AN204348

FM3 IEC60730 Class B 準拠 セルフテストライブラリ

関連製品ファミリ:	シリーズ名	製品番号
	MB9B500B	MB9BF504NB,MB9BF505NB,MB9BF506NB MB9BF504RB,MB9BF505RB,MB9BF506RB
	MB9B400A	MB9BF404NA,MB9BF405NA,MB9BF406NA MB9BF404RA,MB9BF405RA,MB9BF406RA
	MB9B300B	MB9BF304NB,MB9BF305NB,MB9BF306NB MB9BF304RB,MB9BF305RB,MB9BF306RB
	MB9B100A	MB9BF102NA,MB9BF104NA,MB9BF105NA,MB9BF106NA MB9BF102RA,MB9BF104RA,MB9BF105RA,MB9BF106RA

このアプリケーションノートは提供するライブラリ関数の使用法および実装法について説明します。まず, IEC60730 クラス B の要件を示し, 次にどのようにそれを実現するかを説明します。最後に, テスト関数を実際のシステムに統合する方法を例示します。

Contents

1 はじめに 1
1.1 この文書について 1
1.2 IEC60730 について 1
1.3 MB9B100A/MB9B300B/MB9B400A/MB9B500B シリーズ MCU について 2
1.4 FM3 IEC60730 STL デモプロジェクト 3
2 IEC60730 クラス B の要件 4
3 IEC60730 クラス B STL 概要 5
4 IEC60730 クラス B STL API 6
4.1 CPU レジスタテスト 6
4.2 CPU PC テスト 8
4.3 割込みテスト 9
4.4 クロックテスト 11
4.5 不揮発性メモリのテスト 18
4.6 揮発性メモリテスト 26
4.7 IO テスト 27
4.8 AD テスト 29
5 サンプルプロジェクト 30
5.1 ユーザ設定 30
5.2 プロジェクトの構造 30
5.3 サンプルコード 32
6 STL の API の性能 35
7 参照文書 35
8 付録 36
8.1 フラッシュの CRC コード作成方法 36
9 改訂履歴 40
セールス, ソリューションおよび法律情報 41

1 はじめに

1.1 この文書について

このアプリケーションノートは提供するライブラリ関数の使用法および実装法について説明します。まず, IEC60730 クラス B の要件を示し, 次にどのようにそれを実現するかを説明します。最後に, テスト関数を実際のシステムに統合する方法を例示します。

1.2 IEC60730 について

国際電気標準会議 (IEC) は各国の電子技術委員会 (IEC 国内委員会) により構成された標準化のための国際組織です。国際標準 IEC60730-1 は住宅用自動制御のための IEC 専門技術委員会により起草されました。2007 年以降, 一般家電はシステムの安全性を高めるために IEC60730 に準拠しなければならなくなりました。

IEC60730 の付属書 H は, ハードウェアとソフトウェアの双方に実装される電子制御および組込みシステムに適用されます。マイクロコントローラを使用するシステムは今日の家電の中の典型的な例です。特に IEC60730 の付属書 H は, マイクロコントローラのテストおよび診断方法を詳細に説明しています。

付属書 H は, ソフトウェアに関連する標準項目をクラス A, B, または C に分類しています。

Class A　湿度制御, 照明制御, タイマ等, 機器の安全に関わらないと想定される制御機能。

Class B　洗濯機用の熱電開閉器またはドアロックのように, 機器にソフトウェア障害以外の障害が発生した場合の災害防止を意図したコードを含むソフトウェア。

Class C　密閉式水加熱システム用の熱電開閉器等, 他の保護装置を使用せず災害を防止することを意図したコードを含むソフトウェア。

1.3 MB9B100A/MB9B300B/MB9B400A/MB9B500B シリーズ MCU について

MB9B100A/MB9B300B/MB9B400A/MB9B500B シリーズ MCU は FM3 ファミリの 32 ビット汎用 MCU であり, 業界最先端の ARM Cortex ™ -M3 CPU を備え, サイプレスの信頼性が高く高速で安全な組込みフラッシュ技術を集積しています。本 MCU は 80MHz までの CPU 周波数で動作可能であり, 広い電圧範囲 (2.7-5.5V) で動作します。3.3V および 5V システム双方に対応します。

また, 堅牢な一連の周辺機器機能も含まれており, モーター制御タイマ (MFT), ベースタイマ (PWM, PPG, リロード, PWC タイマとして構成可能), ADC, オンチップメモリ (512K までのフラッシュ, 64K までの SRAM) およびさまざまな種類の通信インタフェース (USB, I2C, SIO, LIN, CAN) 等を備えています。

表 1 に示すように, オンチップメモリの容量はパーツ番号で区別されており, パッケージは LQFP と BGA があります。

表 1. MB9B100A/MB9B300B/MB9B400A/MB9B500B シリーズ製品リスト

製品	フラッシュ	SRAM	パッケージ
MB9BF104NA/RA	256kB	32kB	NA: LQFP-100/BGA-112 RA: LQFP-120
MB9BF105NA/RA	384kB	48kB	NA: LQFP-100/BGA-112 RA: LQFP-120
MB9BF106NA/RA	512kB	64kB	NA: LQFP-100/BGA-112 RA: LQFP-120
MB9BF304NB/RB	256kB	32kB	NA: LQFP-100/BGA-112 RB: LQFP-120
MB9BF305NB/RB	384kB	48kB	NA: LQFP-100/BGA-112 RB: LQFP-120
MB9BF306NB/RB	512kB	64kB	NA: LQFP-100/BGA-112 RB: LQFP-120
MB9BF404NA/RA	256kB	32kB	NA: LQFP-100/BGA-112 RA: LQFP-120
MB9BF405NA/RA	384kB	48kB	NA: LQFP-100/BGA-112 RA: LQFP-120
MB9BF406NA/RA	512kB	64kB	NA: LQFP-100/BGA-112 RA: LQFP-120
MB9BF504NB/RB	256kB	32kB	NA: LQFP-100/BGA-112 RB: LQFP-120
MB9BF505NB/RB	384kB	48kB	NA: LQFP-100/BGA-112 RB: LQFP-120
MB9BF506NB/RB	512kB	64kB	NA: LQFP-100/BGA-112 RB: LQFP-120

1.4 FM3 IEC60730 STL デモプロジェクト

このプロジェクトは FM3 IEC60730 セルフテストライブラリの使用法を説明するためのサンプルプロジェクトです。これは IAR EWARM Workbench V6.21 および Keil µVision V4.20 IDE 上で開発され, それぞれ IAR MB9BF506-SK 評価ボードおよび Keil MCB9B500 評価ボード (Vers.2) 上で評価されています。

注意:

1. IAR EWARM Workbench V6.21 と Keil µVision V4.20 より後のバージョンでこの評価用プロジェクトを開く場合, プロジェクト内の MCU タイプ情報が失われている可能性があります。そのため, 確認し, 失われていた場合は修正してください。
2. IAR EWARM Workbench V6.20 以降を使用するユーザは,「Use CMSIS」オプション (「一般オプション」の「ライブラリ設定」タブ) を選択し,「common」フォルダー (パス: ..¥..¥common)の「core_」で始まるヘッダファイルを消去してください。
3. IAR EWARM Workbench V6.21 より前のバージョンでこの評価用プロジェクトを開く場合, MCU タイプ, あらかじめ含まれているファイル (C/C++コンパイラのプリプロセッサテーブル), icf ファイル (デバッグオプションのリンクテーブル), フラッシュローダーファイル (デバッガーオプションのダウンテーブル) が失われている可能性があります。そのため, それらの設定を確認し, 失われていた場合は修正してください。
4. Keil µVision V4.20 より前のバージョンでこの評価用プロジェクトを開く場合, MCU タイプ, あらかじめ含まれているファイル (C/C++コンパイラのプリプロセッサテーブル), デバッグ設定 (プロジェクト設定のデバッグテーブル) が失われている可能性があります。そのため, それらの設定を確認し, 失われていた場合は修正してください。
5. IAR EWARM Workbench V6.20 以降を使用するユーザは, 2 個のファイル (現在のプロジェクトディレクトリ「cmsis_lo_ver」フォルダ内の「core_cmFunc.h」と「core_cmInstr.h」) を「common」フォルダー (パス: ..¥..¥common) にコピーしてください。

2 IEC60730 クラス B の要件

IEC60730 に定義された仕様においてソフトウェアクラス B またはソフトウェアクラス C に区分される機能を有する制御は，ソフトウェアに関連する障害/エラーを防止し管理するための対策を，安全に関するデータ内およびソフトウェアの安全に関するセグメント内で講じることが求められています。これは，当該ソフトウェアがマイクロコントローラ内外の障害を検出する試験方法を持たなければならないことを意味します。

FM3 IEC60730 セルフテストライブラリ (STL) は，MB9B100A / MB9B300B / MB9B400A / MB9B500B シリーズ MCU に対するソフトウェアクラス B 要件 (標準に示された IEC60730 要件の大部分を含む) を対象にしています。クラス B コントローラについて，試験すべき要素，採用すべき手法および実現すべき定義を，付属書 H の表 H.11.12.7 のまとめを用いて以下の表に示します。

表 2. FM3 IEC60730 STL 試験項目

コンポーネント	障害/エラー	STL で使用する手法	定義	STL での有無
1. CPU 1.1 レジスタ 1.2 プログラムカウンタ	 スタックエラー スタックエラー	 スタティックメモリテスト プログラムシーケンスの論理監視	 H. 2.19.6 H.2.18.10.2	 ○ ○
2. 割込み	割込みせず，または頻繁すぎる割込み	タイムスロット監視	H.2.18.10.4	○
3. クロック	誤った周波数	周波数監視	H.2.18.10.1	○
4. メモリ 4.1. 不揮発性メモリ 4.2. 揮発性メモリ 4.3. アドレス [1]	 全 1 ビット不良 DC 不良 スタックエラー	 冗長性チェック スタティックメモリテスト 冗長性チェック	 H.2.19.3.2 H.2.19.6 -	 ○ ○ ○
5. 内部データパス [2]	スタックエラー	-	-	×
6.外部通信 6.1 データ [3] 6.3 タイミング	 ハミング距離 3 誤った時刻ポイント	 - -	 - -	 × ×
7. 入出力周辺機器 7.1 デジタル I/O 7.2 A/D	 機能エラー 機能エラー	 出力照合 入力比較	 H.2.18.12 H.2.18.8	 ○ ○

注意:

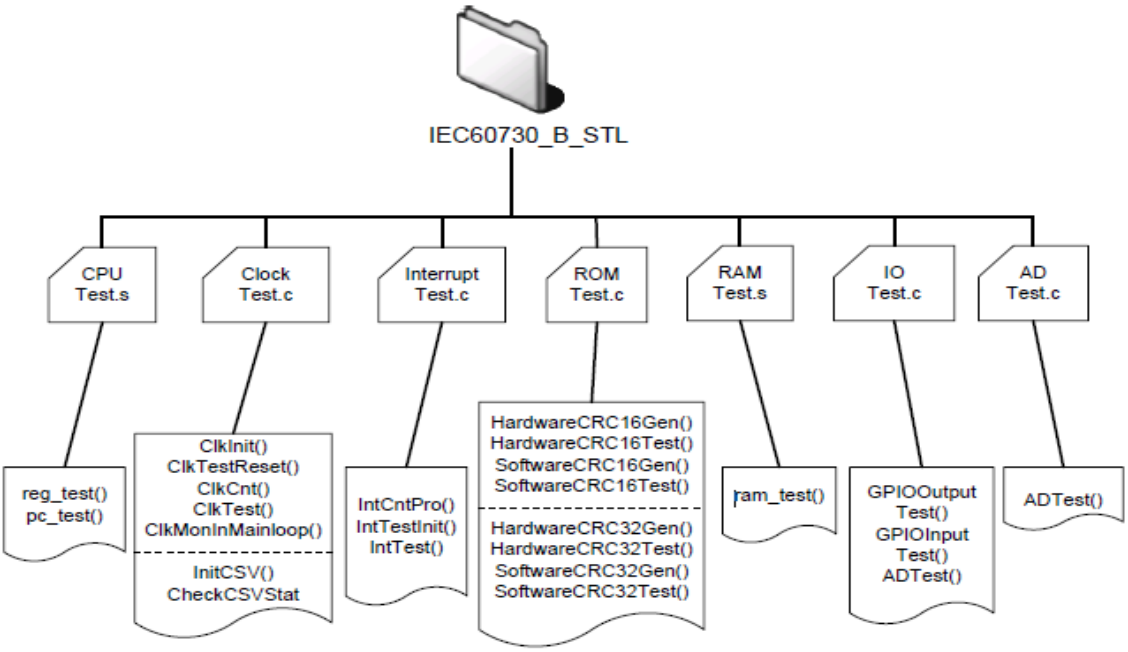
1. アドレス試験は不揮発性および揮発性メモリの試験法により部分的にカバーすることが可能です。たとえば，2 個のセルが同じアドレスにマッピングされるエラーは CRC テストで不揮発性メモリをテストする際に確認できます。
2. 内部データパスは外部メモリ使用時にのみテストされます。
3. 本 STL では外部通信テストは含まれていません。しかし，外部通信データは不揮発性メモリテストと同様の方法でテスト可能です。

3 IEC60730 クラス B STL 概要

次の図に示すように, STL には CPU, 割込み, クロック, メモリおよび入出力周辺モジュールが含まれています。この図は STL 内のファイル構造とソフトウェア API を表しています。STL は C およびアセンブリ言語でコーディングされています。

FM3 IEC60730 STL は ARM と IAR のコンパイラと互換性を持たせています。そこで, STL はコンパイラの違いに応じて 2 種類の CPU test.s および RAM test.s ファイルを提供しています。

図 1. FM3 IEC60730 STL 試験項目



STL はいくつかの独立した機能モジュールで構成され, それらはアプリケーションの要求に従って 1 回または繰り返し実行する必要があります。

1 回だけ実行されるテスト機能はパワーオンセルフテスト (POST) と呼ばれ, システムの初期化時に実行する必要があります。このテストは常に完結しますが破壊的です (イニシャライズが必要)。つまり, すべてのテスト領域をカバーしますが, データはテストの実行後復元されません。PC, レジスタ, ROM/RAM, IO, AD テストはすべて POST です。

繰り返し実行されるテスト機能はビルドインセルフテスト (BIST) と呼ばれ, メインループ内または一定間隔でのタイマ割込みサービスルーチン内で実行されなければなりません。このテストはテストデータを変更せず, プログラム実行中のモニタとして機能します。割込みとクロックは BIST です。

注意:

1. このライブラリは説明されている通りに使用しなければならず, 一部が変更された場合,その部分は新たに検証されなければなりません。
2. 本ライブラリは, このアプリケーションノートに特に記載されていないものを含めて, すべての Cypress Cotex-M3 MCU に転用可能です。
3. 記述を簡単にするため, ファイルと機能名のプリフィックスは省略してあります。
4. この STL には, IAR と Keil IDE のために, CPU および RAM テストに用いる 2 種類のアセンブラファイルが用意されています。
5. クロックとフラッシュテストには 2 種類のテスト方法が使用可能です。

