリケーションノートでは、柔軟なサンプルソフトウェアルーチンを使用して、IEC60730 クラス C 安全規格への準拠を支援する方法のガイドラインを示します。これらのルーチンは VDE Test and Certification Institute GmbH によって認定されており、テスト証明書のコピーは、このアプリケーションノートのダウンロードパッケージで入手できます。

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の実行中に使用されます。このドキュメントとそれに付随するサンプルコードは、これを行う方法の例を提供します。

動作確認デバイス

RL78/G23 マイクロコントローラ

RL78/G14 マイクロコントローラ

RL78/F24 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、命令デコード、CPU レジスタ、内部メモリ、ウォッチドッグ・タイマおよびシステム・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。なお、テスト・ハーネスは、統合開発環境の自動生成機能を使用しています。

セルフテスト・ライブラリ関数は、IEC60730Class-C に準じてモジュール別に分かれています。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。

セルフテスト・ライブラリは、安全部マイコンへの実装を想定しています。安全部マイコンをシステムに組み込むことで、故障などの異常からリスク低減を図ることができます。

RL78 のセルフテスト・ライブラリには以下のセルフテスト関数があります。

- 命令デコード

  RL78MCU の全命令に対して全てのアドレッシングモードの組み合わせで正常に動作するかを検証します。

  IEC 60730-1:2013+A1:2015+A2:2020 Annex H - H2.18.5 equivalence class test を参照してください。

- 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 汎用・レジスタ、スタック・ポインタ（SP）、プログラム・ステータス・ワード（PSW）、拡張レジスタ（ES および CS）。

  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 60730-1:2013+A1:2015+A2:2020 Annex H - Table H.11.12.7 1.CPU を参照してください。

  ※ 汎用レジスタとしてマッピングされたメモリ領域に対して ABRAHAM アルゴリズムで実施します。

- 不変メモリ

  MCU の内部 Flash メモリをテストします。

  IEC 60730-1:2013+A1:2015+A2:2020 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 60730-1:2013+A1:2015+A2:2020 Annex H - H.2.19.1 Abraham test を参照してください。

- システム・クロック

  システム・クロックに対して TAU のインプットキャプチャ機能を使用したテスト（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 Reference - IEC 60730-1:2013+A1:2015+A2:2020 Annex H - H2.18.10.1 Frequency monitoring を参照してください。

- CPU/プログラムカウンタ

  プログラムが規定時間内でシーケンスを実行していることを確認するために、CPU とは独立したクロックで動作する内蔵ウォッチドッグ・タイマを用いて確認しています。期待したシーケンス順に実行しているかを監視するための処理をテストハーネスに実装しています。

  IEC 60730-1:2013+A1:2015+A2:2020 Annex H - H2.18.10.3 time-slot and logical monitoring 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 命令デコードテスト

RL78/G23 コアの全命令において全てのアドレッシングモードで正常に動作するかを検証します。実行にはテスト・ハーネス処理からの関数呼び出しを使用しないものと、テスト・ハーネス処理から関数呼び出しを使用する２種類があります。テスト・ハーネス処理からの関数呼び出しを使用しないテストの起動は修正した“startup.asm”モジュールで“C”環境を初期化する前に全命令を検証します。異常を検出した場合は、stl_RL78_InstructionTest_Fail を呼び出します。

テストを行うアドレッシングモード及び命令は、以下の通りです。

アドレッシングモード・命令の詳細については、RL78 マイクロコントローラ ユーザーズ・マニュアル ソフトウエア編（R01US0015）を参照してください。

(1) 命令アドレスのアドレッシング

命令アドレスのアドレッシングには、下記の４種類があります。

- レラティブ・アドレッシング
- イミーディエト・アドレッシング
- テーブル・インダイレクト・アドレッシング
- レジスタ・ダイレクト・アドレッシング
- 処理データ・アドレスに対するアドレッシング

(2) 処理データ・アドレスのアドレッシングには、下記の 9 種類があります。

- インプライド・アドレッシング
- レジスタ・アドレッシング
- ダイレクト・アドレッシング
- ショート・ダイレクト・アドレッシング
- SFR アドレッシング
- レジスタ・インダイレクト・アドレッシング
- ベースト・アドレッシング
- ベースト・インデクスト・アドレッシング
- スタック・アドレッシング

(3) RL78/G23 の命令

RL78-S3 コアには、以下の 81 種類の命令があります。

【8 ビットデータ転送命令】

MOV　XCH　ONEB　CLRB　MOVS

【16 ビットデータ転送命令】

MOVW　XCHGW　ONEW　CLRW

【8 ビット演算命令】

ADD　ADDC　SUB　SUBC　AND　OR　XOR　CMP　CMP0　CMPS

【16 ビット演算命令】

ADDW　SUBW　CMPW

【乗算積和算命令】

MULU　MULUH　MULH　DIVHU　DIVWU　MACHU　MACH

【増減命令】

INC　DEC　INCW　DECW

【シフト命令】

SHR　SHRW　SHL　SHLW　SAR　SARW

【ローテート命令】

ROR　ROL　RORC　ROLC　ROLWC

【ビット操作命令】

MOV1　AND1　OR1　XOR1　SET1　CLR1　NOT1

【コール・リターン命令】

CALL　CALLT　BRK　RET　RETI　RETB

【スタック操作命令】

PUSH　POP

MOVW SP,src

MOVW AX,SP

ADDW SP,#Byte

SUBW SP,#byte

【無条件分岐命令】

BR

【条件付き分岐命令】

BC　BNC　BZ　BNZ　BH　BNH　BT　BF　BTCLR

【条件付きスキップ命令】

SKC　SKNC　SKZ　SKNZ　SKH　SKNH

【CPU 制御命令】

SEL RBn　NOP　EI　DI　HALT　STOP

【注】　BRK,RETB,RETI,HALT,STOP は、命令テストから除外しています。

2.1.1　CPU 命令・テストーソフトウェア API

表 2-1　ソース・ファイル：周期 CPU 命令・テスト

STL ファイル名	ヘッダ・ファイル
stl_RL78_InstructionTest.asm	stl.h
テスト・ハーネス・ファイル名	ヘッダ・ファイル
main.c	

構文	
unsigned char stl_RL78_InstructionTest (void)	
説明	
次に挙げる RL78 の命令以外のテストします。 EI　　DI 上記に挙げた命令は、初期処理時にテストします。 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・ハーネスの制御ファイル（main.c）は、異常検出時には、stl_RL78_InstructionTest_Fail 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	
unsigned char	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 2-2　ソース・ファイル：イニシャル CPU 命令・テスト

STL ファイル名	ヘッダ・ファイル
stl_RL78_InstructionTest.asm	stl_RL78_InstructionTest.inc
テスト・ハーネス・ファイル名	ヘッダ・ファイル
startup.asm	

構文	
stl_RL78_InitialInstructionTest	
説明	
RL78 の全命令をテストします。 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。 正常終了した場合には、stl_RL78_InstructionTest_Pass ジャンプします。 異常検出時には、stl_RL78_InstructionTest_Fail にジャンプします。	
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