4 IEC60730 クラス B STL API

4.1 CPU レジスタテスト

ARM Cotex-M3 には 19 個のコアレジスタがあり，読み書きが可能です。これらのレジスタをテストする必要があります。

表 3. Cotex-M3 レジスタリスト

レジスタ名	テストするビット
R0-R12	[31:0]
R13 (SP_main, SP_process)[1]	[31:4]
R14 (LR)	[31:0]
APSR[2]	[31:27]
PRIMASK[3]	0
FAULTMASK[4]	0
BASEPRI[5]	[7:4]

注意:

1. ARM Cotex-M3 カーネルには 2 つのスタックポインタがあります。メインスタックポインタ (MSP) とプロセススタックポインタ (PSP) です。ハンドラモードでは MSP，プロセスモードでは MSP または PSP を使用します。R13 は現在の SP を示します。
2. APSR の上位 5 ビットのみが有効です。
3. PRIMASK のビット 0 のみが有効です。
4. FAULTMASK のビット 0 のみが有効です。
5. FM3 MCU の割込み優先レジスタのビット[7:4] により 16 個の割込み優先レベルが設定できます。したがって，BASEPRI のビット[7:4] のみがユーザ割込みのマスクに使用できます。

4.1.1 テストの説明

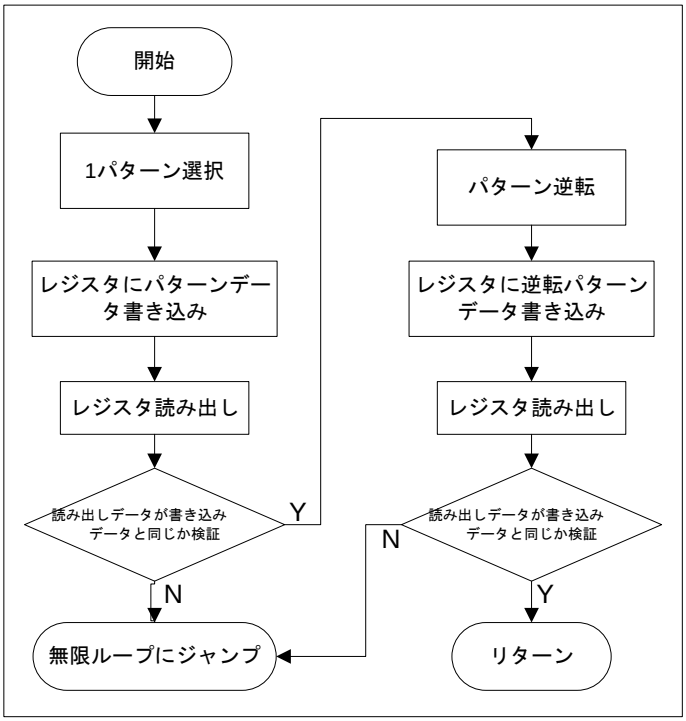
表 H.11.12.7 に示されているように，レジスタは『スタックエラー』のチェックを受けなければなりません。レジスタテストの実行には単純なチェッカーボード法を使用しますが，これはスタックエラー検出に有効な方法です。

このテストは，カーネルレジスタにアクセスする必要があるため，システムが特権モードでリセットされる際にスタートアップファイルで呼び出される必要があります。このテストではレジスタテスト中の割込みが無効になりません。レジスタテストが割り込まれないように，この関数が呼び出される際はアプリケーションで割込みを無効にする必要があります。

レジスタに直接アクセスするために，レジスタテストにはアセンブリ言語が使用されています。このテストは非常に重要なので，レジスタテストでエラーが検出された場合は，プログラムが無限ループに入るように設計されています。

1 個のレジスタをテストするフローチャートを次の図に示します。

図 2. テスト 1 レジスタテスト



4.1.2 API の定義

名称	iec60730_reg_test
パラメータ	なし
リターン	なし

説明:

この API は，R0-R12 (下位: R0-R7，上位: R8-R12)，特殊レジスタ (SP，LR，APSR，PRIMASK，FAULTMASK，BASEPRI) を含むすべてのレジスタをチェッカーボード法でテストするものです。この関数はリセットハンドラで呼び出されなければなりません。

4.2 CPU PC テスト

4.2.1 テストの説明

表 H.11.12.7 に示されているように, PC は『スタックエラー』のチェックを受けなければなりません。PC テストは 8 個のサブルーチンを使用し, 各サブルーチンから得られる PC 値が事前定義値と同じであるかどうかを検証します。

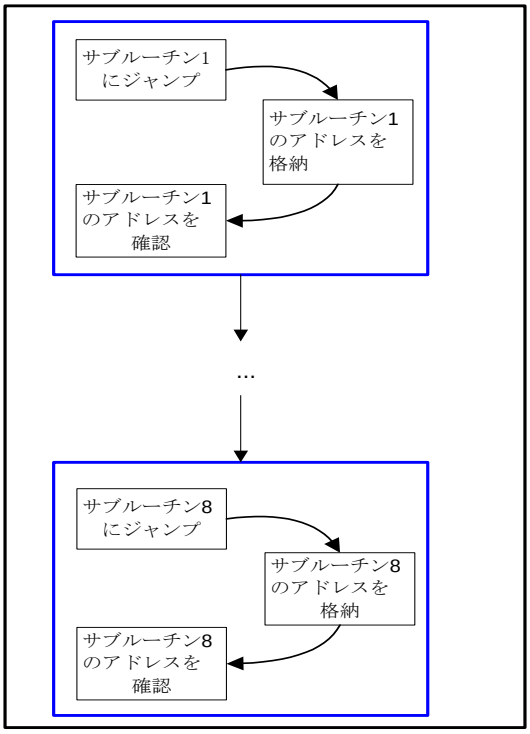
このテストはシステムが特権モードでリセットされる際にスタートアップファイルで呼び出される必要があります。このテストではレジスタテスト中の割込みが無効になりません。レジスタテストが割り込まれないようにこの関数が呼び出される際の割込みの無効設定は, アプリケーションで行う必要があります。

PC レジスタに直接アクセスするために, PC テストにはアセンブリ言語が使用されています。

またこのテストは非常に重要なため, PC テストでエラーが検出された場合は, プログラムが無限ループに入るように設計されています。

PC テストフローを以下に示します。

図 3. PC テストフロー



4.2.2 API の定義

名称	iec60730_pc_test
パラメータ	なし
リターン	なし

説明:

この API は, さまざまな領域でサブルーチンにジャンプしてサブルーチンアドレスを獲得し, 得られたアドレスが正しいかを検証するものです。この関数はリセットハンドラで呼び出されなければなりません。

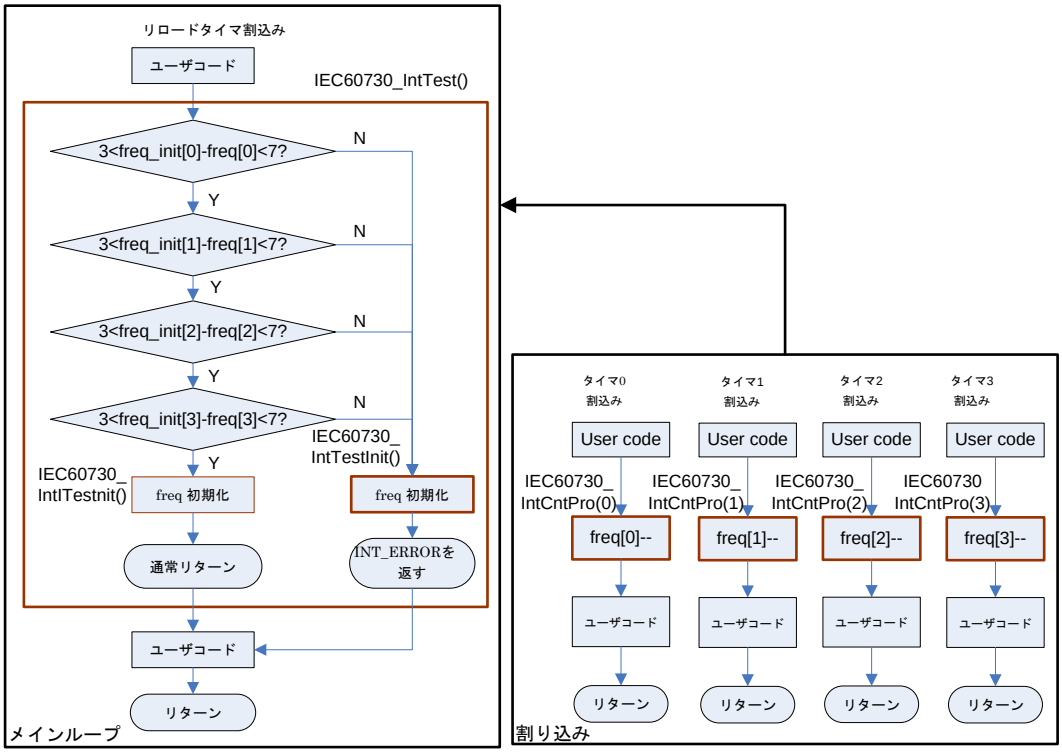
4.3 割込みテスト

4.3.1 テストの説明

クラス B 要件を満たすには，割込みに対して『間違った回数』のチェックを行う必要があります。このテストはシステムに高度に依存するタスクであり，そのため STL は概略処理ができるのみで，ラップアップハンドルを提供することしかできません。つまり，いくつかの特定の割込みが定義されている回数発生したこと (これより少なかったり多かったりしないか) をチェックします。IEC60730_IntTest (割込みテスト関数) が特定の間隔 (タイマまたは線周波数割込みでトリガーされるなど) で呼び出されることを前提としています。監視するすべての特定の割込みハンドラは IEC60730_IntCnt を呼び出すことにより専用のグローバル変数 (Freq) をデクリメントする必要があります。IEC60730_IntTest はその変数を事前定義の上限および下限と比較して，制限を超えている場合は事前設定値に設定してエラーを返します。

例えば，タイマ 0〜3 割込みが 10 秒間に 5 回発生するかどうかを測定します。10 秒タイミングがリロードタイマから得られるとします。タイマ 0-3 の割込み回数範囲を [3, 7] にセットします。

図 4. 割込みテストブロック図



割込みテストとユーザアプリケーションとの依存関係はありません。必要なことはテストしたい割込みに割込みテスト API を追加するだけです。

4.3.2 API の定義

名称	IEC60730_IntTestInit
パラメータ	pFreq: 回数カウンタへのポインタ pFreqLower: 下限回数へのポインタ pFreqUpper: 上限回数へのポインタ pFreqInitial: 回数初期値へのポインタ ArraySize: 割込み情報を格納した配列のサイズ
リターン	なし

説明:

この API は割込みテスト用に str_int_test_par_t 構造体を初期化します。ここには定義済の回数範囲と回数の初期値が含まれています。割込みテスト開始前のシステム初期化時に呼び出される必要があります。

名称	IEC60730_IntCntPro
パラメータ	IntNum: 割込み番号
リターン	なし

説明:

この API は割込み番号で指定される割込みの回数カウンタを減少させます。この関数は特権割込みの中で呼び出される必要があります

名称	IEC60730_IntTest
パラメータ	なし
リターン	0: IEC60730_TEST_NORMAL 1: IEC60730_TEST_FUNC_ERROR

説明:

これは割込みテストのメイン API で, 割込みが時間内に処理されるかを検証します。タイマ割込みまたはメインループ内で一定間隔に呼び出される必要があります。

4.4 クロックテスト

4.4.1 テストの説明

クラス B 要件を満たすには, CPU クロックに対して『間違った周波数』のチェックを行う必要があります。これには, クロックテスト用の標準クロックとして 2 個目の独立したクロックが必要です。本ライブラリにはクロックテスト実行のために 2 つの方法が用意されています。1 番目の方法として, FM3 MCU にはウォッチカウンタが集積されており, これは外部サブクロック (32.768kHz オシレータ) で駆動可能です。このサブクロックを標準クロックとして扱うことができます。2 番目の方法として, FM3 MCU にはクロック故障検出および異常周波数検出機能を持つクロックスーパバイザ (以後 CSV) が集積されています。CSV はクロックテストにも使用できます。

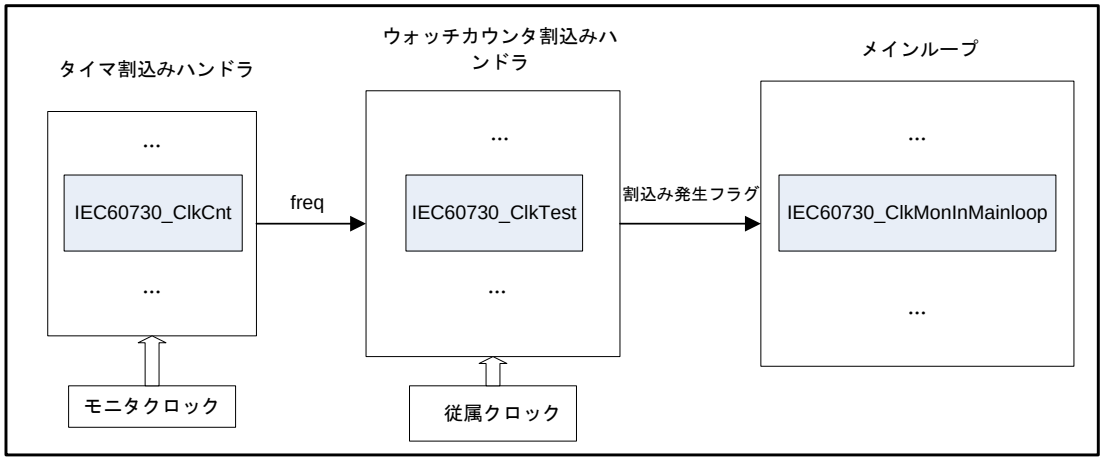
CSV を使用してクロックテストを実行する場合は, IEC60730_user.h ファイルで定義『IEC60730_CLKTEST_USE_CSV』を有効にする必要があります。

ウォッチカウンタを使用してクロックテストを行う

このテストはウォッチカウンタを標準クロックとして使用し, タイマ割込みでカウントされるタイムティックで検証することにより CPU クロック周波数が許容範囲内にあるかどうかをテストします。タイマ割込みのソースクロックは CPU クロックと同一でなければなりません。CPU クロックがサブクロックで駆動されている場合はテストできません。32.768kHz の発振器を正確と想定するからです。

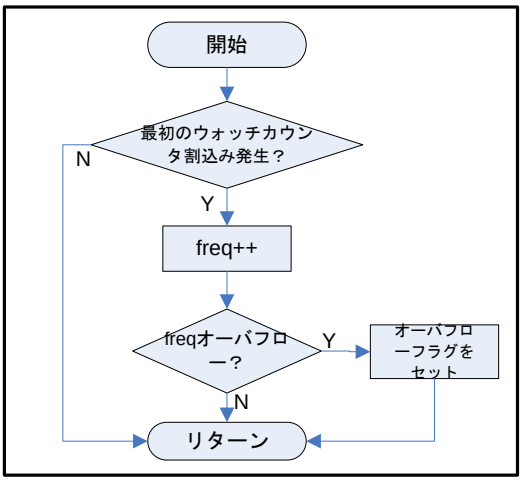
次の図に示すように, テスト関数 IEC60730_ClkCnt, IEC60730_ClkTest および IEC60730_ClkMonMainloop が実装されています。タイマ割込みの発生回数はウォッチカウンタがモニタし, ウォッチカウンタの割込み発生はメインループでチェックされています。