2.2　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。

テストを行う CPU レジスタは以下の通りです。

- 汎用レジスタ：レジスタ・バンク 0～3 の AX、HL、DE、BC

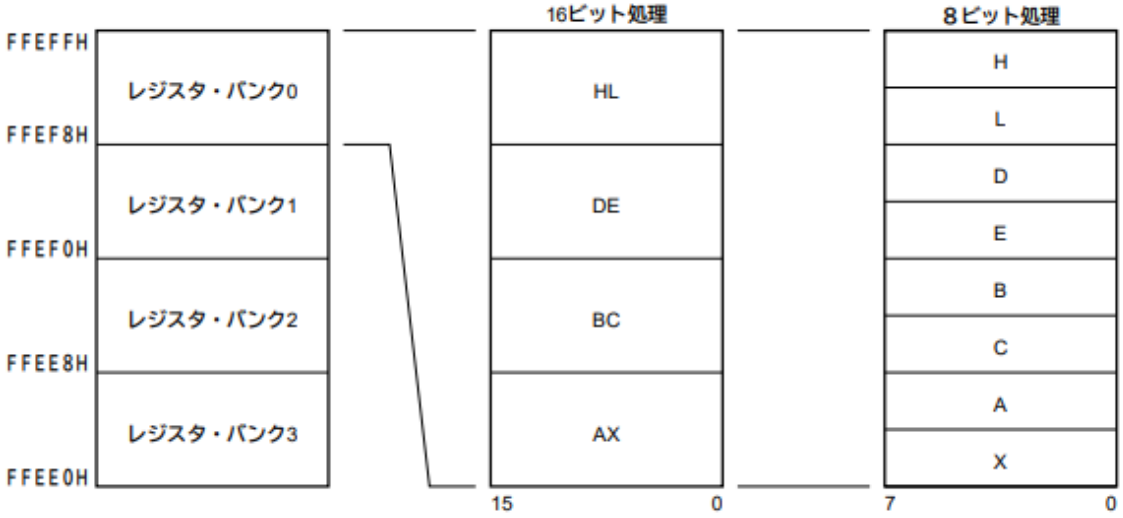

図 2-1 汎用レジスタの構成

- スタック・ポインタ(SP)

略号	15	14	13	12	11	10	9	8
SP	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8

7	6	5	4	3	2	1	0
SP7	SP6	SP5	SP4	SP3	SP2	SP1	0

図 2-2　スタック・ポインタの構成

- プログラム・ステータス・ワード(PSW)

略号	7							0
PSW	IE	Z	RBS1	AC	RBS0	ISP1	ISP0	CY

図 2-3　PSW レジスタの構成

- CS レジスタ

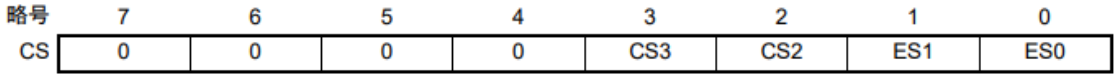

図 2-4　CS の構成

- ES レジスタ

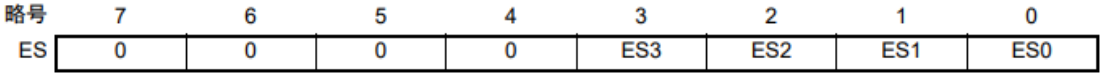

図 2-5　ES の構成

- プログラム・カウンタ(PC)

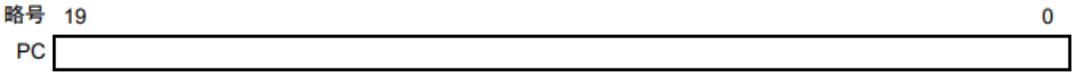

図 2-6　プログラム・カウンタレジスタの構成

2.2.1　CPU レジスタ・テストーソフトウェア API

表 2-3　ソース・ファイル：CPU 汎用レジスタ・テスト

STL ファイル名	ヘッダ・ファイル
stl_RL78_RegisterTest.asm	stl.h
テスト・ハーネス・ファイル名	ヘッダ・ファイル
main.c	

構文	
unsigned char stl_RL78_RegisterTest(unsigned char　Bank)	
説明	
RL78 の汎用レジスタをテストします。 指定したレジスタバンク（バンク 0、1、2、3）のレジスタ AX、HL、DE、BC レジスタとしてマッピングされたメモリ領域に対して ABRAHAM アルゴリズムを使用して実施します。 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、異常検出時には、RegisterTest_Failure を呼び出します。	
入力パラメータ	
unsigned char Bank	試験対象レジスタバンク(0～3)
出力パラメータ	
なし	該当せず
戻り値	
unsigned char	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 2-4　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
異常監視処理ファイル名	ヘッダ・ファイル
r_main.c	

構文	
unsigned char stl_RL78_registertest_psw(void)	
説明	
8 ビットのプログラム・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、異常検出時に RegisterTest_Failure 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	
unsigned char	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 2-5　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
異常監視処理ファイル名	ヘッダ・ファイル
r_main.c	

構文	
unsigned 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、異常検出時に RegisterTest_Failure 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	
unsigned char	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 2-6　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
異常監視処理ファイル名	ヘッダ・ファイル
r_main.c	

構文	
unsigned 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、異常検出時に RegisterTest_Failure 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	
unsigned char	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 2-7　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
異常監視処理ファイル名	ヘッダ・ファイル
r_main.c	

構文	
unsigned 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、異常検出時に RegisterTest_Failure 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	
unsigned char	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 2-8　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	

構文	
unsigned char stl_RL78_RegisterTest_pc(void)	
説明	
以下のテストを実行します。 1.　引数(A レジスタ)に 0x55 を設定します。 2.　引数値の反転した結果を返す関数を呼び出します。 3.　戻り値が 0xAA であることを確認します。 4.　戻り値が戻り番地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・ハーネスの制御ファイル（main.c）は、異常検出時に RegisterTest_Failure 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	
unsigned char	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　不変メモリ・テストー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。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(LSB ファースト)で CRC 値を計算します。

デバッグ時の注意点として、デバッガでソフトウェア・ブレークを設定した場合は、指定したアドレスの命令コードを一時的にブレーク用の命令に書き換えます。そのため、CRC 不一致が発生する場合があります。

2.3.1　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。

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
- LSB ファースト(入力時にビット並びを反転して演算し、演算結果もビット並びを反転して出力)
- 初期値 = 0x0000 または前回の部分 CRC 計算の 16 ビット結果

2.3.2　ハードウェア CRC－ソフトウェア API

表 2-9　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	

構文	
unsigned short stl_RL78_peripheral_crc(unsigned short crc, 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・ファイル（main.c）は、分割された領域の CRC を演算した結果を比較しています。 【注意】 不変メモリの CRC 検査対象範囲及び CRC 値格納番地は、CC-RL で設定した値と同じものを設定して下さい。	
入力パラメータ	
unsigned short crc	CRC 計算の初期値(先頭ブロック時のみ 0 を指定、それ以外は前回の結果)
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）

表 2-10　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 CHECK_CRC_TEST_AREA　lv_CheckCrc;	
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。	
入力パラメータ	
unsigned long m_length;	テストするメモリ範囲（長さ = バイト数）
unsigned long m_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

テスト・ハーネス CRC 演算対象定義

```
typedef struct CRC_RANGE
{
  uint32_t Start;
  uint32_t End;
}CRC_RANGE;
```