図 5. クロックテストブロック図



API IEC60730_ClkCnt はグローバル変数『freq』をカウントするために使用され, タイマ割込みハンドラ中で呼び出されます。タイマのソースクロックは CPU クロックと同じでなければなりません。IEC60730_ClkCnt のフローチャートを以下に示します。

図 6. クロックカウンタのフローチャート

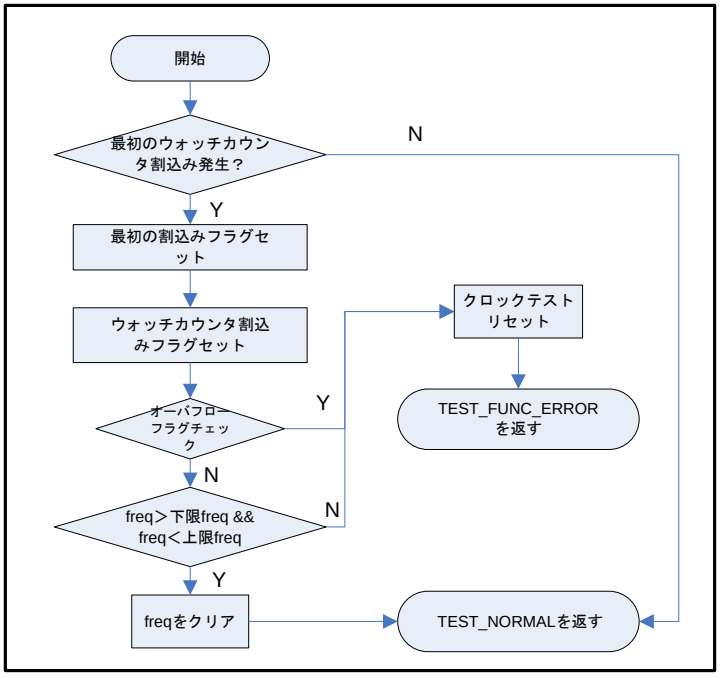


注意:

最初のカウントサイクルが通常サイクルの 2 倍という FM3 MCU のウォッチカウンタの制限があるため，グローバル変数『freq』は最初のウォッチカウンタ割込みが発生するまでカウントします。そのため，最初のウォッチカウンタ割込みは無視しなければなりません。

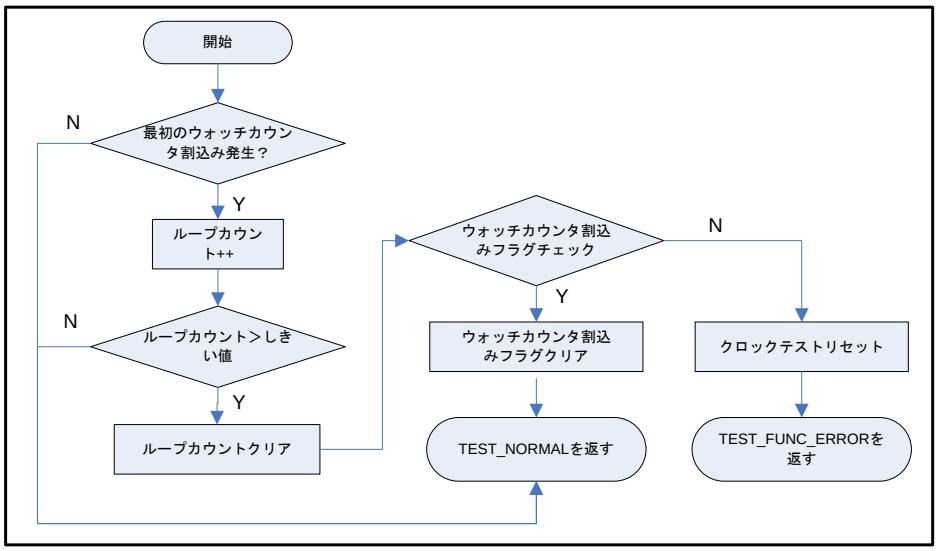
API IEC60730_ClkTest は『freq』が定義されている範囲内にあるかをチェックするもので，ウォッチカウンタ割込みハンドラの中で呼び出されます。

図 7. クロックテストのフローチャート



API IEC60730_ClkMonInMainloop は，ある時間内でのウォッチカウンタ割込みの発生を保証します。この時間は，実際のアプリケーションに従ってユーザが設定したしきい値に依存します。IEC60730_ClkMonMainInloop のフローチャートを次の図に示します。

図 8. クロックメインループモニタのフローチャート



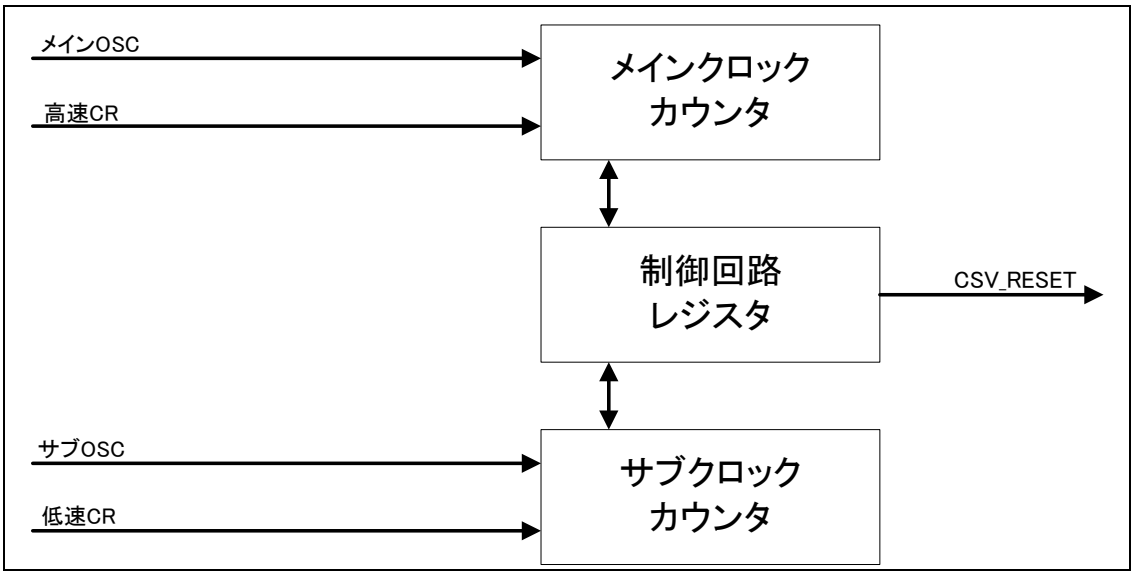
CSV を使ってクロックテストを行う

CSV には 2 つの機能があります。クロック故障検出 (CSV: Clock failure detection by clock Super Visor) と異常周波数検出 (FCS: anomalous Frequency detection by Clock Super visor) です。

クロック故障検出はメインおよびサブクロックをモニタします。指定時間内にモニタクロックの立ち上がりが検出されないと，この機能は発振器の故障と判断しシステムリセット要求を出力します。メインクロックは高速 CR クロックを使用してモニタされており，サブクロックは低速 CR クロックを使用してモニタされています。

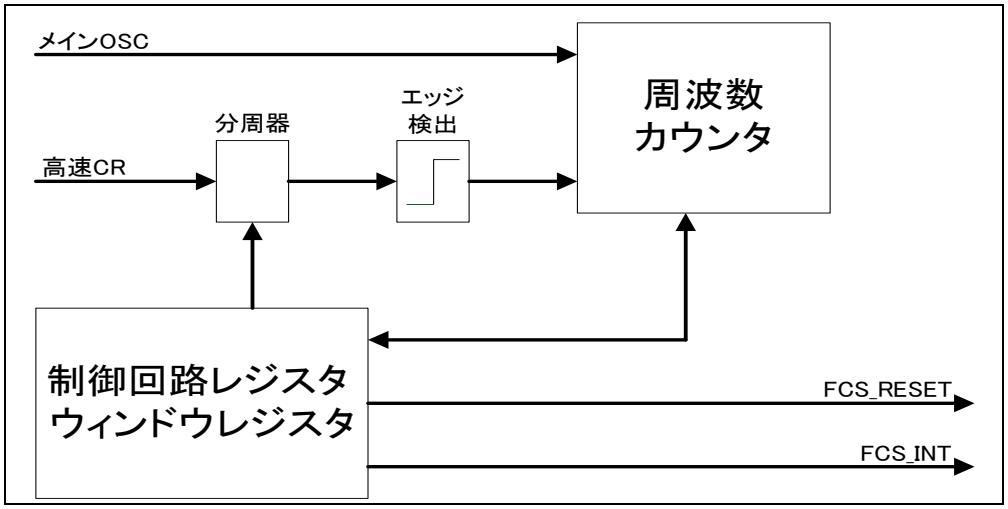
メインクロック用高速 CR の 32 クロック以内，またはサブクロック用低速 CR の 32 クロック以内に立ち上がりが検出されないと，発振器が故障したと判断されます。図 9 にクロック故障検出のブロック図を示します。

図 9. クロック故障検出のブロック図



異常周波数検出はメインクロックをモニタします。高速 CR の分周クロックのエッジと次のエッジとの間の指定された期間で，この機能はメインクロックを使用して内部カウンタをカウントアップします。このカウント値が設定された範囲外に出ると，この機能はメインクロック周波数が異常だと判断し，割込みリクエストまたはシステムリセットリクエストを CPU に出力します。図 10 に異常周波数検出のブロック図を示します。

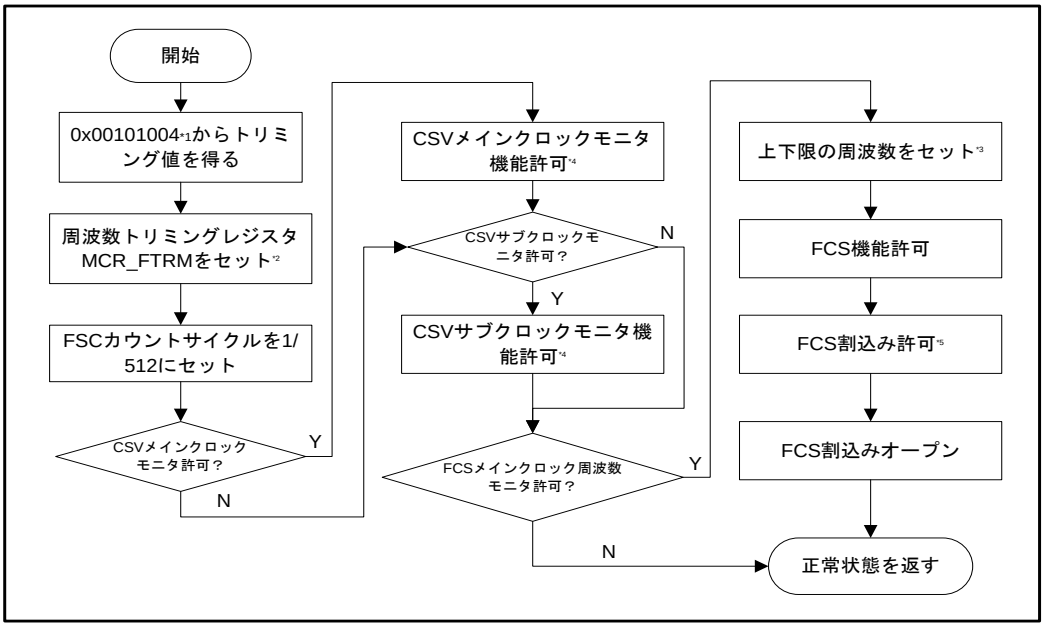
図 10. 異常周波数検出のブロック図



2 つのテスト関数が実装されています。IEC60730_InitCSV と IEC60730_CheckCSVStat です。

API IEC60730_InitCSV はユーザがクロック故障検出および異常周波数検出の禁止/許可を選択できるようにします。これはシステムクロック初期化以前に呼び出されなければなりません。図 11 はそのフローチャートを示します。

図 11. IEC60730_InitCSV フローチャート



注意:

1. 高速 CR トリミングの初期値は工場出荷時にアドレス 0x00101004 に保存されています。
2. アドレス 0x00101004 の CR トリミング値が壊れている場合は, 標準的な値 (0x016B) がトリミングレジスタ MCR_FTRM に書き込まれます。
3. メインクロックの期待精度を設定する際には, 高速 CR 周波数も考慮する必要があります。高速 CR 発振器の精度は 4MHz±3%と考えてください (MB9B100A / MB9B300A / MB9B400A/ MB9B500A シリーズの精度は、データシートにあるように, 25℃で 4MHz±2%です。したがって 4MHz±3%はマージンを取った値としています)。ベース上側および下側カウントは次の公式により計算されます。

   base lower count (+3.0%で動作) = 1/[(freq/512*) × (1 + 0.03)] × freq = 512/1.03= 497

   base upper count (-3.0%で動作) = 1/[(freq/512*) × (1 - 0.03)] × freq = 512/0.97= 528

   本 STL の API (IEC60730_InitCSV) にて精度 5%を設定した場合は, 以下の値が設定されます。

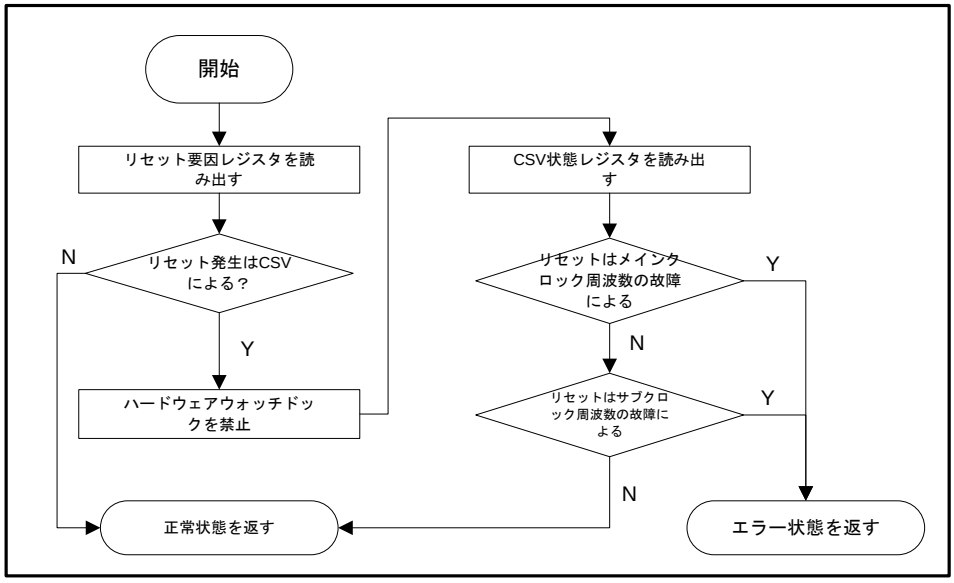
   lower count = 497 × 0.95 = 472

   upper count = 528 × 1.05 = 554
4. CSV 機能を許可後, メインクロック用高速 CR の 32 クロック以内またはサブクロック用低速 CR の 32 クロック以内に立ち上がりエッジが検出されないとリセットがかかります。
5. FCS 機能と FCS 割込みの許可後, メインクロックの周波数が設定範囲外で検出され, また FCS リセットが出力されないように設定されていると, FCS 割込みが発生します。