ROM テストでは、32K バイト単位に CRC 値を計算し、特定の領域に格納された CRC と一致確認をします。

注：　オンチップデバッガは、ROM の最終から 512 バイトを占有します。そのため、CRC 格納番地は、ROM の最終番地-512-(ブロック数*2)が先頭になります。

```
#define CRC_RANGE_NUM (sizeof(CRC_Ranges)/sizeof(CRC_RANGE))
const CRC_RANGE CRC_Ranges[] =
{
  {0x00000,0x07FFF},                    /*  32K */
  {0x08000,0x0FFFF},                    /*  64K */
  {0x10000,0x17FFF},                    /*  96K ,0x17FFF - 512 - (3 * 2) */
  {0x18000,0x1FFFF},                    /* 128K ,0x1FFFF - 512 - (4 * 2) */
  {0x20000,0x27FFF},                    /* 160K */
  {0x28000,0x2FFFF},                    /* 192K ,0x2FFFF - 512 - (6 * 2) */
  {0x30000,0x37FFF},                    /* 224K */
  {0x38000,0x3FFFF},                    /* 256K ,0x3FFFF - 512 - (8 * 2) */
  {0x40000,0x47FFF},                    /* 288K */
  {0x48000,0x4FFFF},                    /* 320K */
  {0x58000,0x5FFFF},                    /* 384K ,0x5FFFF - 512 - (12 * 2) */
```

```
  {0x60000,0x67FFF},                    /* 416K */
  {0x68000,0x6FFFF},                    /* 448K */
  {0x70000,0x77FFF},                    /* 480K */
  {0x78000,0x7FFFF},                    /* 512K ,0x7FFFF - 512 - (16 * 2) */
  {0x80000,0x87FFF},                    /* 544K */
  {0x88000,0x8FFFF},                    /* 576K */
  {0x90000,0x97FFF},                    /* 608K */
  {0x98000,0x9FFFF},                    /* 640K */
  {0xA0000,0xA7FFF},                    /* 672K */
  {0xA8000,0xAFFFF},                    /* 704K */
  {0xB0000,0xB7FFF},                    /* 736K */
  {0xB8000,0xBFFFF - 512 - (24 * 2)}    /* 768K */
};
```

対象マイコンに合わせて、変更して下さい。

CRC 演算結果格納領域定義

stl.h に定義しています。

#define DEF_ROM_CRC　　(0xBFDD0)

対象マイコンに合わせて、変更して下さい。

2.4　可変メモリーSRAM

ABRAHAM テストは、IEC 60730-1:2013+A1:2015+A2:2020 Annex H – H2.19.1 を満たす RAM テスト手法です。

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。追加されたテスト・モジュール（“stl_RL78_InitialRamTest”）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、ABRAHAM テストについて説明します。

2.4.1　アルゴリズム

（1）ABRAHAM

ABRAHAM アルゴリズムは、全体で 30 種類の処理を実行する 10 のエレメントで構成されます。以下の不具合を検出します。

1. 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。

2. 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。

3. カップリング不具合（CF）

・ セルの値の状態や遷移が他のセルの値を変化させてしまう場合。

4. 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。

⇔ (w0)	Initialize	⇔ ：アドレスの昇順もしくは降順に操作を実施
↓ (r0, w1) ↑ (r1)	Sequence 1	↓ ：アドレスの降順に操作を実施
↓ (r1, w0) ↑ (r0)	Sequence 2	↑ ：アドレスの昇順に操作を実施
↑ (r0，w1) ↓ (r1)	Sequence 3	w0 ： セルに 0 を書き込む
↑ (r1，w0) ↓ (r0)	Sequence 4	w1 ： セルに 1 を書き込む
↓ (r0，w1，w0) ↑ (r0)	Sequence 5	r0 ： セルから期待値 0 を読み出す
↑ (r0，w1，w0) ↑ (r0)	Sequence 6	r1 ： セルから期待値 1 を読み出す
⇔ (w1)	Reset	
↑ (r1，w0，w1) ↑ (r1)	Sequence 7	
↓ (r1，w0，w1) ↑ (r1)	Sequence 8	

2.4.2　可変メモリ・テスト - ソフトウェア API

2.4.2.1　時分割 ABRAHAM

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。時分割 ABRAHAM は、分割されたメモリ領域を一つのメモリ領域と見做して ABRAHAM を実行します。

⇔[x,y] (w0)	Initialize	⇔[x,y] ：領域 x と y の範囲でアドレスの昇順
↓ [x,y](r0, w1) ↑ (r1)	Sequence 1	もしくは降順に操作を実施
↓ [x,y](r1, w0) ↑ (r0)	Sequence 2	↓[x,y] ：領域 x と y の範囲でアドレスの降順に
↑ [x,y](r0，w1) ↓ (r1)	Sequence 3	操作を実施
↑[x,y] (r1，w0) ↓ (r0)	Sequence 4	↑[x,y] ：領域 x と y の範囲でアドレスの昇順に
↓ [x,y] (r0，w1，w0) ↑ (r0)	Sequence 5	操作を実施
↑ [x,y] (r0，w1，w0) ↑ (r0)	Sequence 6	w0 ： セルに 0 を書き込む
⇔ [x,y] (w1)	Reset	w1 ： セルに 1 を書き込む
↑ [x,y] (r1，w0，w1) ↑ (r1)	Sequence 7	r0 ： セルから期待値 0 を読み出す
↓ [x,y] (r1，w0，w1) ↑ (r1)	Sequence 8	r1 ： セルから期待値 1 を読み出す

図 2.7 のように 3 分割した場合の[x,y]は、それぞれ[m1,m2]、[m1,m3]、[m2,m3]になります。

例えば、3 つ領域に分割した場合、周期 t で m1 と m2 の組み合わせ、周期 t+1 では m1 と m3 の組み合わせ、周期 t+2 では m2 と m3 の組み合わせで試験を行うことで、メモリ全体を一括して試験した場合と同等の結果が得られます。