API IEC60730_CheckCSVStat はクロック故障検出または異常周波数検出が発生したかどうかのチェックに使用します。この API は IEC60730_InitCSV の前に呼び出されなければなりません。

図 12 はそのフローチャートを示しています。

図 12. IEC60730_CheckCSVStat フローチャート



4.4.2 API の定義

ウォッチカウンタを使用してクロックテストを行う場合

名称	IEC60730_ClkCnt
パラメータ	なし
リターン	なし

説明:

この API はクロック周波数のカウントに使用し, タイマ割込みの中で呼び出されなければなりません。

名称	IEC60730_ClkTest
パラメータ	なし
リターン	0: IEC60730_TEST_NORMAL 1: IEC60730_TEST_FUNC_ERROR

説明:

この API は, タイマ割込みでカウントされるタイムティックを検証することにより, CPU クロックの周波数が許容範囲内にあるかどうかをテストします。これはウォッチカウンタ割込みで呼び出す必要があり, その割込みは別個の 32.768kHz クロック (FM3 MCU のサブクロック) で駆動されています。

名称	IEC60730_ClkMonInMainloop
パラメータ	なし
リターン	0: IEC60730_TEST_NORMAL 1: IEC60730_TEST_FUNC_ERROR

説明:

この API はウォッチカウンタ割込みの発生をモニタするために使用します。メインループ中で呼び出す必要があります。

名称	IEC60730_ClkTestReset
パラメータ	なし
リターン	なし

説明:

この API を用いて割込みテスト変数をリセットします。

名称	IEC60730_ClkInit
パラメータ	FreqLower: タイマ割込み最小発生周波数を示します FreqUpper: タイマ割込み最大発生周波数を示します ClkTestThreshold: 閾値を示します
リターン	なし

説明:

この API はクロックテスト開始前のシステム初期化時に呼び出す必要があります。

パラメータ FreqLower と FreqUpper は実際の例に従って設定します。たとえば, 50ms タイマ割込みをモニタするためにウォッチカウンタの 1s インターバルを使用する場合は, FreqLower=18, FreqUpper=22 の値をタイマクロック周波数の限界として設定します。この周波数の標準値は 20 です。

閾値を推定することは重要です。しきい値は少なくとも 1s/メインループの実行時間でなければなりません。

CSV を使ってクロックテストを行う場合

名称	IEC60730_Check CSVStat
パラメータ	pRegRSTStat: リセット要因レジスタからデータを取得します
リターン	なし

説明:

この API はクロック故障検出または異常周波数検出が発生したかどうかのチェックに使用します。パラメータ『pRegRSTStat』にはリセット要因レジスタから読み出したデータのアドレスが格納されます。この API は CSV により起こるリセットを処理するだけです。そうでない場合には正常状態を返します。この API は IEC60730_InitCSV の前に呼び出されなければなりません。

名称	IEC60730_InitCSV
パラメータ	CSV_MCLKMonEn:0: CSV メインクロックのモニタ禁止 1: CSV メインクロックのモニタ許可 CSV_SCLKMonEn:0: CSV サブクロックのモニタを禁止, 1: CSV サブクロックのモニタを許可 FCS_MONInfo: a fcs_mon_info_t structure typedef struct fcs_mon_info { stl_uint8_t FCSMonEn; /* 0: disable FCS function, 1: enable FCS function */ stl_uint8_t MCLKFreqAccuracy; /* input the excepted accuracy of main clock, 5->5%*/ } fcs_mon_info_t;
リターン	0: IEC60730_TEST_NORMAL 2: IEC60730_TEST_PARA_ERROR

説明:

この API は CSV のメイン/サブクロック機能を許可/禁止でき, メインクロック周波数の期待精度を入力します。これはシステムクロック初期化以前に呼び出されなければなりません。

4.5 不揮発性メモリのテスト

FM3 MCU の不揮発性メモリとはオンチップフラッシュメモリのことです。フラッシュのサイズは表 1 に示すように製品に応じて異なります。

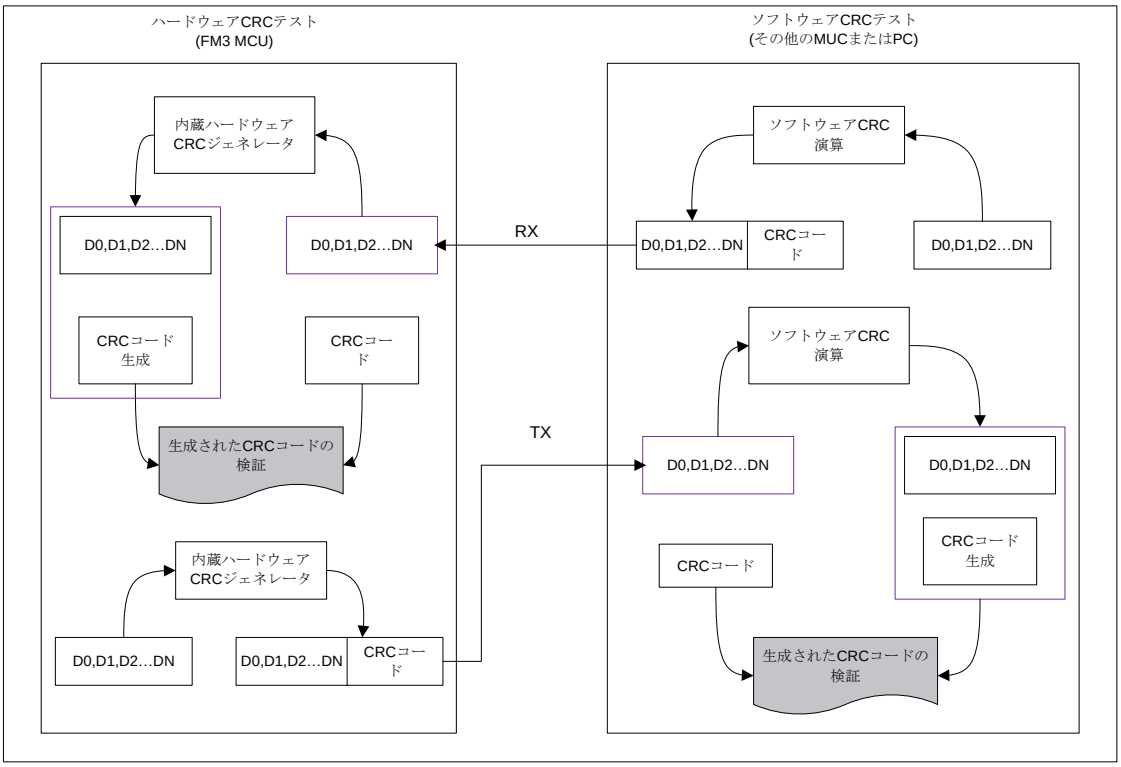
FM3 MCU はオンチップ CRC (巡回冗長検査) モジュールを集積しています。CRC モジュールはエラー検出システムです。CRC コードは入力データ文字列を高次多項式と仮定して，それを定義済みの生成多項式で除算した後の剰余です。通常，データ文字列は送信時に CRC コードがサフィックスとして付けられ，受信したデータは上と同様に生成多項式で除算されます。受信データが割り切れれば正しく受信できたと判断します。オンチップフラッシュメモリのテストにおいても，CRC を利用し，格納されているデータおよびプログラムに故障が発生していないことの検査をします。

CRC モジュールは CCITT CRC16 または IEEE-802.3 CRC32 のどちらかを使用でき，その選択は CRCCR:CRC32 ビットで設定します。このモジュールの生成多項式は，これら 2 つのモード用の数値に固定されています。

- CCITT CRC16 生成多項式: 0x1021 (0x11021 の最上位ビットを省略)
- IEEE-802.3 CRC32 生成多項式: 0x04C11DB7

次の図は，FM3 MCU が他の装置と通信するときに CRC テストを適用する場合を示しています。

図 13. 通信における CRC テスト



4.5.1 テストの説明

クラス B 要件を満たすには，フラッシュテストは『単一ビット不良』のチェックを行う必要があります。このテストは CRC16/32 テストとして実行できます。ハードウェア CRC16/32 テストを実現するためにオンチップの CRC モジュールを使用します。また，ハードウェア CRC と同一の演算を実行するソフトウェア CRC16/32 も提供しています。

フラッシュテストで CRC16 演算を使用する場合は，IEC60730_user.h ファイル中で定義『FLASH_TEST_USE_CRC16』を有効にします。さもなくば CRC32 演算が実行されます。

このテストは，スタートアップ手続き時に実行してコード領域全体をテストするか，または定期的に呼び出してサブブロックをテストすることができます。

フラッシュテストは，プログラム開発時に開発ツールで作成した CRC コードと，テスト時に演算して作成した CRC コードの比較を行います。開発ツールで CRC コードを作成する方法を 8.1 フラッシュの CRC コード作成方法に示します。

注意:

CRC は，ハミング距離 3 の検出を行う H.2.19.4.1 を満たす，外部との通信データのテストに使用することができます。

■ **ハードウェア CRC**

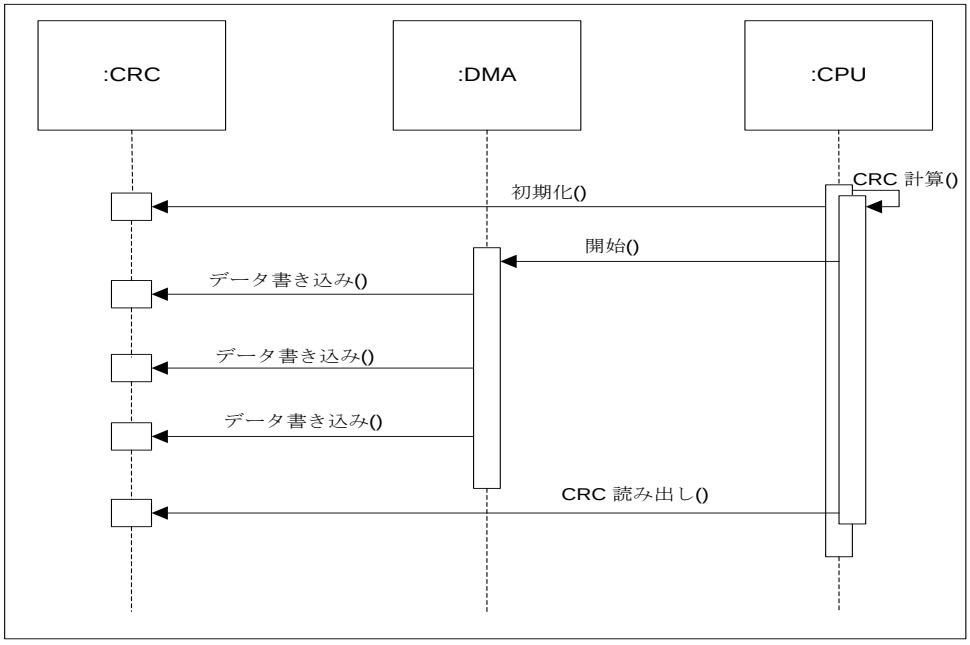
ハードウェア CRC モジュールを使用した CRC コードの生成手順は以下のようになります。

(1) CRC 制御レジスタ CRCCR および初期値レジスタ CRCINIT を初期化する。

(2) 初期値ビット (CRCCR:INIT) に『1』を書き込む。CRCINIT の値が CRC レジスタ CRCR にロードされる。

(3) 入力データレジスタ CRCIN に連続的にデータを書き込む。すると CRC の計算が始まる。CRC コードを得るためには CRC レジスタ (CRCR) を読み出す。

図 14. CRC コード生成のシーケンス



■ ソフトウェア CRC

ソフトウェア CRC16 の演算

CRC テーブル参照法を使用します。CRC コードを生成するためにソフトウェア CRC16 演算は 6 つの手順を実行します。

(1) CRC コードを 0xFFFF で初期化する。

(2) CRC コードを 256 で割って『temp』として保存する。

(3) CRC コードを 8 ビット左シフトする。

(4) CRC コードと、CRC テーブルから取得した値 (テーブルインデックスは『temp』と対象データを XOR した値を使用) の XOR をとり、CRC コードに格納する。

(5) 対象データを 1 バイト分インクリメントする。

(6) 対象データのバイトサイズ分、(2) ～ (5) を繰り返す。

ソフトウェア CRC16 生成コードと CRC16 の表を次の図に示します。

図 15. ソフトウェア CRC16 生成ソースコード

```
stl_uint16_t IEC60730_SoftwareCRC16Gen(stl_uint8_t *pData, stl_uint32_t Size)
{
    stl uint8 t temp;
    stl_uint8_t *p_temp_data = pData;
    stl uint16 t crc = 0xFFFF;
    while(Size-- != 0)
    {
        temp = crc/256;
        crc <<=8;
        crc ^= CRCTable[temp^*p_temp_data];
        p temp data++;
    }
    return crc;
}
```