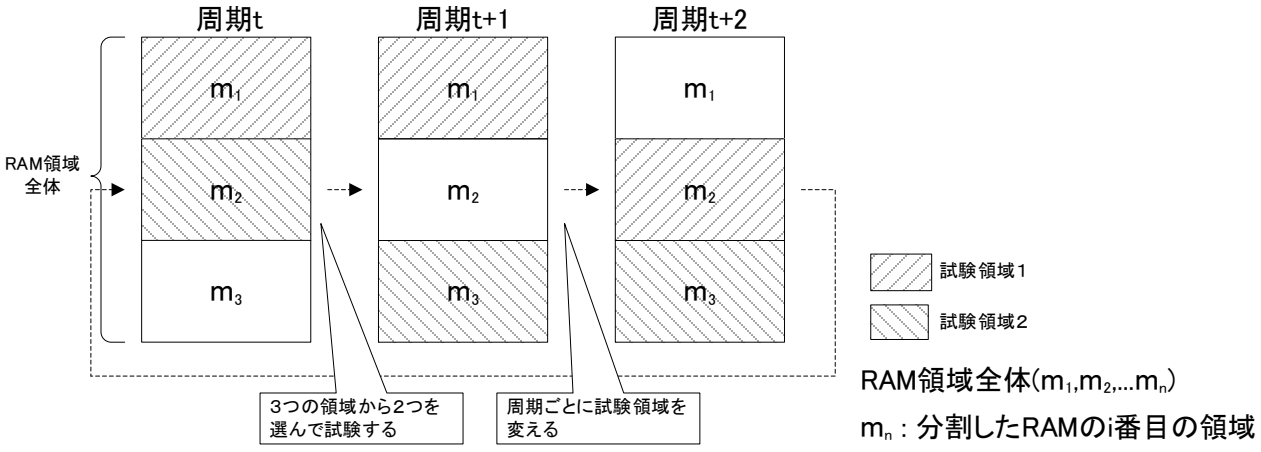

図 2-7　時分割 ABRAHAM の概要

表 2-11　ソース・ファイル：可変メモリテスト

STL ファイル名	ヘッダ・ファイル
stl_RL78_Ram.asm	stl.h
テスト・ハーネス・ファイル名	ヘッダ・ファイル
main.c	

宣言	
unsigned char stl_RL78_RamTest (unsigned char *pRam1, unsigned char *pRam2, unsigned short Size)	
説明	
時分割 ABRAHAM 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。 呼び出し元関数は、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。 事前にテスト対象領域の内容を退避して下さい。テストは破壊的に実施します。 作業領域としてレジスタバンク 1 を使用します。	
入力パラメータ	
unsigned char *pRam1	テストする RAM1 の先頭アドレスのポインタ。
unsigned char *pRam2	テストする RAM2 の先頭アドレスのポインタ。
unsigned short Size	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	
unsigned char	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。

2.4.2.2　イニシャル ABRAHAM

イニシャル ABRAHAM 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。

表 2-12　ソース・ファイル：イニシャル ABRAHAM

STL ファイル名	ヘッダ・ファイル
stl_RL78_InitialmRam.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_InitialRamTest	
説明	
ABRAHAM 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。テスト結果は、stl_RL78_InitialRamTestResult 関数を通じて行われます。 【注】関数“stl_RL78_InitialRamTestResult”はモジュール main.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。

2.5　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）をテストするセルフテスト・モジュールが用意されています。このモジュールは、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。モジュールは、ユーザがヘッダ・ファイル“stl_RL78_hw_clocktest.inc”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、ハードウェア計測の基準値および入力テスト・ポート・ピンを定義します。

2.5.1　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。

- 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）

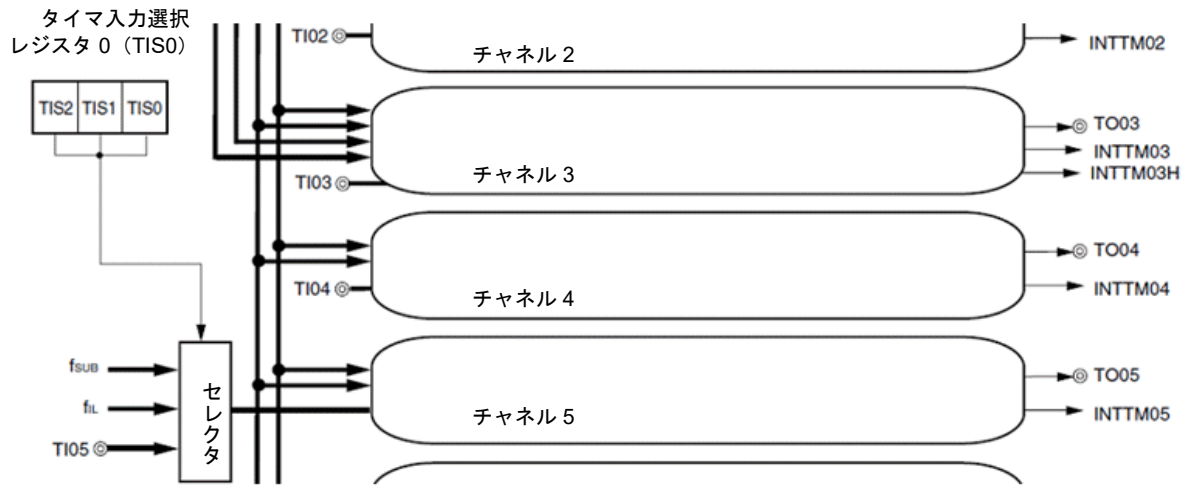

図 2-8　タイマ・アレイ・ユニット・チャネル 5 の構成

【注】RL78/G14,F24 では、チャネル 1 が計測用チャネルです。

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。

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

異常監視処理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 = 32.768 kHz

計算式は 32000000/32768 = 976

【注】　RL78/G14,F24 の内部低速オシレータは、15kHz です。

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。

【タイマ設定（自動生成機能の設定）】

チャネル５：入力パルス間隔測定

ソース入力：fIL

入力信号エッジ：立ち上がりエッジ

割り込み：未使用

【注】RL78/G14,F24 は、チャネル 1 を使用します。

表 2-13　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_RL78_hw_clocktest.inc stl.h
テスト・ハーネス・ファイル名	ヘッダ・ファイル
main.c	

宣言	
void stl_RL78_Init_hw_clocktest (unsigned char Select)	
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・クロックのキャプチャを開始します。	
入力パラメータ	
Select	タイマ・アレイ・ユニットのチャネル 5 入力 0：タイマ入力端子（TI05）の入力信号 5：サブシステム・クロック（fSUB） それ以外：低速オンチップ・オシレータ・クロック（fIL）
出力パラメータ	
なし	該当せず
戻り値	
なし	該当せず

宣言	
unsigned 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_RL78_hw_clocktest.inc）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。	
入力パラメータ	
hwMAXTIME	比較上限値（stl_RL78_hw_clocktest.inc による定義）
hwMINTIME	比較下限値（stl_RL78_hw_clocktest.inc 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_RL78_hw_clocktest.inc による定義）
出力パラメータ	
なし	該当せず
戻り値	
unsigned char	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）。

2.6　内蔵 WDT による監視

プログラムが期待通りに動作しているか、内蔵ウォッチドッグ・タイマで監視しています。
【ウォッチドッグ・タイマ設定（自動生成機能の設定）】
　　ウォッチドッグ・タイマ：使用する
　　HALT/STOP モード時動作：許可
　　オーバーフロー時間：125ms(2^12/fIL)
　　ウィンドウ・オープン期間：50%
　　オーバフロー時間の 75%＋1/4fIL 到達時の割り込み：使用しない

2.6.1　内蔵 WDT による監視

表 2-14　ソース・ファイル内蔵 WDT クリア

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

宣言	
void R_Config_WDT_Restart (void)	
説明	
内蔵 WDT 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

2.7　MCU 異常発生検知

RL78/G23 には、リセット要因を確認するための SFR があります。セルフテスト・ライブラリは、リセット要因を解析し、パワーオンリセット以外の場合、下記の関数を呼び出すよう設定することができます。

要因	オプション設定 (stl_RL78_TestConfig.inc)	呼び出し関数
不正メモリ・アクセス	Illegal_MemoryAccess_enabled	Illegal_MemoryAccess
RAM パリティ・エラー	RAM_Parity_Failure_enabled	RAM_Parity_Failure
ウォッチドッグ・タイマ・オーバ	Watchdog_Test_enabled	Watchdog_Test_Failure
不正命令の実行	Illegal_InstructionExecution_enabled	Illegal_InstructionExecution
電圧異常	Voltage_Test_Reset_enabled	Voltage_Test_Failure_Reset

【注】 RL78/F24 には、RAM パリティエラーはありません。

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+/e²studio 用プロジェクトが提供されます。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。

- 全命令解析・実行テスト
- ABRAHAM アルゴリズムによるイニシャル RAM 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。

- 周期命令・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+ /e²Studio 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+/e²Studio 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_crc 関数）は、「汎用 CRC」に相当します。CC-RL にて CRC 値を組み込む方法は、図 4-3 CS+ /e²Studio へキサ出力オプションを参照して下さい。

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 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。

ハードウェアのペリフェラル・ユニットは、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、その内容を別の領域に再配置してそれに応じてスタック・ポインタを変更しない限り、システムを初期化した後でテストすることはできません。また、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

電源投入時またはリセット時に MCU の全命令解析・実行試験をします。この試験に失敗した場合、メイン関数を呼び出さずに stl_RL78_InstructionTest_Fail にジャンプします。全命令解析・実行試験後に、イニシャル RAM テスト・モジュール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　= 32 KHz）

【注】 RL78/G14,F24 の基準クロックは、15KHz です。

4. 開発環境

- E2-Lite　　オンチップデバッギングエミュレータ
- RL78/G23 Fast Prototyping Board　　RL78/G23（128 ピン LFQFP）
- ツール・チェーン　　CS+　Version 8.1.0.00　CC-RL Version1.13.0
  e^{2}Studio 2024-01.1　　CC-RL Version1.13.0
  - MCU　　R7F100GSN2DFB
- 内部クロック　　32 MHz 高速オンチップオシレータ
  システム・クロック　= 32 MHz
- 低速クロック　　32 kHz 低速オンチップオシレータ

4.1　CS＋の設定

4.1.1　共通オプション

図 4-1　CS+の共通オプション

4.1.2　リンク・オプション

図 4-2　CS+のリンク・オプション

4.1.3　ヘキサ出力オプション

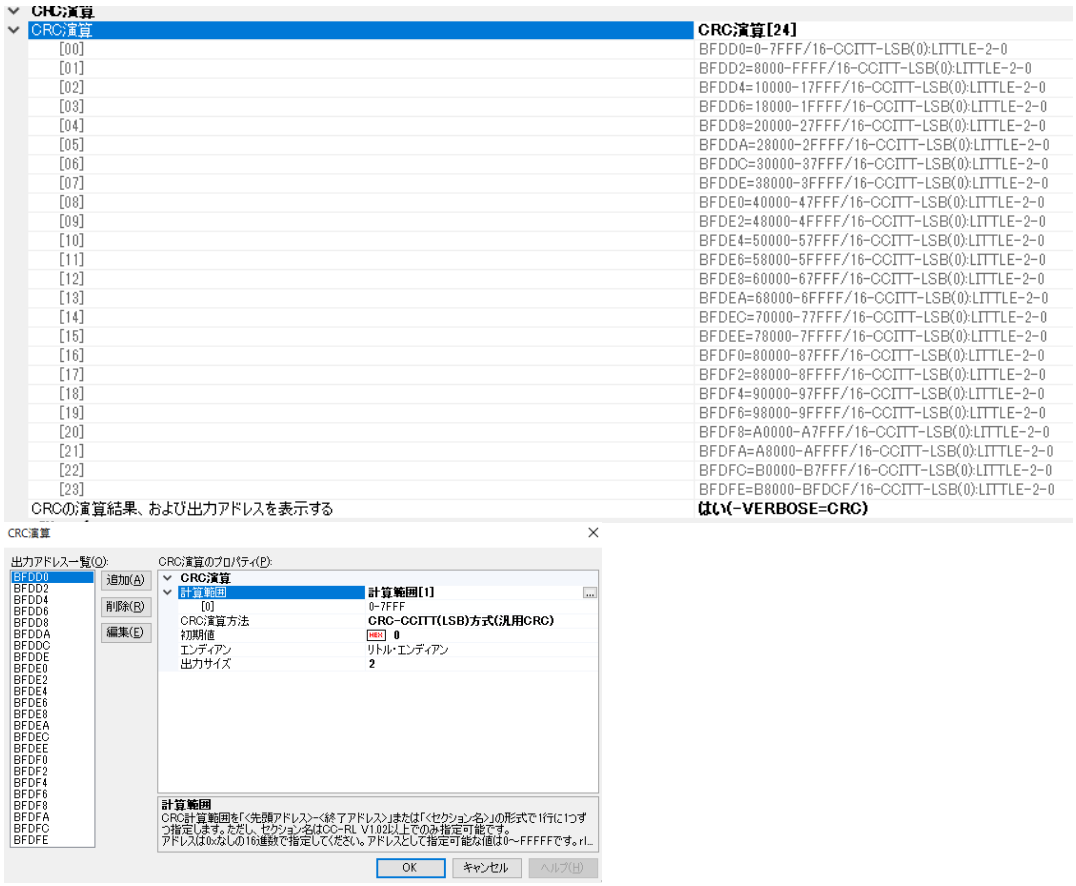

図 4-3　CS+ ヘキサ出力オプション

4.1.4　デバッグツール設定のダウンロードファイル

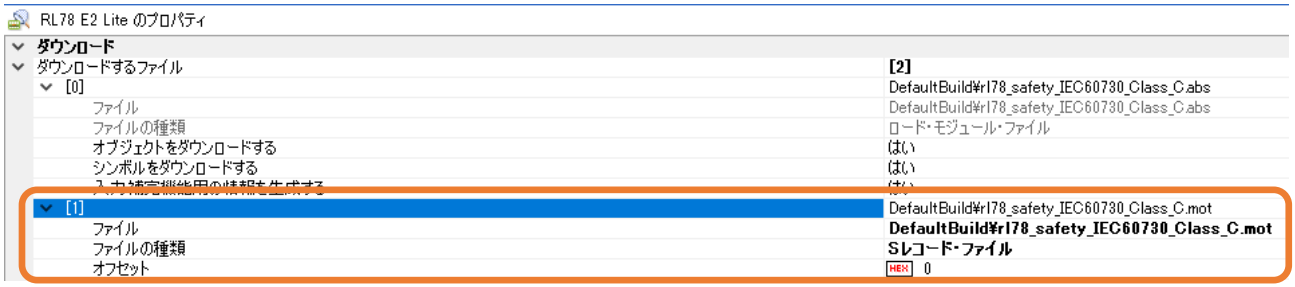

図 4-4　CS+ デバッグツール設定のダウンロードファイル

4.1.5　コード生成(設計ツール)