図 16. CRC16 のテーブル

```
const stl uint16 t crc table[256]={
    0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50A5, 0x60C6, 0x70E7,
    0x8108, 0x9129, 0xA14A, 0xB16B, 0xC18C, 0xD1AD, 0xE1CE, 0xF1EF,
    0x1231, 0x0210, 0x3273, 0x2252, 0x52B5, 0x4294, 0x72F7, 0x62D6,
    0x9339, 0x8318, 0xB37B, 0xA35A, 0xD3BD, 0xC39C, 0xF3FF, 0xE3DE,
    0x2462, 0x3443, 0x0420, 0x1401, 0x64E6, 0x74C7, 0x44A4, 0x5485,
    0xA56A, 0xB54B, 0x8528, 0x9509, 0xE5EE, 0xF5CF, 0xC5AC, 0xD58D,
    0x3653, 0x2672, 0x1611, 0x0630, 0x76D7, 0x66F6, 0x5695, 0x46B4,
    0xB75B, 0xA77A, 0x9719, 0x8738, 0xF7DF, 0xE7FE, 0xD79D, 0xC7BC,
    0x48C4, 0x58E5, 0x6886, 0x78A7, 0x0840, 0x1861, 0x2802, 0x3823,
    0xC9CC, 0xD9ED, 0xE98E, 0xF9AF, 0x8948, 0x9969, 0xA90A, 0xB92B,
    0x5AF5, 0x4AD4, 0x7AB7, 0x6A96, 0x1A71, 0x0A50, 0x3A33, 0x2A12,
    0xDBFD, 0xCBDC, 0xFBBF, 0xEB9E, 0x9B79, 0x8B58, 0xBB3B, 0xAB1A,
    0x6CA6, 0x7C87, 0x4CE4, 0x5CC5, 0x2C22, 0x3C03, 0x0C60, 0x1C41,
    0xEDAE, 0xFD8F, 0xCDEC, 0xDDCD, 0xAD2A, 0xBD0B, 0x8D68, 0x9D49,
    0x7E97, 0x6EB6, 0x5ED5, 0x4EF4, 0x3E13, 0x2E32, 0x1E51, 0x0E70,
    0xFF9F, 0xEFBE, 0xDFDD, 0xCFFC, 0xBF1B, 0xAF3A, 0x9F59, 0x8F78,
    0x9188, 0x81A9, 0xB1CA, 0xA1EB, 0xD10C, 0xC12D, 0xF14E, 0xE16F,
    0x1080, 0x00A1, 0x30C2, 0x20E3, 0x5004, 0x4025, 0x7046, 0x6067,
    0x83B9, 0x9398, 0xA3FB, 0xB3DA, 0xC33D, 0xD31C, 0xE37F, 0xF35E,
    0x02B1, 0x1290, 0x22F3, 0x32D2, 0x4235, 0x5214, 0x6277, 0x7256,
    0xB5EA, 0xA5CB, 0x95A8, 0x8589, 0xF56E, 0xE54F, 0xD52C, 0xC50D,
    0x34E2, 0x24C3, 0x14A0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
    0xA7DB, 0xB7FA, 0x8799, 0x97B8, 0xE75F, 0xF77E, 0xC71D, 0xD73C,
    0x26D3, 0x36F2, 0x0691, 0x16B0, 0x6657, 0x7676, 0x4615, 0x5634,
    0xD94C, 0xC96D, 0xF90E, 0xE92F, 0x99C8, 0x89E9, 0xB98A, 0xA9AB,
    0x5844, 0x4865, 0x7806, 0x6827, 0x18C0, 0x08E1, 0x3882, 0x28A3,
    0xCB7D, 0xDB5C, 0xEB3F, 0xFB1E, 0x8BF9, 0x9BD8, 0xABBB, 0xBB9A,
    0x4A75, 0x5A54, 0x6A37, 0x7A16, 0x0AF1, 0x1AD0, 0x2AB3, 0x3A92,
    0xFD2E, 0xED0F, 0xDD6C, 0xCD4D, 0xBDAA, 0xAD8B, 0x9DE8, 0x8DC9,
    0x7C26, 0x6C07, 0x5C64, 0x4C45, 0x3CA2, 0x2C83, 0x1CE0, 0x0CC1,
    0xEF1F, 0xFF3E, 0xCF5D, 0xDF7C, 0xAF9B, 0xBFBA, 0x8FD9, 0x9FF8,
    0x6E17, 0x7E36, 0x4E55, 0x5E74, 0x2E93, 0x3EB2, 0x0ED1, 0x1EF0
};
```

■ ソフトウェア CRC32 の演算

CRC テーブル参照法を使用します。CRC コードを生成するためにソフトウェア CRC32 演算は 6 つの手順を実行します。

(1) CRC コードを 0xFFFFFFFF で初期化する。

(2) CRC コードを 24 ビット右シフトして『temp』として保存する。

(3) CRC コードを 8 ビット左シフトした値と、CRC テーブルから取得した値 (テーブルインデックスは『temp』と対象データを XOR した値を使用) の XOR をとり、CRC コードに格納する。

(4) 対象データを 1 バイト分インクリメントする。

(5) 対象データのバイトサイズ分、(2) ～ (4) を繰り返す。

(6) 最後に CRC コードをビット反転させる。

ソフトウェア CRC32 生成コードと CRC32 の表を次の図に示します。

図 17. ソフトウェア CRC32 生成ソースコード

```
stl_uint32_t IEC60730_SoftwareCRC32Gen(stl_uint8_t *pData, stl_uint32_t Size)
{
    stl uint8 t temp;
    stl_uint8_t *p_temp_data = pData;
    stl uint32 t crc = 0xFFFFFFFF;
    while(Size--)
    {
        temp=( crc >> 24 );
        crc = ( crc << 8 ) ^ CRCTable[temp^*p temp data];
        p_temp_data++;
    }
    return ~crc;
}
```

図 18. CRC32 の表

```
const stl uint32 t CRCTable[256]={
    0x00000000L,  0x04c11db7L,  0x09823b6eL,  0x0d4326d9L,
    0x130476dcL,  0x17c56b6bL,  0x1a864db2L,  0x1e475005L,
    0x2608edb8L,  0x22c9f00fL,  0x2f8ad6d6L,  0x2b4bcb61L,
    0x350c9b64L,  0x31cd86d3L,  0x3c8ea00aL,  0x384fbdbdL,
    0x4c11db70L,  0x48d0c6c7L,  0x4593e01eL,  0x4152fda9L,
    0x5f15adacL,  0x5bd4b01bL,  0x569796c2L,  0x52568b75L,
    0x6a1936c8L,  0x6ed82b7fL,  0x639b0da6L,  0x675a1011L,
    0x791d4014L,  0x7ddc5da3L,  0x709f7b7aL,  0x745e66cdL,
    0x9823b6e0L,  0x9ce2ab57L,  0x91a18d8eL,  0x95609039L,
    0x8b27c03cL,  0x8fe6dd8bL,  0x82a5fb52L,  0x8664e6e5L,
    0xbe2b5b58L,  0xbaea46efL,  0xb7a96036L,  0xb3687d81L,
    0xad2f2d84L,  0xa9ee3033L,  0xa4ad16eaL,  0xa06c0b5dL,
    0xd4326d90L,  0xd0f37027L,  0xddb056feL,  0xd9714b49L,
    0xc7361b4cL,  0xc3f706fbL,  0xceb42022L,  0xca753d95L,
    0xf23a8028L,  0xf6fb9d9fL,  0xfbb8bb46L,  0xff79a6f1L,
    0xe13ef6f4L,  0xe5ffeb43L,  0xe8bccd9aL,  0xec7dd02dL,
    0x34867077L,  0x30476dc0L,  0x3d044b19L,  0x39c556aeL,
    0x278206abL,  0x23431b1cL,  0x2e003dc5L,  0x2ac12072L,
    0x128e9dcfL,  0x164f8078L,  0x1b0ca6a1L,  0x1fcdbb16L,
    0x018aeb13L,  0x054bf6a4L,  0x0808d07dL,  0x0cc9cdcaL,
    0x7897ab07L,  0x7c56b6b0L,  0x71159069L,  0x75d48ddeL,
    0x6b93dddbL,  0x6f52c06cL,  0x6211e6b5L,  0x66d0fb02L,
    0x5e9f46bfL,  0x5a5e5b08L,  0x571d7dd1L,  0x53dc6066L,
    0x4d9b3063L,  0x495a2dd4L,  0x44190b0dL,  0x40d816baL,
    0xaca5c697L,  0xa864db20L,  0xa527fdf9L,  0xa1e6e04eL,
    0xbfa1b04bL,  0xbb60adfcL,  0xb6238b25L,  0xb2e29692L,
    0x8aad2b2fL,  0x8e6c3698L,  0x832f1041L,  0x87ee0df6L,
    0x99a95df3L,  0x9d684044L,  0x902b669dL,  0x94ea7b2aL,
    0xe0b41de7L,  0xe4750050L,  0xe9362689L,  0xedf73b3eL,
    0xf3b06b3bL,  0xf771768cL,  0xfa325055L,  0xfef34de2L,
    0xc6bcf05fL,  0xc27dede8L,  0xcf3ecb31L,  0xcbffd686L,
    0xd5b88683L,  0xd1799b34L,  0xdc3abdedL,  0xd8fba05aL,
    0x690ce0eeL,  0x6dcdfd59L,  0x608edb80L,  0x644fc637L,
    0x7a089632L,  0x7ec98b85L,  0x738aad5cL,  0x774bb0ebL,
    0x4f040d56L,  0x4bc510e1L,  0x46863638L,  0x42472b8fL,
    0x5c007b8aL,  0x58c1663dL,  0x558240e4L,  0x51435d53L,
    0x251d3b9eL,  0x21dc2629L,  0x2c9f00f0L,  0x285e1d47L,
    0x36194d42L,  0x32d850f5L,  0x3f9b762cL,  0x3b5a6b9bL,
    0x0315d626L,  0x07d4cb91L,  0x0a97ed48L,  0x0e56f0ffL,
    0x1011a0faL,  0x14d0bd4dL,  0x19939b94L,  0x1d528623L,
    0xf12f560eL,  0xf5ee4bb9L,  0xf8ad6d60L,  0xfc6c70d7L,
    0xe22b20d2L,  0xe6ea3d65L,  0xeba91bbcL,  0xef68060bL,
    0xd727bbb6L,  0xd3e6a601L,  0xdea580d8L,  0xda649d6fL,
    0xc423cd6aL,  0xc0e2d0ddL,  0xcda1f604L,  0xc960ebb3L,
    0xbd3e8d7eL,  0xb9ff90c9L,  0xb4bcb610L,  0xb07daba7L,
    0xae3afba2L,  0xaafbe615L,  0xa7b8c0ccL,  0xa379dd7bL,
    0x9b3660c6L,  0x9ff77d71L,  0x92b45ba8L,  0x9675461fL,
    0x8832161aL,  0x8cf30badL,  0x81b02d74L,  0x857130c3L,
    0x5d8a9099L,  0x594b8d2eL,  0x5408abf7L,  0x50c9b640L,
    0x4e8ee645L,  0x4a4ffbf2L,  0x470cdd2bL,  0x43cdc09cL,
    0x7b827d21L,  0x7f436096L,  0x7200464fL,  0x76c15bf8L,
    0x68860bfdL,  0x6c47164aL,  0x61043093L,  0x65c52d24L,
    0x119b4be9L,  0x155a565eL,  0x18197087L,  0x1cd86d30L,
    0x029f3d35L,  0x065e2082L,  0x0b1d065bL,  0x0fdc1becL,
    0x3793a651L,  0x3352bbe6L,  0x3e119d3fL,  0x3ad08088L,
    0x2497d08dL,  0x2056cd3aL,  0x2d15ebe3L,  0x29d4f654L,
    0xc5a92679L,  0xc1683bceL,  0xcc2b1d17L,  0xc8ea00a0L,
    0xd6ad50a5L,  0xd26c4d12L,  0xdf2f6bcbL,  0xdbee767cL,
    0xe3a1cbc1L,  0xe760d676L,  0xea23f0afL,  0xeee2ed18L,
    0xf0a5bd1dL,  0xf464a0aaL,  0xf9278673L,  0xfde69bc4L,
    0x89b8fd09L,  0x8d79e0beL,  0x803ac667L,  0x84fbdbd0L,
    0x9abc8bd5L,  0x9e7d9662L,  0x933eb0bbL,  0x97ffad0cL,
    0xafb010b1L,  0xab710d06L,  0xa6322bdfL,  0xa2f33668L,
    0xbcb4666dL,  0xb8757bdaL,  0xb5365d03L,  0xb1f740b4L
};
```

4.5.2 API の定義

CRC16 を用いたフラッシュテストの実装

名称	IEC60730_HardwareCRC16Gen
パラメータ	pData: テストデータのアドレス Size: データサイズ
リターン	CRC の値

説明:

この API は, 内蔵ハードウェアの CRC モジュールによって CRC16 生成を行います。CCITT CRC16 生成多項式: 0x1021 (0x11021 の最上位ビットを省略)

名称	IEC60730_HardwareCRC16Test
パラメータ	pData: テストデータのアドレス Size: データサイズ Crc: 期待される CRC コード
リターン	0: IEC60730_TEST_NORMAL 1: IEC60730_TEST_FUNC_ERROR

説明:

この API はハードウェア CRC16 テストを実行します。スタートアップ手続きで呼び出して全コード領域をテストするか, コード実行中にサブブロックを定期的にテストします。

名称	IEC60730_SoftwareCRC16Gen
パラメータ	pData: テストデータのアドレス Size: データサイズ
リターン	CRC の値

説明:

この API はソフトウェア CRC 演算により CRC16 生成を実行します。CRC テーブル参照法を使用します。

名称	IEC60730_SoftwareCRC16Test
パラメータ	pData: テストデータのアドレス Size: データサイズ Crc: 期待される CRC コード
リターン	0: IEC60730_TEST_NORMAL 1: IEC60730_TEST_FUNC_ERROR

説明:

この API はソフトウェア CRC16 テストを実行します。このテストは FM3 MCU と通信する他のシステムでも使用可能です。

CRC32 を用いたフラッシュテストの実装

名称	IEC60730_HardwareCRC32Gen
パラメータ	pData: テストデータのアドレス Size: データサイズ
リターン	CRC の値

説明:

この API は, 内部ハードウェア CRC モジュールで CRC32 生成を行います。

CRC32 生成多項式: 0x04C11DB7

名称	IEC60730_HardwareCRC32Test
パラメータ	pData: テストデータのアドレス Size: データサイズ Crc: 期待される CRC コード
リターン	0: IEC60730_TEST_NORMAL 1: IEC60730_TEST_FUNC_ERROR

説明:

この API はハードウェア CRC32 テストを実行します。スタートアップ手続きで呼び出して全コード領域をテストするか, またはコード実行中にサブブロックを定期的にテストします。

名称	IEC60730_SoftwareCRC32Gen
パラメータ	pData: テストデータのアドレス Size: データサイズ
リターン	CRC の値

説明:

この API はソフトウェア CRC 演算により CRC32 生成を行います。テーブル参照法を使用します。

名称	IEC60730_SoftwareCRC32Test
パラメータ	pData: テストデータのアドレス Size: データサイズ Crc: 期待される CRC コード
リターン	0: IEC60730_TEST_NORMAL 1: IEC60730_TEST_FUNC_ERROR

説明:

この API はソフトウェア CRC32 テストを実行します。このテストは FM3 MCU と通信する他のシステムでも実行可能です。

4.6 揮発性メモリテスト

FM3 MCU での揮発性メモリテストとは SRAM テストのことです。SRAM サイズは表 1 に示すように製品に応じて異なります。

4.6.1 テストの説明

クラス B 要件を満たすためには, SRAM テストは『DC 不良』のチェックを行う必要があります。

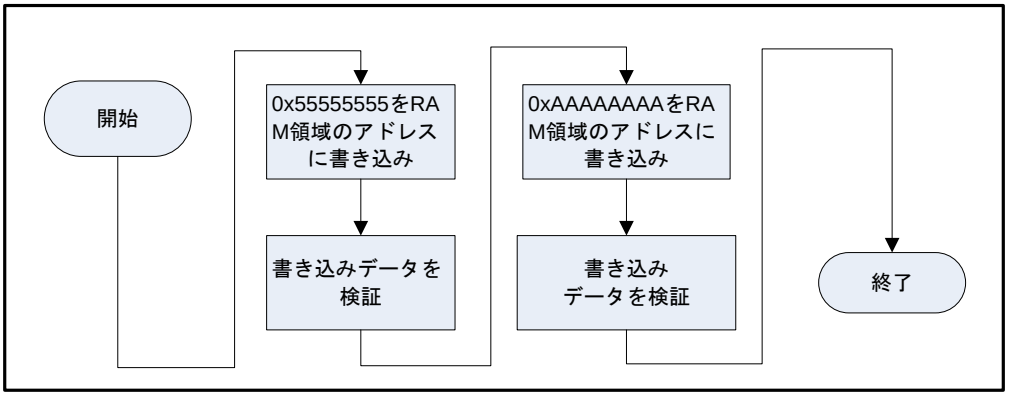
この SRAM テストの実行には単純なチェッカーボード法を使用します。

このテストはスタートアップ手続き時に実行して SRAM 領域全体をテストできます。また, コード実行中にサブブロックを定期的にテストすることも可能です。ただし, このテスト終了時にはデータが破壊さることに注意が必要です。