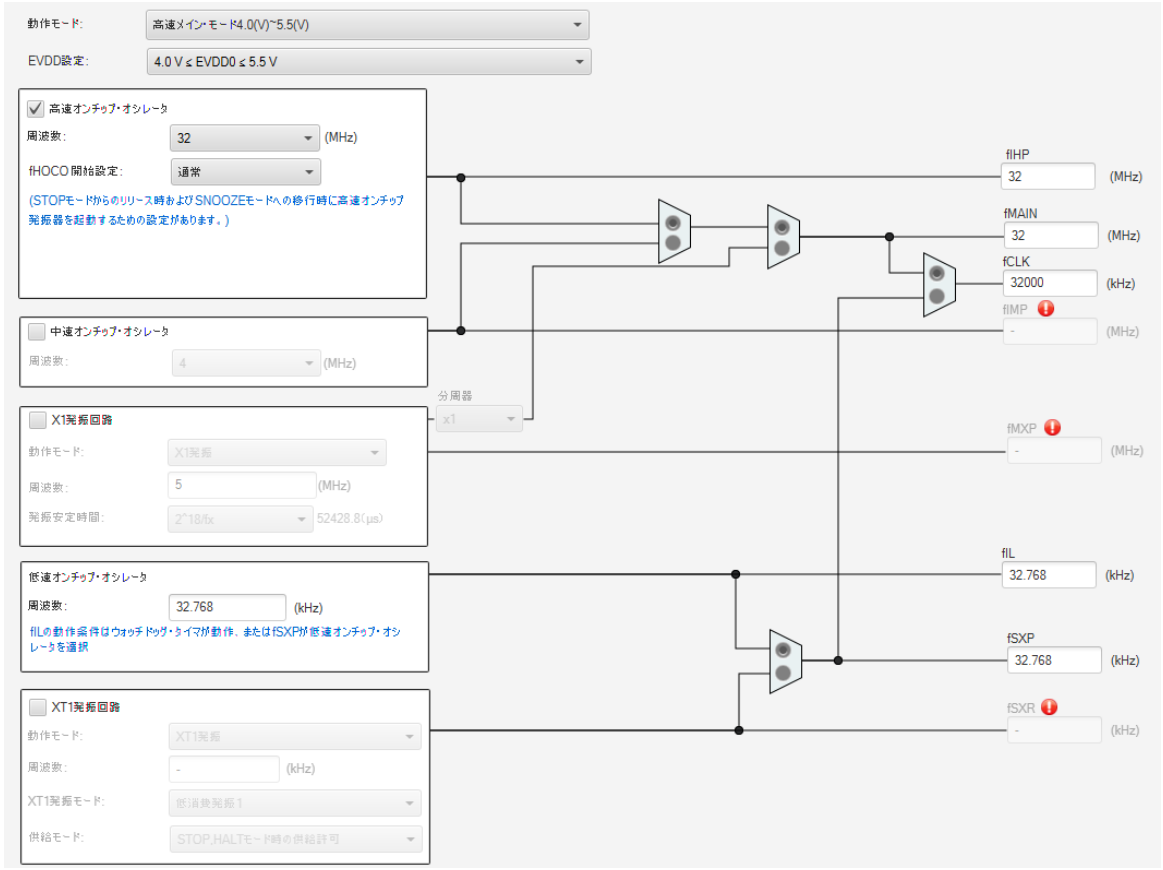

図 4-5　クロック発生回路

図 4-6　システム設定

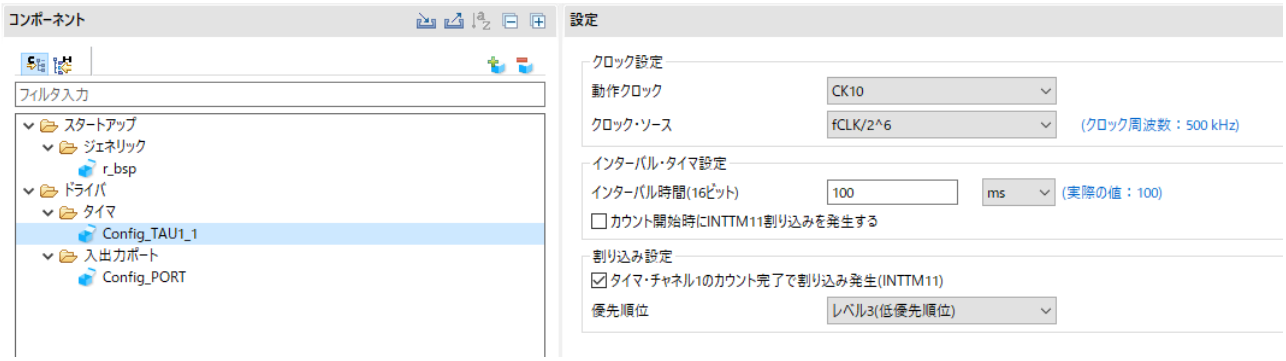

図 4-7　周期タイマ設定

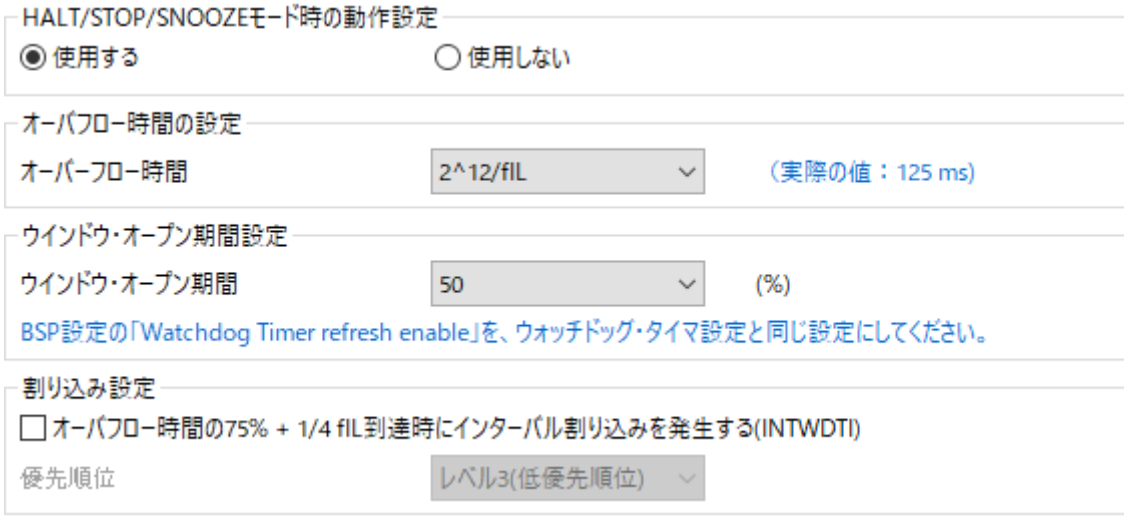

図 4-8　WDT 設定

図 4-9　ポート設定

4.2　e²Studio の設定

4.2.1　Compiler オプション

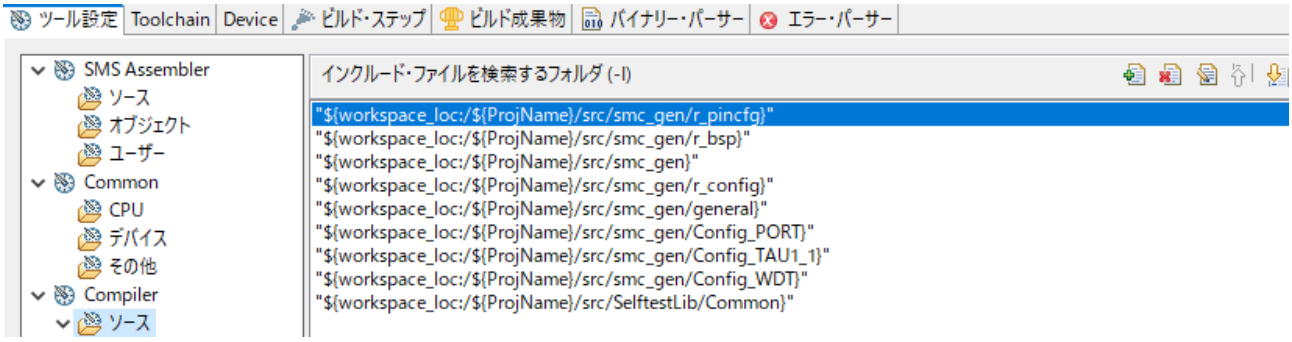

図 4-10　C ソース　インクルードパス

図 4-11　最適化

4.2.2　Assembler オプション

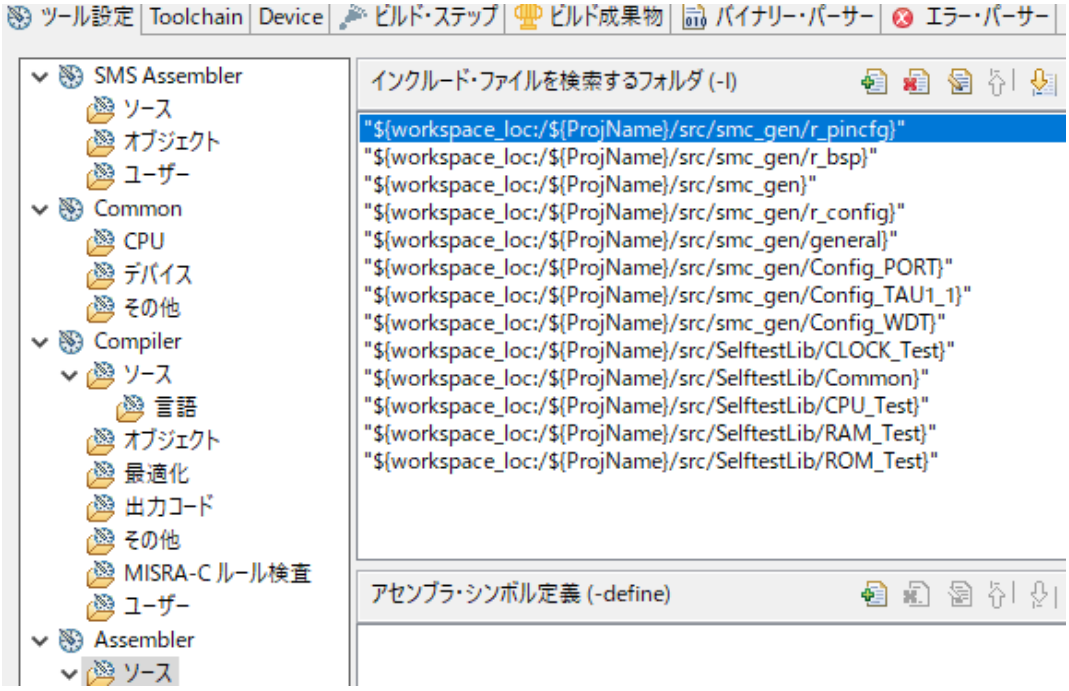

図 4-12　asm ソース　インクルードパス

4.2.3　Linker オプション

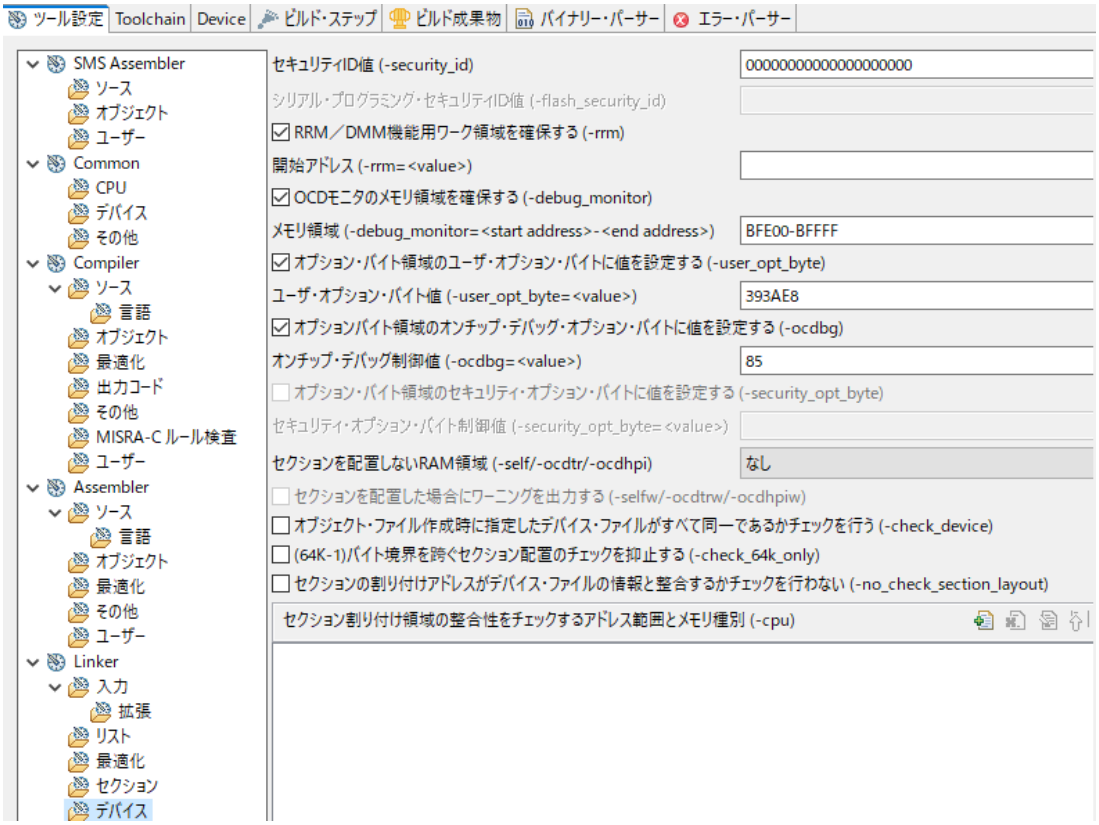

図 4-13　デバイス設定

4.2.4　Converter オプション

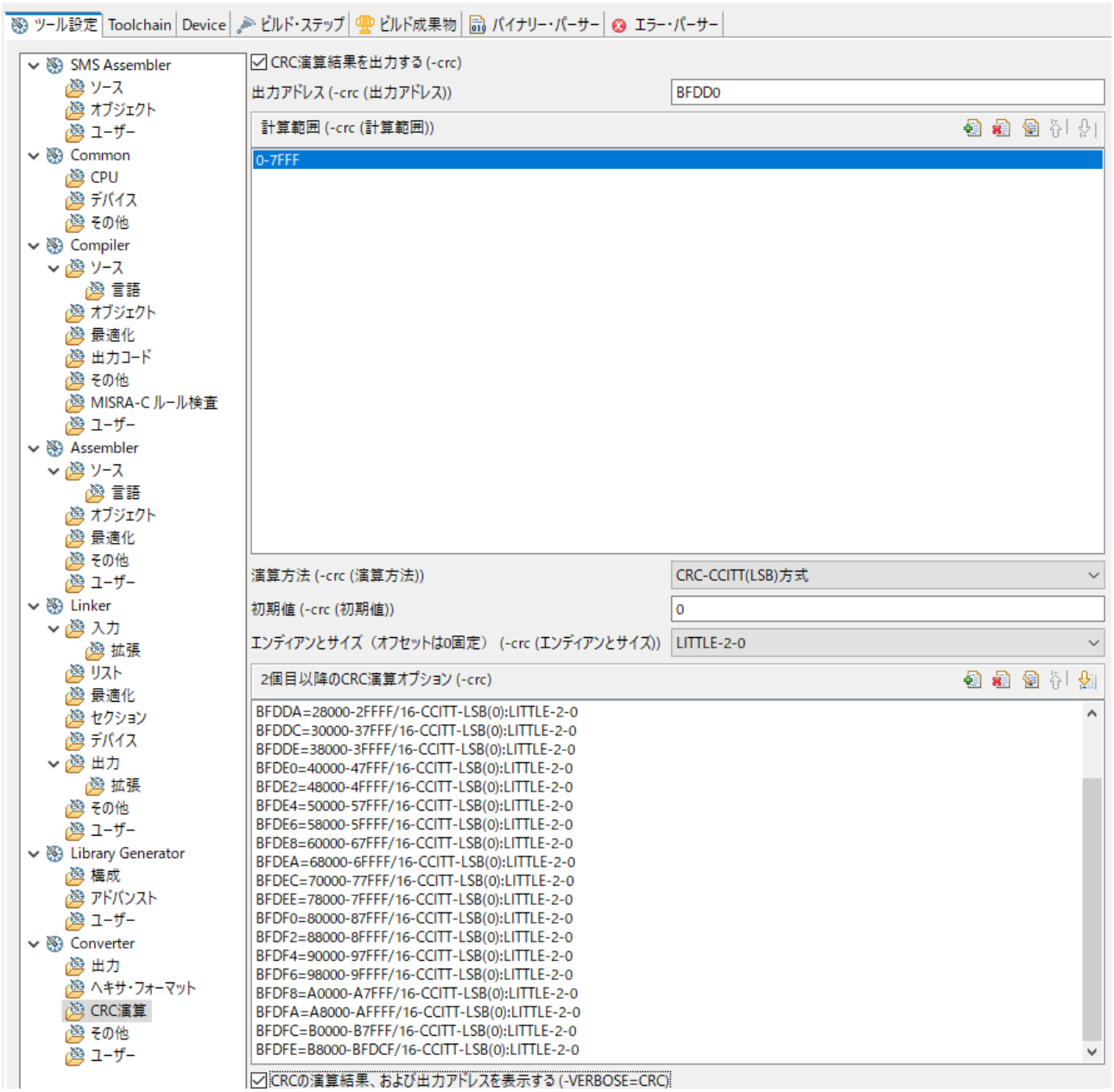

図 4-14　CRC 演算設定