このテストはすべての RAM 領域が対象なので, このテストでは変数を使用しないことが推奨されています。そのため, レジスタテストの実装にはアセンブリ言語が使用されています。またこのテストは非常に重要なので, RAM テストでエラーが検出されるとプログラムが無限ループに入るように設計されています。

1 ワードのデータをテストする手順を以下に示します。

図 19. チェッカーボード法による 1 ワードのテスト



4.6.2 API の定義

名称	iec60730_ram_test
パラメータ	StartAddr (R0): 開始 RAM アドレス EndAddr (R1): 終了 RAM アドレス
リターン	なし

説明:

この API はチェッカーボード法で SRAM 領域をテストします。すなわち, 『0』と『1』を交互にメモリに書き込み, 書き込まれたデータを読み出すことで書き込みが正しく行われたかを検証します。このテストはスタック故障と DC 故障を検出できます。

このテストはスタートアップ手続きで呼び出すか, 定期的に呼び出すことが可能です。ただしテスト終了後にデータは保存されません。

4.7 IO テスト

FM3 MCU には 8 個までの IO ポート (ポート 0 ～ ポート 8) があり, ポートごとに 16 個のチャネルがあります。これらのポートは, パッケージに対応した構成となっています。

4.7.1 テストの説明

クラス B 要件を満たすためには, GPIO に対して『機能エラー』のチェックを行う必要があります。したがって, 入力, 出力機能の双方に対して機能テストが行われます。IO の方向は表 4. IO 機能の構成に示す IO レジスタで設定できます。GPIO の詳細な設定に関してはペリフェラルマニュアルを参照してください。

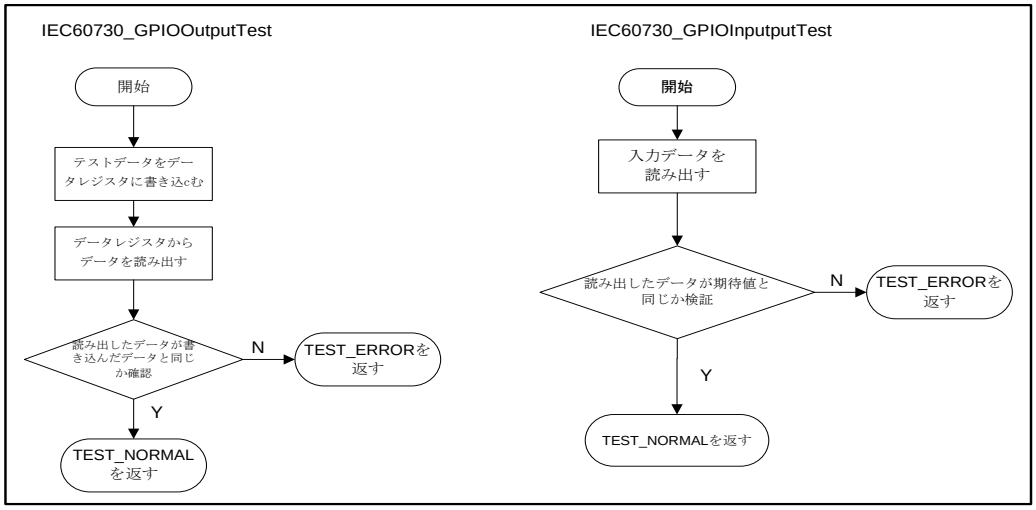
- 入力 IO 設定: ADE=0,PFR=0,DDR=0
- 出力 IO 設定: ADE=0,PFR=0,DDR=1

表 4. IO 機能の構成

I/O ポートの機能		ADE/ SPSR	PFR	DDR	PCR
使用可能なメイン機能	使用可能なサブ機能				
特殊な端子 アナログ入力 USB 水晶	N/A	1	-	-	切断
GPIO 入力端子	周辺機能入力端子	0	0	0	有効
GPIO 出力端子	GPIO 機能入力端子 (FB) 周辺機能入力端子 (FB)			1	切断
周辺機能出力端子	GPIO 機能入力端子 (FB) 周辺機能入力端子 (FB)		1	-	切断
周辺機能双方向端子	GPIO 機能入力端子 (FB) 周辺機能入力端子 (FB)				有効
周辺機能入力端子	GPIO 機能入力端子				有効

IO 入力テストは, 選択した IO 入力値の PDIR に保存された値が期待値と同じかどうかをチェックします。IO 出力テストは, PDOR に保存された出力値が正しいかどうかをチェックします。これらのテストはスタートアップ手続きの中で機能テストとして実行されなければなりません。

図 20. IO 入力/出力テストのフローチャート



4.7.2 API の定義

名称	IEC60730_GPIOOutputTest
パラメータ	Port: ポート番号 Bit: ビット番号 Value: 出力値
リターン	0: IEC60730_TEST_NORMAL 1: IEC60730_TEST_FUNC_ERROR 2: IEC60730_TEST_PARA_ERROR

説明:

この API は, 出力端子のレベルを設定し, 読み出した値が期待値どおりであるかをチェックすることで, GPIO 出力テストを実行します。

名称	IEC60730_GPIOInputTest
パラメータ	Port: ポート番号 Bit: ビット番号 Value: 期待される端子レベル
リターン	0: IEC60730_TEST_NORMAL 1: IEC60730_TEST_FUNC_ERROR 2: IEC60730_TEST_PARA_ERROR

説明:

この API は, 入力ピンから値を読み取り, その値が期待値どおりであるかをチェックすることで, GPIO 入力テストを実行します。

4.8 AD テスト

FM3 MCU には 12 ビットの AD モジュールが集積されています。最大 3 ユニットで 16 チャネルです。

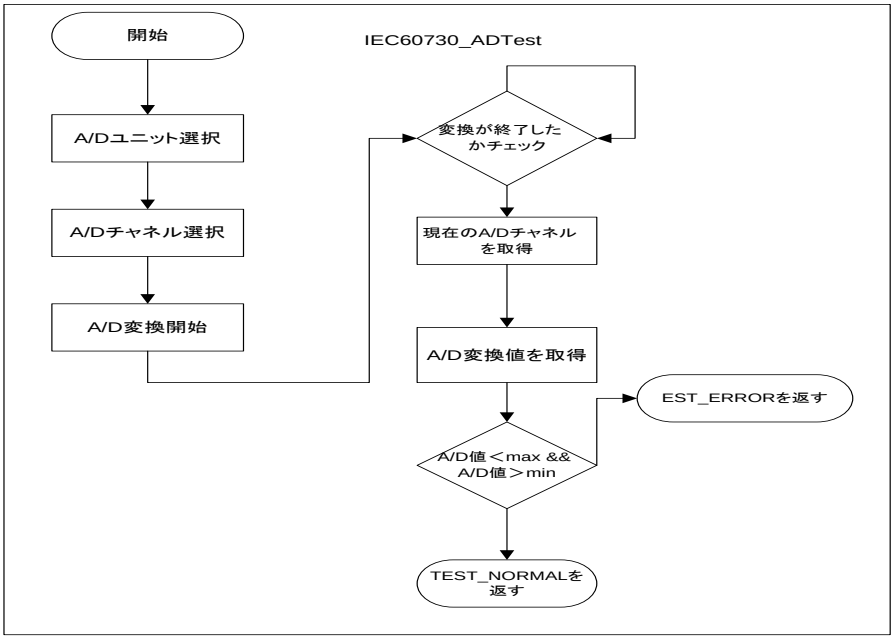
4.8.1 テストの説明

クラス B 要件を満たすためには, AD に対して『機能エラー』のチェックを行う必要があります。

このテストは, 選択された AD チャネルから AD 信号を取得し, AD 変換された値が期待範囲内であるかをチェックします。

スキャンモードを使用し, 複数チャネルを同時にテスト可能です。1 つのチャネルをチェックする AD テストのフローチャートを次の図に示します。

図 21. AD テストのフローチャート



4.8.2 API の定義

名称	IEC60730_ADTest
パラメータ	ADTest_Info: a ad_test_info_t structure typedef struct ad_test_info { uint8_t ADUnit; /* unit num, 8/10 bit A/D -> 0/1/2 */ uint8_t *Ch; /* pointer to AD channel num */ uint8_t ChSize; /* channel size */ uint16_t *ExpLowerValue; /* pointer to expected lower value */ uint16_t *ExpUpperValue; /* pointer to expected upper value */ } ad_test_info_t;
リターン	0: IEC60730_TEST_NORMAL 1: IEC60730_TEST_FUNC_ERROR 2: IEC60730_TEST_PARA_ERROR

説明:

この API は, AD 変換の結果が期待範囲内であるかどうかをチェックする, AD テストが実装されています。このテストはスタートアップ手続き中に実行されなければなりません。

5 サンプルプロジェクト

IAR と Keil IDE に対応した 2 つのデモプロジェクトを示します。この章では IAR MB9BF506-SK 評価ボード (MCU: MB9BF506R) を使用した IAR 用のデモプロジェクトを紹介し, IEC60730 STL を実際のシステムにどのように統合するかを説明します。

5.1 ユーザ設定

ユーザは最初に IEC60730_user.h ファイル内の定義を設定しなければなりません。

5.1.1 定義『MCU_TYPE_MB9BF500』

IAR の MB9BF506R-SK 評価ボードを使用する場合は、定義『MCU_TYPE_MB9BF500』を無効にします。

5.1.2 定義『IEC60730_FLASHTEST_USE_CRC16』

フラッシュテストに CRC16 演算を使用する場合は, この定義を有効にします。フラッシュテストに CRC32 演算を使用する場合は, この定義を無効にします。

このデモプログラムでは CRC16 演算を使用します。

5.1.3 定義『IEC60730_CLKTEST_USE_CSV』

クロックテストの実行に CSV を使用する場合は, この定義を有効にします。そうでない場合は, サブクロックで駆動されるウォッチカウンタを標準タイマとしてクロックテストが実行されます。このデモプログラムでは後者の方法が示されます。

5.2 プロジェクトの構造

クラス B STL ルーチンは 2 つのメインプロセス, つまりスタートアップと定期的セルフテストに分かれています。定期的テストは適用される前にセットアップブロックにより初期化されなければなりません。

5.2.1 スタートアップセルフテスト

PC, レジスタ, SRAM テストはすべてスタートアップセルフテストで, リセットハンドラで呼び出されなければなりません。フラッシュ, AD, IO は, プログラムがメインファンクションにジャンプした後, システムクロックの初期化後にテストできます。AD テストでは, チャネル 13 (ポテンシオメーター入力), チャネル 7, 8, 9 (慣性センサー入力) がテストに使用されます。IO 入力テストでは, ジョイスティック入力ピン P30 (左), P31 (右), P40 (上), P41 (下) がテストに使用されます。

5.2.2 テストの定期的初期化

割込みテストとクロックテストはテスト開始前に初期化されなければなりません。

■ 割込みテスト初期化

デュアルタイム割込みをリロードタイマ 0 ～3 のモニタに使用します。初期化設定パラメータを次の表に示します。

表 5. 割込みテストの初期値

割込み名称	割込み間隔	デュアルタイマ割込み	標準度数	定義済み範囲
リロードタイマ 0	2.5ms	25ms	10	[8,12]
リロードタイマ 1	1ms	25ms	25	[22, 28]
リロードタイマ 2	500us	25ms	50	[45,55]
リロードタイマ 3	250us	25ms	100	[95,105]

■ クロックテストの初期化

CPU クロックは HCLK で，このシステムのデュアルタイマのソースクロックは PCLK0 (HCLK/2) に設定されています。したがってウォッチカウンタにより，デュアルタイマのソースクロックを CPU クロックの代わりに間接的にテストすることができます。

ウォッチカウンタの割込み間隔は 1s，デュアルタイマの割込み間隔は 25ms になっているので，デュアルタイマの標準度数は 40 であり許容範囲は 45 から 55 の間に設定されています。メインループの実行に 10 サイクルかかると仮定します。メインループの最小実行時間は 1/8000000s となるので，しきい値を 10000000 に設定します。

5.2.3 定期的なテスト

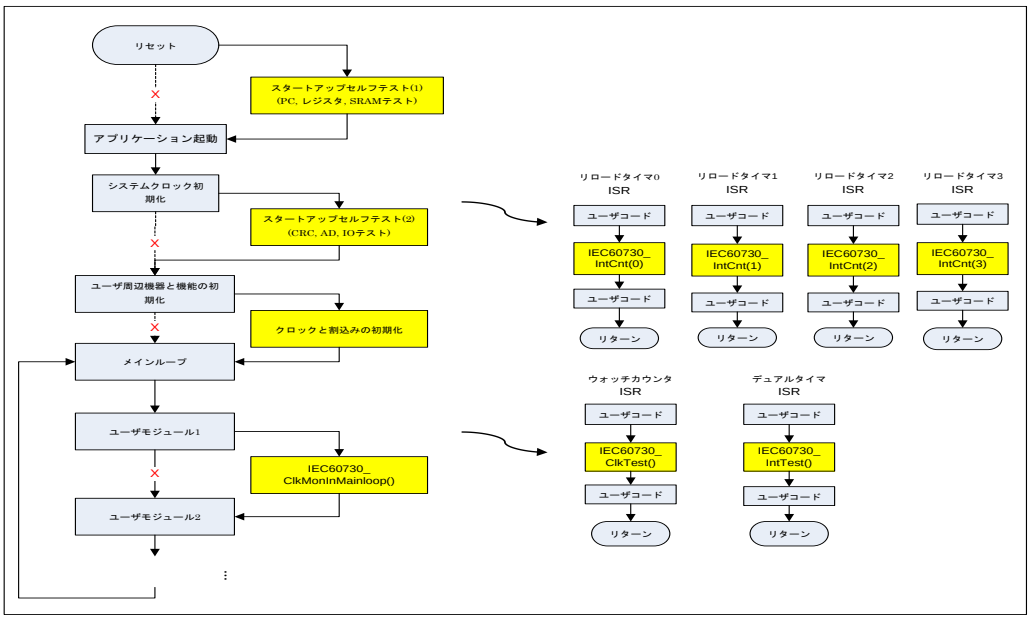
割込みおよびクロックテストはコード実行中に定期的に実行されなければなりません。

IEC60730_IntTest をデュアルタイマ割込みに統合し，IEC60730_IntCnt をそれぞれのリロードタイマ割込みに統合します。

IEC60730_ClkTest をウォッチカウンタ割込みに，IEC60730_ClkCnt をデュアルタイマ割込みに，IEC60730_ClkMonInMainloop をメインループにそれぞれ統合します。

図 22 にクラス B ソフトウェアパッケージをこのアプリケーションソフトウェアにどのように統合するかについての基本原則を示します。

図 22. プロジェクトの構造



5.3 サンプルコード

5.3.1 スタートアップファイル

- リセットハンドラ

図 23. リセットハンドラのサンプルコード

```
Reset Handler
                bl iec60730_reg_test  ; after reset, test register first
                bl iec60730_pc_test   ; test pc
                ldr r0, =0x20000000    ; set RAM start address
                ldr r1, =0x20007fff    ; set RAM end address
                bl iec60730_ram_test  ; test all D-RAM area
```

5.3.2 メインファイル

- メインファンクション

図 24. メインファンクションのサンプルコード

```
int32_t main(void)
{
  int32 t cntr = 0;
  uint32 t hw crc,sw crc;
  uint8 t bit;
  uint8 t a[10] = {0x00,0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88,0x99};
  uint8_t ch[4] = {7,8,9,13};       /* ch7: X-axes input */
                                    /* ch8: Y-axes input */
                                    /* ch9: Z-axes input */
                                    /* ch13: VR2 input */
  /* place board in horizon */
  /*
  * x=0x800
  * y=0x800
  * z=0xb41
  */
  /* move the VR2 at middle */
  /* VR2 value = 0xfff/2 = 0x7ff */
  uint16_t low[16] = {0x700,0x700,0xb00,0x500};
  uint16 t up[16] = {0x900,0x900,0xc00,0xB00};
  ad test info t ADTest Info = {AD UINT0, ch, sizeof(ch)/sizeof(uint8 t), low, up};
  SystemInit();
  /* use hardware CRC16 to calculate expected crc first,
      then verify if the CRC code calculated by software is same with expected crc */
  hw_crc = IEC60730_HardwareCRC16Gen(a, sizeof(a));
  if(IEC60730 TEST NORMAL != IEC60730 SoftwareCRC16Test(a, sizeof(a), hw crc)){while(1);};
  /* use software CRC16 to calculate expected crc first,
      then verify if the CRC code calculated by hardware is same with expected crc */
  sw crc = IEC60730 SoftwareCRC16Gen(a, sizeof(a));
  if(IEC60730_TEST_NORMAL != IEC60730_HardwareCRC16Test(a, sizeof(a), sw_crc)) {while(1);};
  /* GPIO output test
  * test P32-P39 (control LED1-LED8)
  */
#ifdef MB9BF506R SK
  for(bit=BIT NUM 2;bit<BIT NUM 9;bit++)
  {
      if(IEC60730_TEST_NORMAL != IEC60730_GPIOOutputTest(PORT_NUM_3, bit, TEST_PIN_LOW)
      || IEC60730_TEST_NORMAL != IEC60730_GPIOOutputTest(PORT_NUM_3, bit, TEST_PIN_HIGH))
      {
          while(1);
      }
  }
#endif
  /* GPIO input test
  * test P30(LEFT)
  * test P31(RIGHT)
```

```
   * test P40(UP)
   * test P41(DOWN)
   */
#ifdef MB9BF506R SK
   if(IEC60730 TEST NORMAL != IEC60730 GPIOInputTest(PORT NUM 3, BIT NUM 0, 1)){while(1);};
   if(IEC60730 TEST NORMAL != IEC60730 GPIOInputTest(PORT NUM 3, BIT NUM 1, 1)){while(1);};
   if(IEC60730 TEST NORMAL != IEC60730 GPIOInputTest(PORT NUM 4, BIT NUM 0, 1)){while(1);};
   if(IEC60730_TEST_NORMAL != IEC60730_GPIOInputTest(PORT_NUM_4, BIT_NUM_1, 1)){while(1);};
#endif
   /* AD test
   * check if ch7,ch8,ch9,ch13 input is in expected range.
   */
#ifdef MB9BF506R_SK
   if(IEC60730_TEST_NORMAL != IEC60730_ADTest(ADTest_Info)){while(1);};
#endif
   /*Init LEDs*/
   LED Init();
   /*Init Buttons*/
   Button_Init();
   /*Init LCD Pins*/
   HD44780 IO Init();
   /* Interrupt test initialization */
IEC60730 IntTestInit(IntTest Freq,IntTest FreqLower,IntTest FreqUpper,IntTest FreqInit,sizeof(I
ntTest Freq)/s
izeof(uint32_t));
   /* clock test initialization
   * test CPU clock by checking if the 25ms interval time is set for dual timer,
   * the occurrence frequency of dual times is about 40 in a 1s interval(produced by watch counter)
   * 1 cycle time = (1/80MHz). Assume it takes 10 cycles to implement main loop.
   * So minimum
   */
   IEC60730_ClkInit(35,45,10000000);
   /* init watch counter */
   WTC Init();
   /* init dual timer */
   DT Init();
   /* init 4 base timers */
   BT_Init();
   /*Power up LCD*/
   if(HD44780 OK != HD44780 PowerUpInit())
   {
       return 1;
   }
#ifdef MB9BF506R_SK
   /* LCD display */
   HD44780 StrShow(1,1,(const HD44780 STRING DEF *)"IEC60730 Class B");
   HD44780 StrShow(3,2,(const HD44780 STRING DEF *)"Self-Test Lib");
#endif
   /*Main Loop*/
   while(1)
   { /*Wait for timer tick*/
       if(Tmr1Tick)
       { /*Clear timer tick flag*/
           Tmr1Tick = 0;
           /*update counter*/
           cntr--;
           /*leds update*/
           LED PDOR &= ~LED MASK;
           LED PDOR |= ((cntr) & LED MASK);
       }
       if(IEC60730_TEST_NORMAL != IEC60730_ClkMonInMainloop()) /* monitor watch counter
       {
           while(1);
       }
   }
}
```

- デュアルタイマ ISR

図 25. デュアルタイマ ISR

```
void DT QDU IRQHandler(void)
{
  Tmr1Tick = 1;
  FM3_DTIM->TIMER1INTCLR= 1;
  /* count the clock tick */
  IEC60730 ClkCnt();
  /* implement interrupt test */
  if(IEC60730 TEST NORMAL != IEC60730 IntTest())
  {
      while(1);
  }
}
```

- ウォッチカウンタ ISR

図 26. ウォッチカウンタ ISR

```
void CLK IRQHandler(void)
{
  if(bFM3_INTREQ_IRQ24MON_WCINT)
  {
      FM3 WC->WCCR &= 0xFE; /* Clear interrupt flag */
      /* implement clock test */
      if(IEC60730 TEST NORMAL != IEC60730 ClkTest())
      {
          while(1);
      }
  }
}
```

- リロードタイマ ISR

図 27. リロードタイマ ISR

```
void BTIM_IRQHandler(void)
{
  if(FM3 BT0 RT->STC&0x01)
  {
      FM3 BT0 RT->STC = FM3 BT0 RT->STC & 0xFE;
      IEC60730 IntCntPro(0);          /* count frequency value for interrupt 0 */
  }
  else if(FM3_BT1_RT->STC&0x01)
  {
      FM3 BT1 RT->STC = FM3 BT1 RT->STC & 0xFE;
      IEC60730 IntCntPro(1);         /* count frequency value for interrupt 1 */
  }
  else if(FM3_BT2_RT->STC&0x01)
  {
      FM3 BT2 RT->STC = FM3 BT2 RT->STC & 0xFE;
      IEC60730 IntCntPro(2);         /* count frequency value for interrupt 2 */
  }
  else if(FM3_BT3_RT->STC&0x01)
  {
      FM3 BT3 RT->STC = FM3 BT3 RT->STC & 0xFE;
      IEC60730 IntCntPro(3);         /* count frequency value for interrupt 3 */
  }
}
```

6 STL の API の性能

表 6. STL API の性能

API 名称	実行時間 (サイクル)	使用スタック (バイト)	使用 ROM (バイト)	使用 RAM (バイト) (グローバル変数)
iec60730_pc_test	158	0	136	0
iec60730_reg_test	498	0	642	0
IEC60730_IntTestInit	191 (4 割込み)	8	58	0
IEC60730_IntCntPro	47	0	36	0
IEC60730_IntTest	339 (4 割込み)	0	74	20
IEC60730_ClkCnt	54	0	32	0
IEC60730_ClkTest	92	0	80	29
IEC60730_ClkMonInMainloop	79	8	68	0
IEC60730_ClkTestReset	60	0	30	0
IEC60730_InitCSV	280	8	224	0
IEC60730_CheckCSVStat	30	0	56	0
IEC60730_HardwareCRC16Gen	320 (10 バイトデータ)	4	98	0
IEC60730_HardwareCRC16Test	348 (10 バイトデータ)	8	20	0
IEC60730_SoftwareCRC16Gen	326 (10 バイトデータ)	4	52+ 512 (CRC 表)	0
IEC60730_SoftwareCRC16Test	354 (10 バイトデータ)	8	22	0
IEC60730_HardwareCRC32Gen	320 (10 バイトデータ)	4	92	0
IEC60730_HardwareCRC32Test	348 (10 バイトデータ)	8	20	0
IEC60730_SoftwareCRC32Gen	244 (10 バイトデータ)	4	42+ 1024 (CRC 表)	0
IEC60730_SoftwareCRC32Test	266 (10 バイトデータ)	8	22	0
iec60730_ram_test	262 (16 バイトデータ)	0	84	0
IEC60730_GPIOOutputTest	264	24	254	0
IEC60730_GPIOInputTest	265	28	266	0
IEC60730_ADTest	1996 (4 チャネル)	80	1064	0

注意:

1. コードの実行サイクルは通常の動作状態でテストします。
2. この STL の ROM サイズは 3528 バイトです。(フラッシュテストには CRC16, クロックテストにはウォッチカウンタを使用します)。

7 参照文書

[1]. IEC 60730-1 Reference Manual Edition3.2, 2007

[2]. ARMv7-M Architecture Application Level Reference Manual, 2008

[3]. Cortex-M3 r2p0 Technical Reference Manual, 2008

[4]. MB98bF500a- DS706-00021-1v0-E (MB9B500 Data Sheet)

[5]. MB9Bxxx-MN706-00002-1v0-E (MB9Axxx/MB9Bxxx Series Periphery Manual)

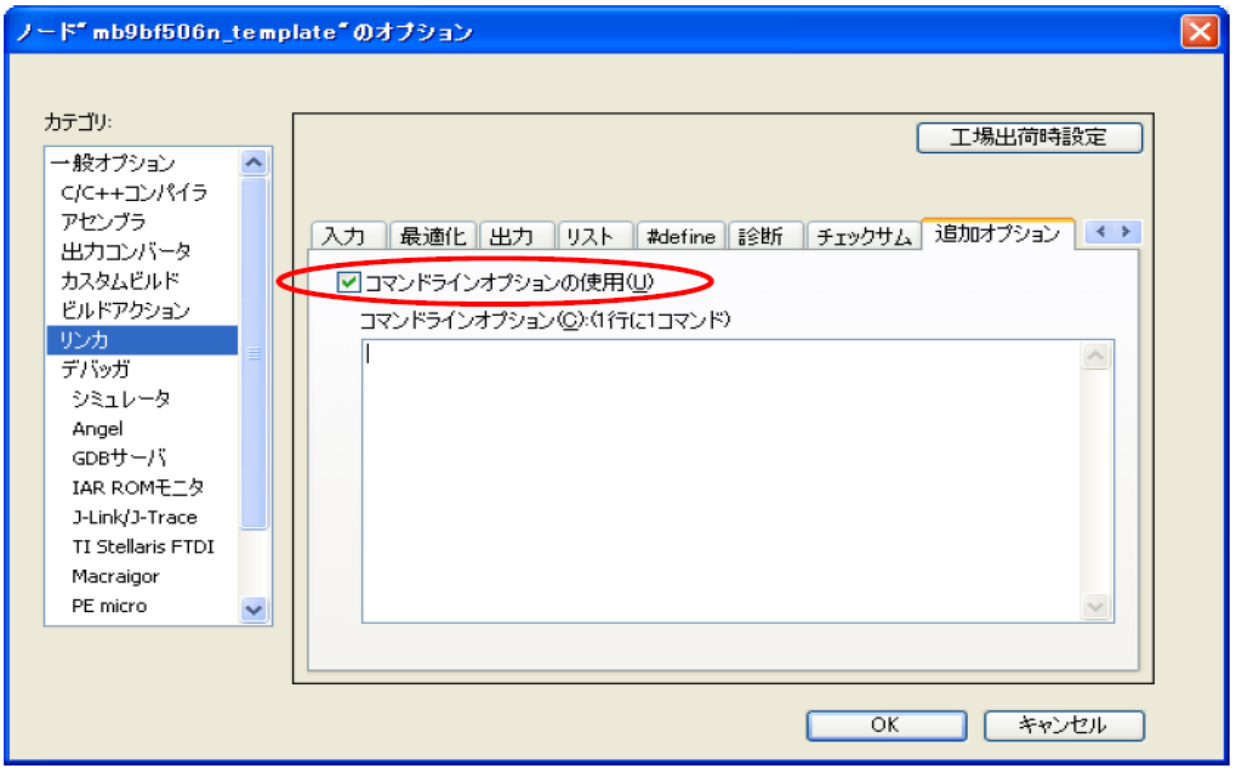
8 付録

8.1 フラッシュの CRC コード作成方法

4.5 不揮発性メモリのテストで使用するデータ/プログラムの CRC コードの生成方法について、IAR Embedded Workbench の例を以下に説明します。詳細は IAR のマニュアルを参照してください。

8.1.1 コマンドラインの起動

「プロジェクト」→「オプション」→「リンカ」→「追加オプション」タブを選択し、"コマンドラインオプションの使用" にチェックを入れます。



8.1.2 コマンドの入力

--place_holder コマンド

CRC コードを作成して ROM にセクションを生成するために、"--place_holder" オプションを使用します。セクションのサイズを 4byte, アライメントを 1 に設定するため下記のコマンドを設定します。

--place_holder __checksum,4,.checksum,1

--fill コマンド

CRC コードを作成するためには、対象領域の未使用領域に対して任意の値をフィルする必要があります。そのために、"--fill" コマンドを使用します。対象領域 0x00000000-0x00003FFF にフィル値 0xFF を設定するコマンドは以下の設定となります。

--fill 0xFF;0x0000-0x3FFF

対象領域 0x00000000-0x00003FFF, 0x5000-0x5FFF, 0x6500-0x6FFF にフィル値 0xFF を設定するコマンドは以下の設定となります。