出力先アドレス・計算範囲は、マイコンに合わせて設定して下さい。

4.2.5　デバイスの構成

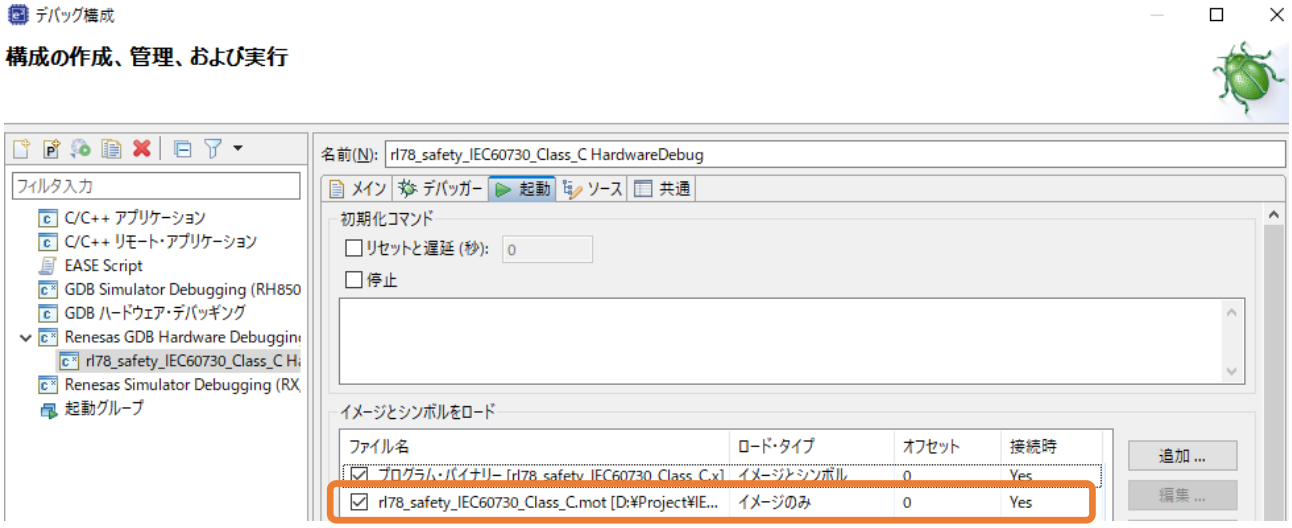

図 4-15　e²studio デバッグツール設定のダウンロードファイル

5. ベンチマーク結果

ライブラリ関数	テスト対象バイト数	処理時間	ROM サイズ
CPU 命令でコード・テスト stl_RL78_InitialInstructionTest	-	600µs	562bytes
CPU 命令でコード・テスト stl_RL78_InstructionTest	-	300µs	12939bytes
CPU 汎用レジスタ・テスト stl_RL78_registertest	-	200µs	1126bytes
CPU レジスタ・テストーPSW stl_RL78_registertest_psw	-	1.343µs	43bytes
CPU レジスタ・テストーSP stl_RL78_registertest_stack	-	1.125µs	50bytes
CPU レジスタ・テストーCS stl_RL78_registertest_cs	-	1.031µs	43bytes
CPU レジスタ・テストーES stl_RL78_registertest_es	-	1.031µs	41bytes
CPU レジスタ・テストーPC stl_RL78_registertest_pc	-	0.857µs	17bytes
ハードウェア CRC stl_RL78_peripheral_crc	1024 バイト	700us	73bytes
システム・RAM テスト stl_RL78_RamTest	32 バイト * 2	1.4ms	1305bytes
イニシャル・RAM テスト stl_RL78_InitialRamTest	508 バイト	10.8ms	980bytes
ハードウェア・クロック・テスト stl_RL78_hw_clocktest	-	56.40µs	136bytes

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（R01AN1062J）

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/

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2025.3.31		初版
1.01	2025.10.20		開発環境に e^{2}studio のバージョン記載

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。