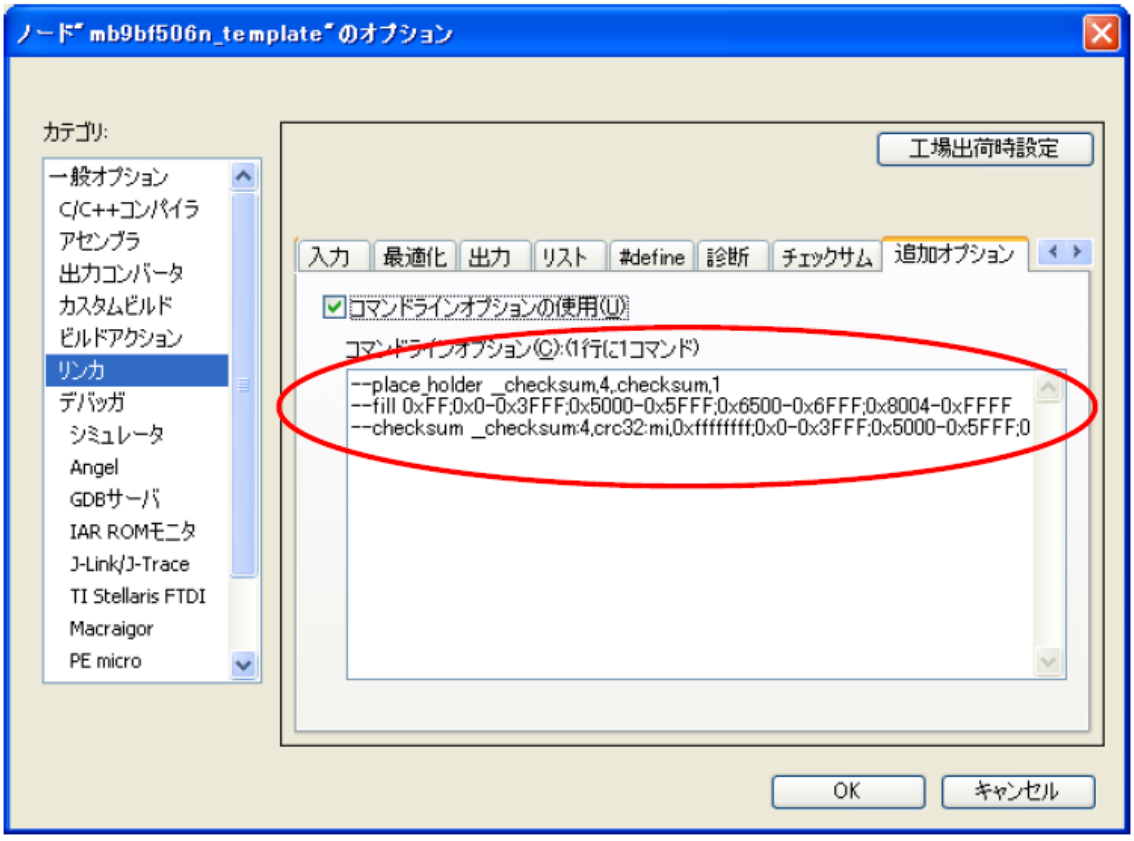
--fill 0xFF;0x0-0x3FFF;0x5000-0x5FFF;0x6500-0x6FFF

--checksum コマンド

使用する CRC のアルゴリズムを設定します。CRC コードを格納するシンボル名を__checksum, サイズを 4byte, アルゴリズムを CRC32, 計算を LSB first, CRC コードを 0xFFFFFFFF で初期化, 対象領域 0x00000000-0x00003FFF, 0x5000-0x5FFF, 0x6500-0x6FFF の設定をするコマンドは以下のようになります。

--checksum __checksum:4,crc32:mi,0xffffffff;0x0-0x3FFF;0x5000-0x5FFF;0x6500-0x6FFF

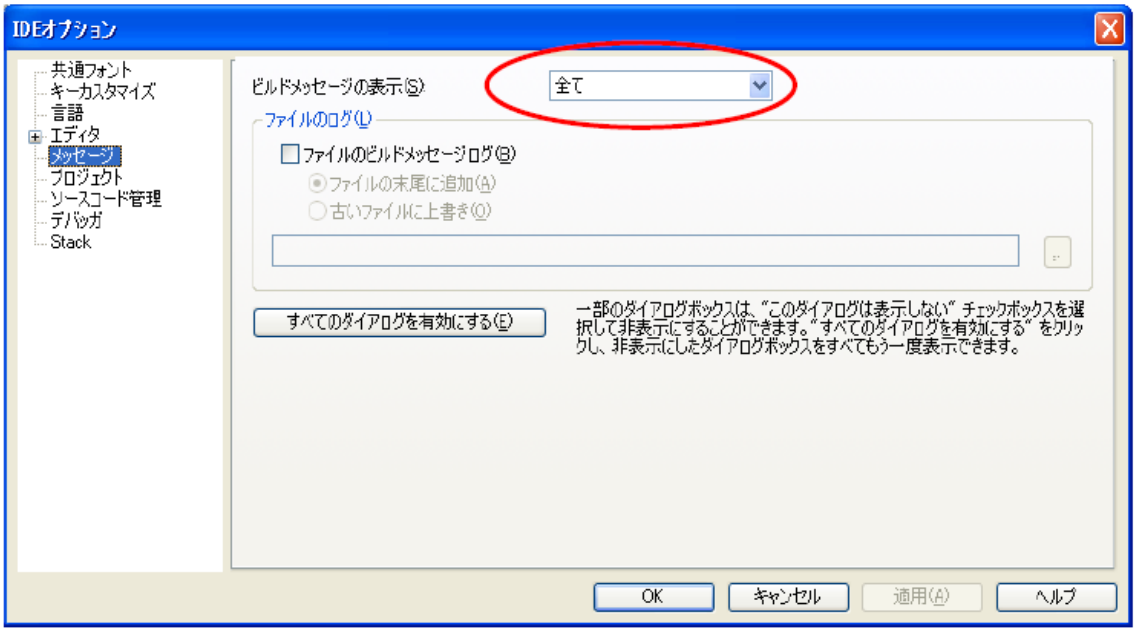
1., 2., 3. を入力した後、"OK" を押してウィンドウを閉じます。



8.1.3 メッセージウィンドウ表示内容の設定

メイク時のメッセージをメッセージウィンドウへ表示するように設定します。

「ツール」→「オプション」→「メッセージ」を選択します。"ビルドメッセージの表示(S):" のコンボボックスで "全て" を選択します。右下の "OK" または "適用" ボタンを押します。



8.1.4 リンカ設定ファイルの設定

生成した CRC コードをフラッシュメモリに格納するため、リンカ設定ファイルに設定を追加します。debug モードの場合は ”mb9bf506_ram.icf” ファイル、release モードの場合は ”mb9bf506.icf” ファイルです。0x8000 番地に CRC コードを格納する場合は以下の設定を追加します。

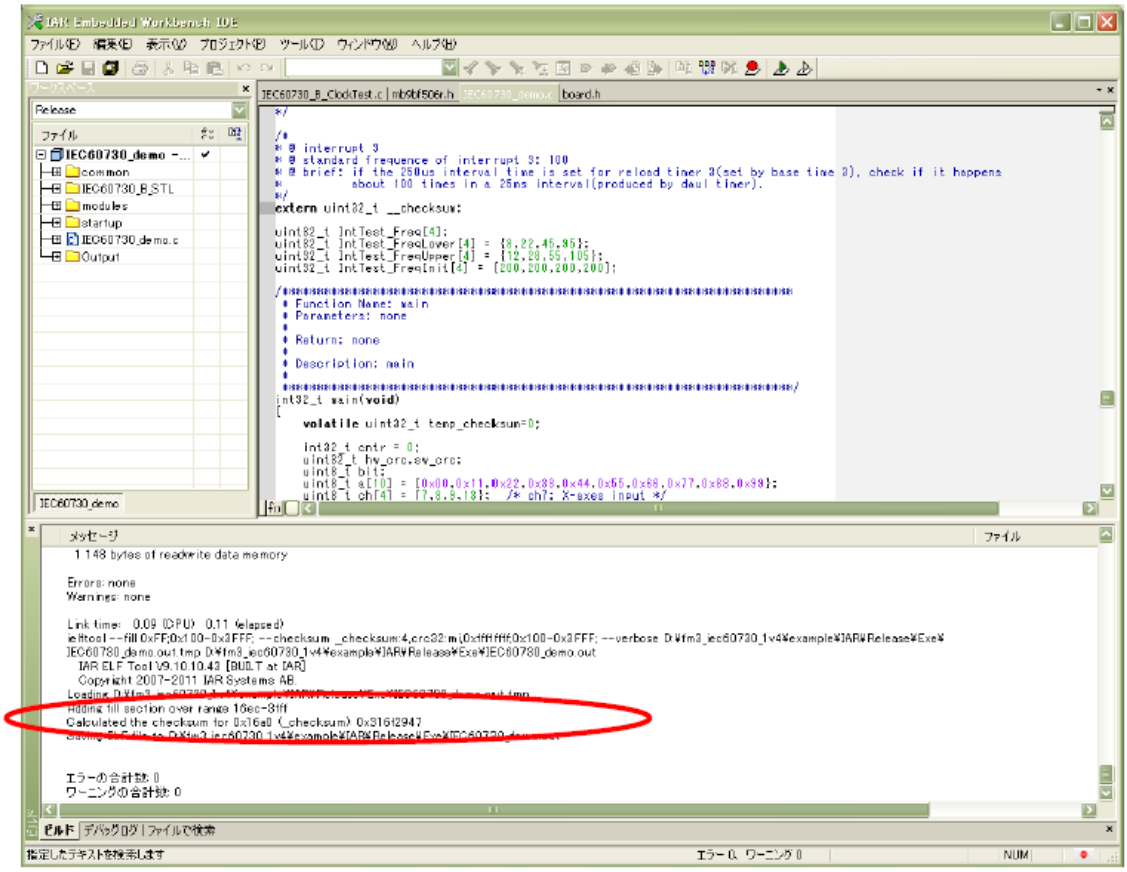
```
define symbol __ICFEDIT_checksum_start__ = 0x00008000;

define block CHECKSUM {ro section .checksum};

define symbol __ICFEDIT_checksum_start__ = 0x00008000;
```

8.1.5 CRC コード作成

メイク (ビルド) を実行し、CRC コードが作成されたことを確認します。



9 改訂履歴

文書名: AN204348 - FM3 IEC60730 Class B 準拠 セルフテストライブラリ

文書番号: 002-04349

版	ECN 番号	変更者	発行日	変更内容
**	-	MIQI	11/01/2012	サイプレスとしてスパンション アプリケーションノート AN706-00037-2v0-J をドキュメントコード 002-04349 に登録しました。 本版の内容およびフォーマットに変更はありません。
*A	5705385	MIQI	04/21/2017	これは英語版の 002-04348 Rev. *A を翻訳した日本語版です。

セールス, ソリューションおよび法律情報

ワールドワイドな販売と設計サポート

サイプレスは、事業所、ソリューション センター、メーカー代理店、および販売代理店の世界的なネットワークを保持しています。お客様の最寄りのオフィスについては、サイプレスのロケーション ページをご覧ください。

製品

ARM® Cortex® Microcontrollers	cypress.com/arm
車載用	cypress.com/automotive
クロック&バッファ	cypress.com/clocks
インターフェース	cypress.com/interface
IoT (モノのインターネット)	cypress.com/iot
メモリ	cypress.com/memory
マイクロコントローラ	cypress.com/mcu
PSoC	cypress.com/psoc
電源用 IC	cypress.com/pmic
タッチ センシング	cypress.com/touch
USB コントローラー	cypress.com/usb
ワイヤレス／RF	cypress.com/wireless

PSoC® ソリューション

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

サイプレス開発者コミュニティ

フォーラム | WICED IOT Forums | Projects | ビデオ | ブログ | トレーニング | Components

テクニカルサポート

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

© Cypress Semiconductor Corporation, 2012-2017. 本書面は、Cypress Semiconductor Corporation 及び Spansion LLC を含むその子会社（以下、「Cypress」という。）に帰属する財産である。本書面（本書面に含まれ又は言及されているあらゆるソフトウェア又はファームウェア（以下、「本ソフトウェア」という。）を含む）は、アメリカ合衆国及び世界のその他の国における知的財産法令及び条約に基づき、Cypress が所有する。Cypress はこれらの法令及び条約に基づく全ての権利を留保し、また、本段落で特に記載されているものを除き、Cypress の特許権、著作権、商標権又はその他の知的財産権のライセンスを一切許諾していない。本ソフトウェアにライセンス契約書が伴っておらず、かつ、あなたが Cypress との間で別途本ソフトウェアの使用方法を定める書面による合意をしていない場合、Cypress は、あなたに対して、（1）本ソフトウェアの著作権に基づき、（a）ソースコード形式で提供されている本ソフトウェアについて、Cypress ハードウェア製品と共に用いるためにのみ、組織内部でのみ、本ソフトウェアの修正及び複製を行うこと、並びに（b）Cypress のハードウェア製品ユニットに用いるためにのみ、（直接又は再販売者及び販売代理店を介して間接のいずれかで）エンドユーザーに対して、バイナリーコード形式で本ソフトウェアを外部に配布すること、並びに（2）本ソフトウェア（Cypress により提供され、修正がなされていないもの）に抵触する Cypress の特許権のクレームに基づき、Cypress ハードウェア製品と共に用いるためにのみ、本ソフトウェアの作成、利用、配布及び輸入を行うことについての非独占的で譲渡不能な一身専属的ライセンス（サブライセンスの権利を除く）を付与する。本ソフトウェアのその他の使用、複製、修正、変換又はコンパイルを禁止する。

適用される法律により許される範囲内で、Cypress は、本書面又はいかなる本ソフトウェアに関しても、明示又は黙示をとわず、いかなる保証（商品性及び特定の目的への適合性の黙示の保証を含むがこれらに限られない）も行わない。適用される法律により許される範囲内で、Cypress は、別途通知することなく、本書面を変更する権利を留保する。Cypress は、本書面に記載のあるいかなる製品又は回路の適用又は使用から生じる一切の責任を負わない。本書面で提供されたあらゆる情報（あらゆるサンプルデザイン情報又はプログラムコードを含む）は、参照目的のためのみに提供されたものである。この情報で構成するあらゆるアプリケーション及びその結果としてのあらゆる製品の機能性及び安全性を適切に設計し、プログラムし、かつテストすることは、本書面のユーザーの責任において行われるものとする。Cypress 製品は、兵器、兵器システム、原子力施設、生命維持装置若しくは生命維持システム、蘇生用の設備及び外科的移植を含むその他の医療機器若しくは医療システム、汚染管理若しくは有害物質管理の運用のために設計され若しくは意図されたシステムの重要な構成部分として用いるため、又はシステムの不具合が人身傷害、死亡若しくは物的損害を生じさせることになるその他の使用（以下、「本目的外使用」という。）のためには、設計、意図又は承認されていない。重要な構成部分とは、装置又はシステムのその構成部分の不具合が、その装置若しくはシステムの不具合を生じさせるか又はその安全性若しくは実効性に影響すると合理的に予想できる、機器又はシステムのあらゆる構成部分をいう。Cypress 製品のあらゆる本目的外使用から生じ、若しくは本目的外使用に関連するいかなる請求、損害又はその他の責任についても、Cypress はその全部又は一部をとわず一切の責任を負わず、かつ、あなたは Cypress をそれら一切から免除するものとし、本書により免除する。あなたは、Cypress 製品の本目的外使用から生じ又は本目的外使用に関連するあらゆる請求、費用、損害及びその他の責任（人身傷害又は死亡に基づく請求を含む）から Cypress を免責補償する。

Cypress、Cypress のロゴ、Spansion、Spansion のロゴ及びこれらの組み合わせ、WICED、PSoC、CapsSense、EZ-USB、F-RAM、及び Traveo は、米国及びその他の国における Cypress の商標又は登録商標である。Cypress の商標のより完全なリストは、cypress.com を参照のこと。その他の名称及びブランドは、それぞれの権利者の財産として権利主張がなされている可能性がある。